Manejo del software Ginga para el desarrollo de aplicaciones interactivas para televisión digital, basado en el estándar Brasileño ISDB-Tb

UNIVERSIDAD POLITÉCNICA SALESIANA.
SEDE CUENCA.

CARRERA DE INGENIERIA ELECTRONICA.

Trabajo de grado previo a la obtención
del Título de Ingeniero Electrónico.

TEMA:

Manejo del software Ginga para el desarrollo de aplicaciones interactivas para televisión digital, basado en el estándar Brasileño ISDB-Tb.

AUTORES:

Pablo Teodoro Galabay Toalongo.
Freddy Rafael Vivar Espinoza.

DIRECTOR:

Ingeniero Edgar Ochoa Figueroa.

Cuenca, Junio 2012

Breve reseña de los autores e información de contacto:

Pablo Teodoro Galabay Toalongo.
Egresado de la carrera de ingeniería electrónica
Universidad Politécnica Salesiana
p.galabay@gmail.com

FreddyRafael Vivar Espinoza.
Egresado de la carrera de ingeniería electrónica
Universidad Politécnica Salesiana
freddyraphv@hotmail.com

Dirigido por:

Ing. Edgar Ochoa Figueroa.
Magister en Gestión de Telecomunicaciones
Ingeniero Eléctrico
Docente de la Universidad Politécnica Salesiana
Carrera de Ingeniería Electrónica
eochoafigueroa@gmail.com

Certificación.

En calidad de DIRECTOR DE LA TESIS **"Manejo del software Ginga para el desarrollo de aplicaciones interactivas para televisión digital, basado en el estándar Brasileño ISDB-Tb."**, elaborada por Pablo Galabay y Freddy Vivar, declaro y certifico la aprobación del presente trabajo de tesis basándose en la supervisión y revisión de su contenido.

Cuenca, Junio del 2012

Ingeniero Edgar Ochoa Figueroa.

DIRECTOR DE TESIS

Responsabilidad y Autoría.

Los autores del trabajo de tesis titulado **"Manejo del software Ginga para el desarrollo de aplicaciones interactivas para televisión digital, basado en el estándar Brasileño ISDB-Tb"** Pablo Teodoro Galabay Toalongo y Freddy Rafael Vivar Espinoza, autorizan a la Universidad Politécnica Salesiana la libre difusión de este documento exclusivamente para fines académicos o investigativos por cualquier medio. El análisis de los conceptos y las ideas vertidas en la presente tesis son de total responsabilidad de los autores.

Cuenca, Junio del 2012

Pablo Teodoro Galabay Toalongo.
AUTOR.

Freddy Rafael Vivar Espinoza
AUTOR.

Nomenclatura

ABNT: Associação Brasileira de Normas Técnicas

ACAP: Advanced Common Application Platform

API: Application Programming Interface

ARIB: Association of Radio Industries and Businesses

ATSC: Advanced Television Systems Committee

BML: Broadcast Markup Language

DAE: Módulos de Ambiente de Aplicación Declarativo

DASE: Estándar de los EE.UU para la capa de middleware de set-top-boxes de TV digital

DAVIC: Digital Audio Visual Council

DCDE: Declarative Content Decoding Engine

DSM-CC: Digital Storage Media, Command and Control

DTMB: Digital Terrestrial Multimedia Broadcasting

DVB-J: Digital Vídeo Broadcasting - Java

DVB: Digital Vídeo Broadcasting

EPL: Eclipse Public License

GEM: Globally Executable MHP

HAVi: Home Audio Video Interoperability

HDTV: High Definition Televisión

HTML: HyperText Markup Language - Lenguaje de marcado de hipertexto

IANA: Internet Assigned Numbers Authority

IDE: Integrated Development Environment

IDREF: Referencia a un ID definido en otros atributos

IP: Internet Protocol

ISDB-T: Terrestrial Integrated Services Digital Broadcasting

ISDB-Tb: Sistema Brasileño de Televisión Digital - Terrestre

JDK: Java Development Kit

JVM: Máquina Virtual de Java

LUA: Lenguaje de programación imperativo, estructurado y bastante ligero, que fue diseñado como un lenguaje interpretado con una semántica extensible.

MHP: Multimedia Home Platform

MIME: Multipurpose Internet Mail Extensions

MPEG: Moving Pictures Experts Group

NCL: Nested Context Language

OCAP: Open Cable Application Platform

PAE: Ambiente de Aplicación de Procedimiento

PCDE: Procedural Content Execution Engine

SDTV: Televisión de definición estándar

SKY: Sistema de televisión satelital que actualmente opera en Brasil

SMIL: Linguagem de Integração de Multimédia Sincronizada

SSH: Secure Shell

STB: Set-Top-Box

STD-23: Application Execution Engine Platform for Digital Broadcasting

STD-B24: Data Coding and Transmission Specification for Digital Broadcasting

TV: Televisión

UFPb: Universidad Federal de Paraiba

URI: Uniform resource identifier

VSTB: Virtual SetTopBox

W3C: World Wide Web Consortium

XHTML: eXtensible HyperText Markup Language

XML: Extensible Markup Language

Índice general

Índice de figuras

Índice de cuadros

Dedicatoria.

El presente trabajo de tesis se la dedico a mi familia que en todo momento me brindaron su apoyo, de manera especial a mis padres por ayudarme a cumplir mis objetivos como persona y estudiante. A ti padre por ser mi ejemplo de superación; a ti madre por haberme dado la vida, por tus consejos y amor; a mis hermanos y a mi enamorada por estar siempre dándome aliento.

Pablo Teodoro Galabay Toalongo.

Dedico el presente trabajo a mis padres, de manera muy especial a mi madre que fue mi apoyo incondicional en todo momento, a mi padre por darme un ejemplo de superación, a mis hermanos y a todos mis amigos que de una manera u otra siempre me estuvieron apoyando para alcanzar llegar mis metas.

Principalmente quiero dedicar este trabajo de tesis a mi gran amigo Diego Sánchez por haberme acompañado durante todos los años de mi vida universitaria, pero que no pudo verme culminar Mis estudios universitarios porque Dios lo llamo a su lado.

Freddy Rafael Vivar Espinoza.

Agradecimiento.

Agradezco a Dios por darme la vida para terminar mi carrera universitaria, a mis padres y hermanos por la confianza y apoyo incondicional, a mi director Ing. Edgar Ochoa Figueroa, Mgt por haberme guiado durante todo el trabajo de tesis.

A todas aquellas personas quienes compartieron sus conocimientos para hacer posible la culminación de este trabajo de tesis

Pablo Teodoro Galabay Toalongo.

Agradezco a Dios por haberme dado la sabiduría y las fuerzas para superar todos los obstáculos que se presentaron a lo largo de toda mi carrera, a mi madre y a mi padre por brindarme su apoyo incondicional, a mis amigos que de una u otra manera me ayudaron durante todo este trajinar.

Freddy Rafael Vivar Espinoza.

Resumen.

El presente trabajo de tesis previo a la obtención del título de ingeniería electrónica, centra su estudio teórico en el manejo del middleware Ginga mediante la utilización de su motor de presentación denominado Ginga NCL, cuyo lenguaje de programación se denomina NCL. En el Ecuador bajo recomendación de la SUPERTEL se decidió adoptar la norma japonesa-brasileña ISDB-Tb, para que opere como el estándar de TV Digital terrestre para desarrollo de aplicaciones interactivas. Para lo cual iniciamos realizando una introducción a la TV Digital interactiva, los estándares de middlewares desarrollados para que los países en función de sus necesidades sociales y geográficas puedan optar por uno u otro, describiremos la arquitectura de referencia del middleware Ginga propuestos por la Universidad Católica de Rio de Janeiro y la Universidad Federal de Paraíba, que lo han dividido en dos subsistemas principales, el primero el de entorno o presentación Ginga NCL y el segundo de la parte Java, llamado Ginga-J que en conjunto permiten la presentación de aplicaciones interactivas.

Ginga-J, está compuesta por un conjunto de APIs y su máquina virtual Java que permiten la implementación de aplicaciones interactivas, su arquitectura diferencia entre entidades de hardware o de recursos y software del sistema, se basan en un conjunto de tres APIs denominados: Verde, Amarillo y Azul que han sido desarrollados para satisfacer las necesidades especificas de Brasil y a su vez para que puedan mantener compatibilidad con la norma GEM.

En el entorno de presentación Ginga-NCL describiremos los elementos que forman parte de éste, y del lenguaje de script Lua que es utilizado por el módulo Ginga NCL para implementar objetos imperativos en documentos NCL, además detallaremos la estructura de un documento NCL como es el encabezado, la sección head, donde se definen las regiones, los descriptores, los conectores y las reglas utilizadas por el programa, el cuerpo del programa, los puertos de entrada y la finalización del programa. Se mencionan las herramientas de desarrollo como son el Composer y el Eclipse NCL y la de presentación como son el Emulador Ginga NCL y el Set Top Box Virtual. Y finalmente concluiremos nuestro trabajo generando un manual de programación del middleware Ginga-NCL que sirva como precedente para fututas investigaciones y trabajos de todos los que estén inmersos en este campo.

Introducción.

Las exigencias de la sociedad mundial, el avance de la ciencia y tecnología, ha incursionado en el sistema de la TV, haciendo que está sufra un cambio significativo, la migración de la TV analógica convencional hacia un sistema completamente digital, cuyo desarrollo ha ido evolucionando con el transcurso de los años, surgiendo diversos estándares para la TV Digital Terrestre, cada uno con características especiales enfocados a las necesidades y condiciones geográficas acorde a los países donde han sido creados. La TV Digital trae muchas innovaciones con respecto a la TV analógica, a más de la transmisión del video en alta definición y la mejora en el aprovechamiento del espectro radioeléctrico, brinda al usuario opciones y aplicaciones de interactividad, pudiendo esté realizar consultas bancarias, voto directo en concursos y la navegación por el internet, etc desde su control remoto. En virtud de lo anotado ofrecemos información básica de lo que es el sistema de TV Digital Terrestre, realizaremos el manejo del software Ginga para el desarrollo de aplicaciones interactivas para televisión digital, basada en el estándar Brasileño ISDB-Tb llegando a generar un manual de manejo de Ginga NCL.

El presente trabajo se encuentra divido en cuatro capítulos, los mismos que se detallan a continuación:

El capitulo uno presenta una breve introducción a la TV Digital interactiva, así como los middlewares desarrollados para que los países puedan optar por uno u otro de acuerdo a sus necesidades sociales y geográficas realizando una breve introducción a la arquitectura de referencia del middleware Ginga, las cuales lo han divido en dos subsistemas principales, el primero el de entorno de presentación Ginga NCL y el segundo de la parte Java, llamado Ginga J, que en conjunto permiten la presentación de aplicaciones interactivas.

El capitulo dos describe las aplicaciones de procedimiento (Ginga J), que está compuesto por un un conjunto de APIs y su máquina virtual Java que permiten la implementación de aplicaciones de TV Digital. Su arquitectura distingue entre entidades de hardware o de recursos, software del sistema. Ginga-J se basa en un conjunto de de APIs denominados: Verde, Amarillo y Azul que se han desarrollarlo para satisfacer las necesidades especificas de Brasil y a su vez para que puedan tener compatibilidad con el API de la norma GEM.

En el capitulo tres se presenta el estudio de las aplicaciones declarativas (Ginga NCL), desarrollado por la PUC-Rio, así como los elementos que forman parte de esté para en conjunto permitir la elaboración de aplicaciones interactivas bajo el motor de presentación Ginga –NCL. Se describe Lua que es un lenguaje de script adoptado por el modulo Ginga NCL para implementar objetos imperativos en documentos NCL. Además se detalla la estructura de un documento NCL como es el encabezado, la sección head, donde se definen las regiones, los descriptores, los conectores y las reglas utilizadas por el programa, el cuerpo del programa, los puertos de entrada y la finalización del programa. Se mencionan las herramientas de desarrollo como son el Composer y el Eclipse NCL y la de presentación como son el Emulador Ginga NCL y el Set Top Box Virtual. NCL puede describir la presentación del contenido en varios dispositivos, y permite la producción de este en vivo, de una manera no lineal. Ginga-NCL es el único estándar internacional de middleware tanto para IPTV como TDT.

Finalmente en el capitulo cuatro se presenta el resultado de este estudio, mediante la generación de manual de programación del middleware Ginga NCL, detallando paso a paso todos los requerimientos necesarios para empezar con el desarrollo de una aplicación interactiva sencilla hasta llegar a una aplicación compleja.

La información expresada en el presente trabajo es el resultado de una investigación profunda y documentada, cuyo propósito es tener un sustento teórico-práctico, que sirva como base para consultas y el desempeño profesional de quienes estamos inmersos en esta profesión.

Capítulo 1

Televisión Digital.

1.1. Televisión Digital Interactiva.

Un gran número de equipos que se utilizan actualmente para la creación de programas televisivos son digitales, a pesar de que en nuestros hogares todavía recibimos señales analógicas de televisión (ver Figura 1.1), pero su calidad y disponibilidad no serian posibles sin una producción y distribución digital. [1]

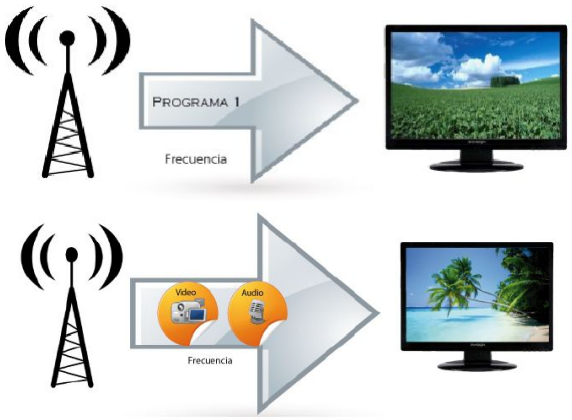


Figura 1.1: Transmisión Analógica
Fuente: http://tvd.lifia.info.unlp.edu.ar

La TV Digital se resume a la conversión de la señal de TV analógica a un formato binario, que puede ser emitido de diferentes maneras; por satélite, terrestre o cable, para posteriormente ser decodificado por el propio televisor a través de un set-top-box [2]

El set-top-box es un hardware o dispositivo externo, que permite, que un televisor convencional, (TV analógica), como los que tenemos en los hogares pueda reproducir las señales televisivas producidas con tecnología digital, a los cuales se les dota de un middleware, para que puedan soportar características de interactividad con los usuarios y por ende adicionen muchas funcionalidades a estos. Conforme avancemos con el desarrollo de esta tesis se describirá a los middlewares para televisión digital.

La televisión digital interactiva, funciona a partir de la difusión de la televisión directa, junto con datos y servicios interactivos, consiente llegar a conseguir una mejor calidad HDTV en la recepción y visualización de las señales en dispositivos

fijos, móviles y portátiles, además se logra una mayor eficiencia en el aprovechamiento del espectro radioeléctrico, lo que conduce a la posibilidad de ofrecer más canales al usuario (ver Figura 1.2); un punto en contra de este tipo de televisión es que al digitalizar la señal de televisión se produce una cantidad de bits enorme, que sin la utilización de la compresión, harían muy difícil su transporte y almacenamiento.

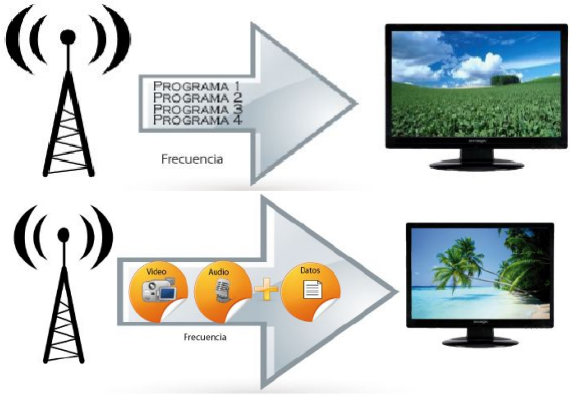


Figura 1.2: Transmisión Digital
Fuente: http://tvd.lifia.info.unlp.edu.ar

En nuestro país, luego de haberse realizado las respectivas pruebas técnicas por parte la Superintendencia de Telecomunicaciones[1], se adopto oficialmente por el estándar Brasilero ISDB-Tb, que es una modificación del estándar Japonés de TV Digital ISDB-T. Entre las modificaciones están el uso de MPEG-4 en lugar de MPEG-2 para la codificación del vídeo y la aparición del middleware Ginga como plataforma de desarrollo y presentación de los contenidos interactivos [2].

1.1.1. La televisión de alta definición HDTV

La televisión de alta definición es sólo una de las opciones entre los muchos avances tecnológicos que proporciona la plataforma de transmisión digital, el sistema de televisión digital permite la transmisión de programas en resoluciones superiores, aumentando considerablemente la calidad de la imagen de la televisión. Actualmente los espectros de televisión son de formato 4:3, en HDTV el formato es 16:9, esto representa una imagen 33 % mas larga que la anterior. La idea es simular el ambiente de cinema, mejorando el uso da visión periférica, lo que permite una mayor experiencia sensorial. Como se puede observar en la Figura 1.3, se puede observar mucha mas información en el formato 16:9.[3][4]

El comité de normas de televisión avanzada (ATSC) ha establecido las normas voluntarias para la televisión digital, incluida la manera en que sonido y vídeo son codificados y transmitidos, así como proporcionar las directrices para los diferentes niveles de calidad. Esta manera, se crearon 18 formatos para la transmisión de vídeo digital, destacando las siguientes diferencias entre formato de pantalla, resolución y taza de exhibición de cuadros.

Las 18 normas para la televisión digital puede ser organizado como se muestra en el Cuadro 1.1, en donde la gran diferencia está taza de exhibición de cuadros.

Es fácil ver que se establecieron seis tipos HDTV y doce SDTV, en la resolución SDTV sube a 480, mientras que la HDTV va hasta 1080.

[1]SUPERTEL, Televisión Digital Terrestre Ecuador, 2010, http://www02.supertel.gob.ec/tdt-ecuador/

Figura 1.3: Comparación entre aspectos 16:9 y 4:3
Fuente: Televisão de Alta Definição

18 formatos primarios de televisión digital.			
Definición	**Resolución**	**Proporción**	**Ritmo de exhibición de imágenes (i=integrado, p=progresivo)**
HDTV	1920 x 1080	16 : 9	24p, 30p, 60i
HDTV	1280 x 720	16 : 9	24p, 30p, 60p
SDTV	704 x 480	16 : 9	24p, 30p, 60i, 60p
SDTV	704 x 480	4 : 3	24p, 30p, 60i, 60p
SDTV	704 x 480	4 : 3	24p, 30p, 60i, 60p

Cuadro 1.1: Las normas primarias (HDTV, SDTV)

Sin embargo, a pesar de la señal digital es de una calidad superior a la señal analógica, no necesariamente es alta definición. Para ver una imagen de televisión de alta definición y escuchar el sonido Dolby Surround, existe la necesidad de la estación de transmitir una señal de alta definición y al espectador con una televisión, alta definición. Aunque los televisores analógicos son capaces de sintonizar canales del sistema digital utilizando los receptores convertidores Sept-top-box, al convertir la señal digital se pierde la calidad de visualización de la misma.

1.2. Tipos de Interacción

Los niveles de interactividad pueden clasificarse en dos grupos:

- Sin interacción (sólo TV Digital)
- Con interacción : Posibilitan la participación directa del telespectador en el nuevo escenario de la televisión digital. El usuario deja de ser un sujeto pasivo para convertirse en un nuevo sujeto activo e identificado, capaz de interactuar con las aplicaciones y servicios personalizados de la televisión digital, los tipos de interactividad que se pueden ofreser son:
    - *Interacción local:* el usuario interactúa con información, transmitida con cierta periodicidad y almacenada en el receptor pero que no es enviada

a un servidor.

- *Interacción con upload:* envío de datos vía canal de retorno
- *Interacción avanzada:* envío y recepción vía canal de retorno

1.3. Modelos de TV Digital

Los modelos de televisión digital, teniendo en cuenta los medios de transmisión se pueden clasificar en los siguientes:

- Modelo de Televisión Digital por Satélite
- Modelo de Televisión Digital por Cable
- Modelo de Televisión Digital Terrenal
- Modelo de Televisión Digital por Microondas
- Modelo de Televisión Digital IP

1.4. Estándares de Transmisión de Televisión Digital

Los estándares de transmisión digital que están en uso o en proceso de adopción en algunos países, pueden resumirse en cinco:

- El estándar Americano, ATSC
- El estándar Europeo, DVB
- El estándar Japonés, ISDB-T
- El estándar Brasileño SBTVD-Tb
- El estándar DTMB, que es la norma china.

La inclusión de los estándares a nivel mundial dependen de la ubicación geográfica de los países así como de el estándar mas factible según la geografía del país que lo va a implementar ver Figura 1.4.

1.5. Middlewares para Interactividad con Televisión Digital

1.5.1. Middleware

Para la ejecución de aplicaciones interactivas en televisión Digital, es necesario el uso de un terminal de acceso conocido con el nombre de Set-Top-Box; este permite que los usuarios puedan controlar y manejar dichas aplicaciones [2].

Esto es posible gracias a la capa de software implementada en este terminal, permitiendo abstraer la complejidad del hardware, es aquí donde los componentes son los responsables de la ejecución de las aplicaciones, dibujarlas en la pantalla del televisor, gestionar los eventos capturados por ellos y supervisar todas las etapas de su ciclo de vida .

Por lo tanto el middleware es una capa intermedia o APIgenérico, que admite el acceso a las aplicaciones y servicios interactivos siempre que sea de la misma

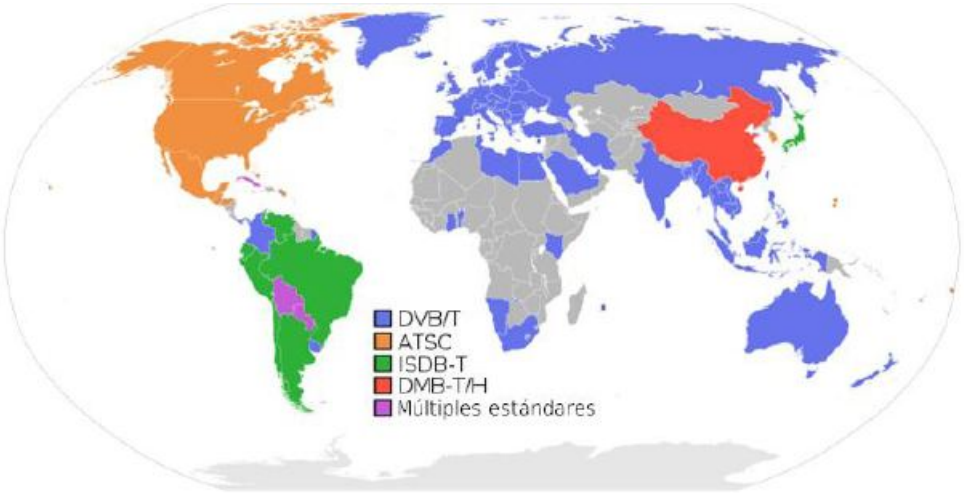


Figura 1.4: Modelos de referencia a nivel mundial
Fuente: http://tvd.lifia.info.unlp.edu.ar

manera, independientemente de la plataforma de hardware o de software donde se están ejecutando como se muestra en la Figura 1.5 .

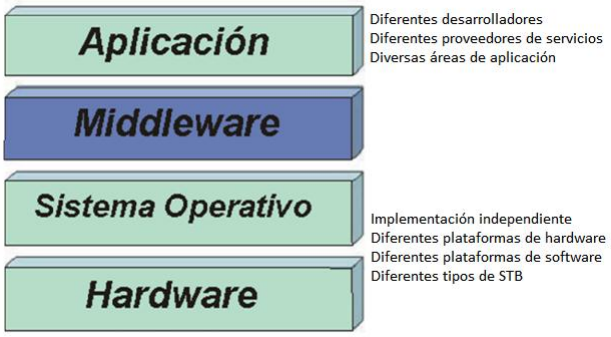


Figura 1.5: Estructura general de un terminal de acceso
Fuente: http://comunidad.ginga.org.ar

Hasta la actualidad, para evitar la proliferación de estándares se han creado varios tipos de middlewares para la Televisión Digital como el americano europeo y japonés, éstos han optado por un middleware estándar en sus receptores digitales. Son middlewares diferentes, aunque con características específicas, los cuales siguen las recomendaciones de la norma GEM(ITU-T J.201) (ITU-T, 2001) [5].

1.5.1.1. Multimedia Home Platform.

Este middleware fue desarrollado para el estándar DVB y es adoptado por Europa. MHPproporciona un entorno para televisión interactiva, independientemente de hardware y software específicos, abiertos e interoperables para los receptores y decodificadores para TV Digital. Su entorno de ejecución está basado en el uso de una máquina virtual de Java y un conjunto de interfaces de programación (API). Estas API permiten a los programas escritos en Java el acceso a los recursos y las instalaciones del receptor digital de una manera estandarizada. A las aplicaciones DVB que usan el API de Java comúnmente se les llama aplicaciones DVB-J o Xlet.

[2]Por lo tanto se pude decir que el núcleo de la norma MHP está basado en una plataforma conocida como DVB-J, que incluye la máquina virtual Java. Una entidad del software de sistema, denominada Gestor de Aplicaciones, la que se encarga de coordinar la ejecución de las aplicaciones y las comunicaciones con su entorno. Un conjunto de paquetes Java que proveen los interfaces entre las aplicaciones, las funciones de un receptor DVB y las redes de comunicación a las que está conectado. De igual forma, también se define en la norma, los formatos de los contenidos que el receptor debe poder gestionar, la torre de protocolos que debe implementar y la señalización adecuada para coordinar el correcto funcionamiento del conjunto, como se ve en la Figura 1.6 [6].

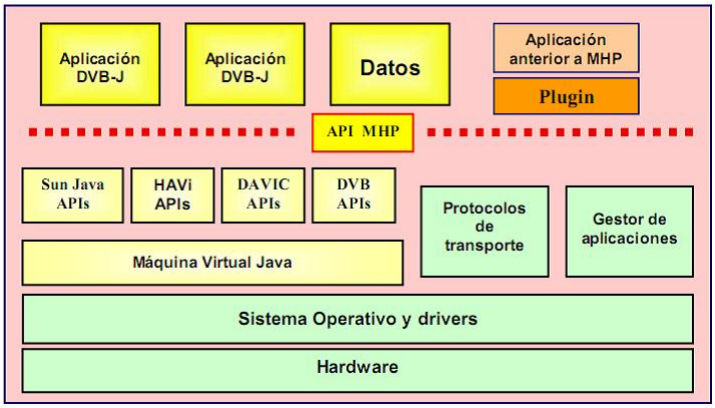


Figura 1.6: Arquitectura de un receptor MHP
Fuente: http://www.mhp.org

Con la aparición de nuevas redes de banda ancha y con el transcurso de los años, se ha realizado mejoras para este middleware, se ha llegado a crear versiones nuevas de MHP, cada una con nuevas características de interactividad para los usuarios. Las versiones generadas son MHP 1.0, MHP 1.1 y MHP 1.2, las innovaciones que presenta cada una de estos son los que se presentan en la Figura 1.7 .

Figura 1.7: Versiones de MHP
Fuente: http://www.mhp.org

Las versiones de MHP, no son comunes entre ellas en lo referente al soporte de aplicaciones; el proyecto DVB introdujo el concepto de perfiles de soporte a

[2]Pagina Ocial de MHP. Multimedia home platform. 2011. URL: http://www.mhp.org.[6]

la aplicación de las normas. Cada perfil hace referencia a áreas de aplicación especificas y define los requisitos necesarios que deben tener los STB. Los perfiles de soporte creados son: Enhanced Broadcast, Interactive Broadcast y el Internet Access.

El middleware MHP, además del uso del API de Java, posibilita usar lenguajes de programación semejantes al HTML (Utilizado para la programación de páginas web), a éste se le denomina DVB-HTML [5].

1.5.1.2. ARIB -Association of Radio Industries Businesses (ARIB)

El middleware de ISDB-T, está estandarizado por la organización japonesa ARIB, entidad que crea y mantiene el estándar de TV Digital ISDB-T .

ARIB ha definido dos estándares principales y cada uno tiene las normativas necesarias para regular, manejar y controlar este proceso, siendo éstas: ARIB STD-B24 y ARIB STD-23. [7]

ARIB STD-B23: Se ha desarrollado en base al estándar DVB-MHP, que define una plataforma para el motor ejecutor de aplicaciones interactivas, ARIB STD-B23 busca establecer una base común entre los estándares MHP, ACAP y actualmente con GEM, encaminado para que en el futuro se pueda compartir información y ejecutar aplicaciones entre estos estándares.

ARIB STD-24: este estándar define un lenguaje declarativo BML, desarrollado por la propia ARIB. El lenguaje BML se basa en XML, y es utilizado para la especificación de los servicios multimedia interactivos .

El middleware ARIB consta de una arquitectura formada por dos partes: presentación y ejecución. El modelo de ejecución utiliza la máquina virtual Java para la interpretación de los bytecodes, que representan los procedimientos y/o funciones relacionadas con el contenido que se transmite. A su vez, la plantilla de presentación especifica la sintaxis del lenguaje de marcado BML, en el cual se incluye etiquetas y atributos utilizados para la autoría del contenido declarativo de TV Digital. La codificación BML se le define como un lenguaje que está basado en XML. La arquitectura del middleware japonés es la que se presenta en la Figura 1.8, aquí se muestra de forma separada, la parte responsable de la presentación y la parte responsable de la ejecución.

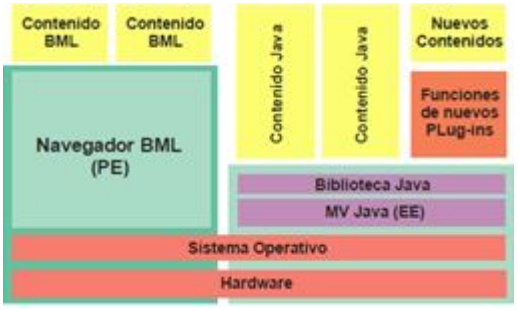


Figura 1.8: Arquitectura ARIB
Fuente: http://rodrigo.laiola.com.br/academic/laiola_monografia_interatividade.pdf

1.5.1.3. DASE- DTV Applications Software Environment

Es el estándar americano, que fue desarrollado por el ATSC para la capa de middleware de set top boxes de TV Digital, inicialmente fue diseñado como DASE nivel 1, éste se limitaba a un nivel de interactividad local, porque no contemplaba la existencia de un canal de retorno. Actualmente existe la versión DASE nivel 3, que a más de contemplar sólo la interactividad, tiene la finalidad de integrar la televisión con el Internet, permitiendo ofrecer opciones de navegadores web y manejo de correo electrónico en el terminal de acceso . Las aplicaciones DASE pueden ser declarativas y de procedimiento.

Las aplicaciones declarativas están representadas por documentos hipermedia y multimedia escritos por medio de un lenguaje de marcado basado en XHTML, de forma similar a MHP. En lo que se refiere a las aplicaciones de procedimiento utilizan una máquina virtual Java como un mecanismo que facilita la ejecución de instrucciones, igualmente de manera similar a MHP.

Cabe mencionar que una aplicación DASE no es exclusivamente declarativa o de procedimiento, en virtud que una aplicación de procedimiento puede hacer referencia a un contenido declarativo y viceversa. En la Figura 1.9 se muestra la arquitectura general de DASE, en ésta, la capa superior representa las aplicaciones que pueden ser declarativas o de procedimiento [2].

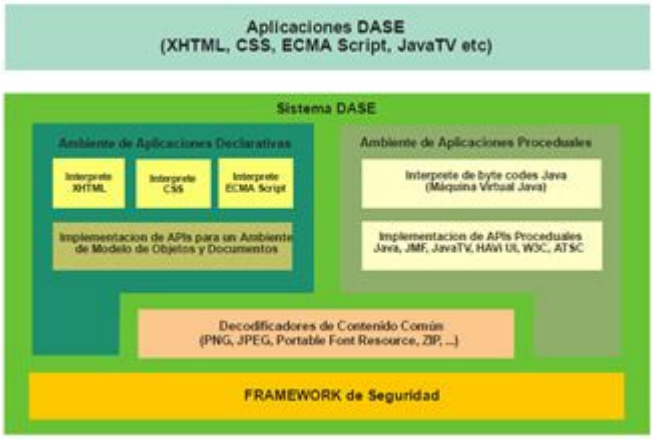


Figura 1.9: Arquitectura DASE
Fuente: http://comunidad.ginga.org.ar

El sistema DASE está formado por los módulos de Ambiente de Aplicación Declarativo o DAE y el Ambiente de Aplicación de Procedimiento o PAE. Un DAE es un subsistema lógico que procesa lenguajes de marcado, hojas de estilos y scripts. Su componente principal es el DCDE, cuya finalidad es hacer un análisis sintáctico del documento. Por otro lado, un PAE es el módulo encargado de procesar el contenido de los objetos activos o ejecutables, su componente principal es el PCDE, que por ejemplo, contiene la máquina virtual Java .

1.5.1.4. ACAP- Advanced Common Application Platform

Estándar creado por el Comité de Sistemas Avanzados de Television (ATSC), como base común para todos los sistemas de TV Digital interactivos de Estados Unidos, ya sean por cable, terrestres o por satélite; está basado en GEM y añade algunos elementos de OCAPque son adecuados para el mercado de USA.

El estándar OCAP, desarrollado por CableLabs, se deriva de MHP, es adaptado a las características técnicas y de negocios de las empresas de difusión por cable en los Estados Unidos.

ACAP sigue las especificaciones del estándar GEM y utiliza el mismo conjunto de APIs de Java y el modelo de aplicación que se utiliza en MHP. Las diferencias entre MHP y ACAP, es que este último tiene obligado un canal de retorno, el soporte a las aplicaciones almacenadas y la versión del carrusel de objetos de DSM-CC .

El estándar ACAP divide las aplicaciones en declarativas y de procedimiento. Las aplicaciones que tengan contenido sólo de procedimiento, que sean desarrolladas en Java y que permitan combinar gráficos, videos e imágenes, son denominadas aplicaciones ACAP-J; mientras que las aplicaciones declarativas se las conoce como ACAP-X, y son representadas por documentos multimedia escritos en un lenguaje de marcas como XHTML, reglas de estilo, scripts. Cabe mencionar que una aplicación ACAP, no tiene que ser completamente de procedimiento o declarativa, de ahí que una aplicación ACAP-J puede hacer referencia a un contenido declarativo, así como una aplicación ACAP-X puede hacer referencia a aplicaciones de procedimiento .

En la Figura 1.10 se presenta la arquitectura del middleware ACAP, para los sistemas de aplicaciones declarativas y de procedimiento. El entorno ACAP-X, es responsable de interpretar el contenido de una aplicación declarativa ACAP, el entorno ACAP-J, es el responsable del tratamiento de los contenidos de procedimiento a través de la máquina virtual Java y de los APIs de extensión .

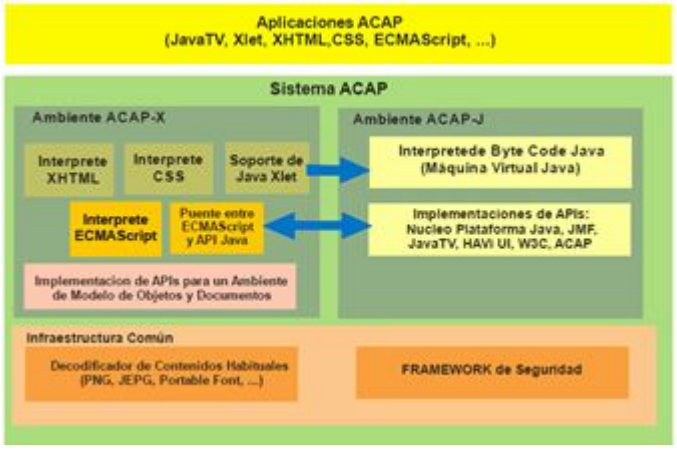


Figura 1.10: Arquitectura ACAP
Fuente: http://comunidad.ginga.org.ar

1.5.1.5. GEM Globally Executable MHP

El estándar GEM es considerado como un estándar de armonización entre plataformas internacionales de middleware existentes, tales como CableLabs (EE.UU), ARIB (Japón), con la finalidad de que todas las aplicaciones puedan ser utilizadas y presentadas en los terminales de acceso de estos estándares. Por ello el grupo DVB decidió crear la especificación denominada GEM. La especificación GEM, formalmente no puede considerarse como una especificación completa para terminales de acceso, lo correcto es decir que GEM es un framework a partir del cual la

implementación del middleware de un terminal de acceso puede ser instanciada, o todavía que GEM es un estándar al cual las implementaciones existentes deben ser adaptadas para obtener una conformidad que garantice la ejecución global de aplicaciones [8][9].

El GEM y su relación con los middlewares de otros estándares de TV Digital son los que se listan a continuación (ver Figura 1.11).

- Multimedia Home Platform (MHP), multiplataforma de middleware, desarrollada por el proyecto DVB.
- OpenCable Application Platform (OCAP), que es la capa de middleware de TV Digital para los sistemas de cable americanos.
- ARIB B.23, la especificación de la Asociación de Industrias y Negocios de radiodifusión de Japón;
- Advanced Common Application Platform (ACAP), el estándar de middleware de ATSC para televisión terrestre.
- BD-J, la plataforma Java para discos Blu-ray.
- Ginga-J, framework del middleware Brasileño.

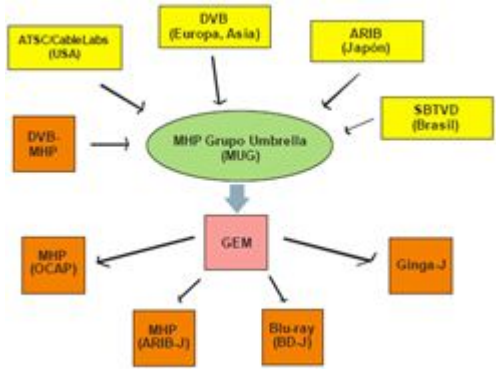


Figura 1.11: Concepto de GEM.
Fuente: http://www.hindawi.com/journals/ijdmb/2009/579569

1.5.1.6. Middleware Ginga

El middleware Ginga fue definido para el sistema brasileño de TV Digital, está basado en el estándar japonés.

Cuando se trabaja con un sistema complejo, como lo es la TV Digital interactiva, la mejor forma para entenderlo es a través de su arquitectura, que es la mejor manera de mostrar los principales elementos de un sistema, expresando claramente sus interacciones y ocultando los detalles menos importantes bajo el punto de vista adoptado. En la Figura 1.12 se muestra la arquitectura de TV Digital, incluido su middleware.

La idea central de la arquitectura en capas, consiste en que cada una ofrezca servicios para la capa superior y use los servicios ofrecidos por la inferior. De esta manera, las aplicaciones que se ejecutan en TV Digital interactiva usen la capa del middleware, que intermedia toda la comunicación entre las aplicaciones y el resto de los servicios ofrecidos por las capas inferiores .

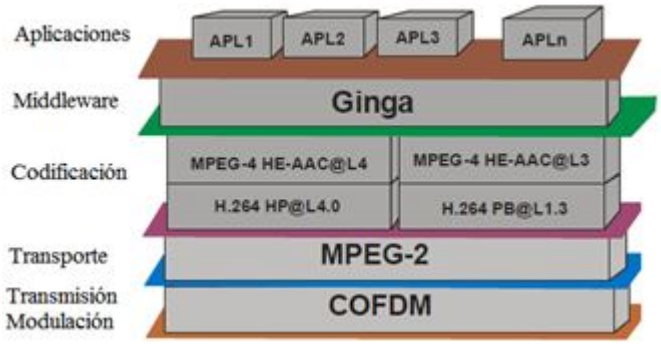


Figura 1.12: Arquitectura en capas del estándar SBTVD.
Fuente: http://tvdginga.wordpress.com/2010/05/20/o-middleware-ginga

Ginga es la capa de software intermedio (middleware), permite el desarrollo de aplicaciones interactivas para TV Digital, independientemente de la plataforma de hardware de fabricantes de terminales de acceso (STB) .

El middleware Ginga se divide en dos subsistemas interconectados, Ginga J (para aplicaciones de procedimiento Java) y Ginga-NCL (para aplicaciones declarativas NCL). Dependiendo de la funcionalidad del diseño de cada aplicación, un paradigma de programación es más adecuado que el otro .

En particular, las aplicaciones declarativas frecuentemente hacen uso de scripts, cuyo contenido es de naturaleza procedural. En el middleware Ginga una aplicación declarativa puede hacer referencia a una aplicación procedural, de la misma manera, una aplicación procedural puede hacer referencia a una aplicación declarativa. Por lo tanto, ambos tipos de aplicaciones Ginga pueden utilizar los beneficios que brindan los ambientes para las aplicaciones declarativas o procedurales [5].

1.6. Arquitectura del middleware ginga

1.6.1. Ambientes de Programación

Ambientes de programación en el ámbito de las aplicaciones para la TV Digital, existen tres posibles clasificaciones para el tipo de ejecución del software [10]:

- ***Procedural*** – necesita de una plataforma de ejecución (máquina virtual) y en el caso del middleware Ginga este módulo es denominado Ginga-J. Por utilizar el lenguaje de programación Java, posibilita que el programador sea capaz de establecer todo el flujo de control y ejecución de su programa;
- ***Declarativo*** – necesita de un Browser y se presenta semejante como una página HTML (HyperText Markup Language) que puede contener scripts y hojas de estilo. En el middleware Ginga, este módulo es se llama Ginga-NCL, que utiliza como base el lenguaje NCL, que define como una separación entre la estructura y el contenido. Generalmente las aplicaciones declarativas hacen uso de contenidos en script, que en el caso del NCL hay soporte del lenguaje Lua.
- ***Híbrido*** – representa la unión de los dos grupos, procedural y declarativo. Esta arquitectura se hace necesaria, pues las aplicaciones de TV Digital

son usualmente desarrolladas utilizando estos dos paradigmas de programación. Sin embargo, estos ambientes de programación no están precisamente disjuntos, o sea, uno puede utilizar las facilidades del otro a través de las API contenidas en el middleware. Esta característica posibilita la creación de aplicaciones híbridas, que son soportadas por la arquitectura del Ginga, ilustrada por la Figura 1.13.

1.6.2. Ginga

Este middleware está formado por un conjunto de tecnologías estandarizadas e innovaciones brasileñas que lo convierten en la especificación de middleware más avanzada y la mejor solución para las necesidades de TV Digital de Brasil y otros países que recién están iniciando en el mundo de la TV Digital [5].

Ginga a diferencia de los sistemas hipermedia convencionales, en donde prevalece el modelo de "servicio tipo pull", aquí el navegador o programa intérprete solicita un nuevo documento y se procede con la búsqueda del contenido, como generalmente ocurre con los navegadores web. En TV prevalece el modelo de "servicio tipo push",en este caso el canal o emisora ofrece para la difusión flujos de audio, vídeo y datos multiplexados con otros datos, pudiendo ser recibidos algunos de ellos por demanda, pero siendo predominante el "tipo de servicio push". Por lo tanto además de cambiar el drásticamente el paradigma de servicio, el usuario puede comenzar a ver un programa ya iniciado, cambiar de canal y consecuentemente salir y entrar en otros programas en curso [9].

Otro aspecto interesante, es que permite realizar la edición de documentos durante su exhibición en los programas en vivo y en programas modificados por retransmisoras.

La arquitectura del middleware Ginga, puede ser dividida en tres módulos: Ginga-CC (Common Core), el entorno de presentación Ginga-NCL (declarativo) y el entorno de ejecución Ginga-J (procedural). La arquitectura del middleware Ginga es la que se ilustra en la Figura 1.13.

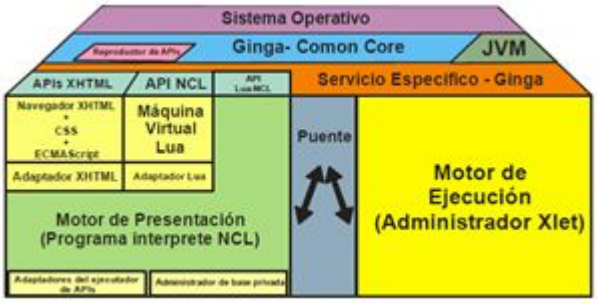


Figura 1.13: Arquitectura del middleware Ginga.
Fuente:http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html

Ginga-NCL y Ginga-J tienen la característica de ser construidos sobre los servicios ofrecidos por el módulo de núcleo común (Ginga- Common- Core), el Ginga CC está constituido por diversos módulos o componentes que facilitan la interacción del Hardware con las necesidades de las aplicaciones tanto declarativas como las de procedimiento. La composición de módulo común se muestra en la Figura 1.14.

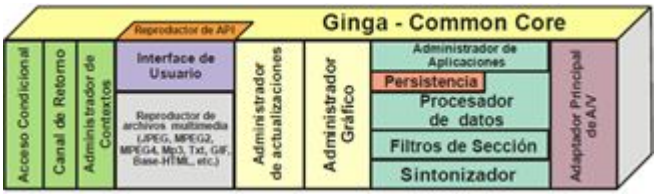


Figura 1.14: Ginga Common Core
http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html

Ginga-NCL fue desarrollado por la Pontificia Universidad Católica de Río de Janeiro (PUC), provee un entorno de presentación multimedia para aplicaciones declarativas escritas en el lenguaje NCL . Este lenguaje es una aplicación XML con facilidades para los aspectos de interactividad.

En síntesis Ginga-NCL, debe permitir el soporte a dos lenguajes procedurales, como LUA y JAVA. Lua es un lenguaje script de NCL diseñado para una programación procedimental con utilidades para la descripción de datos, brindando además soporte para la programación orientada a objetos, programación funcional y programación orientada a datos. Java debe tener las especificaciones de Ginga-J.

Ginga-J fue desarrollado por la Universidad Federal de Paraiba (UFPb), para proveer una infraestructura de ejecución de aplicaciones basadas en lenguaje Java, conocidas con el nombre de Xlet, con facilidades específicamente para el ambiente de interactividad en TV Digital [9][5].

Ginga-J es un subsistema lógico del Sistema Ginga que procesa aplicaciones procedimentales Xlets Java. El Xlets no necesita ser previamente almacenado en el STB, puede ser enviado por el canal de distribución. Es decir, el modelo Xlet se basa en la transferencia de código ejecutable por el canal de distribución para el STB y posterior carga y ejecución del mismo, de forma automática o manual.

Capítulo 2

Aplicaciones de Procedimiento (Ginga-J)

2.1. Ginga J.

Ginga-J está compuesto por un conjunto de APIs y su máquina virtual Java, que incorpora varias innovaciones para permitir la implementación de aplicaciones de TV Digital y satisfacer las necesidades especificas de TV Digital de Brasil, se puede realizar la manipulación desde datos multimedia hasta protocolos de acceso; que a su vez mantiene compatibilidad con la mayoría de middlewares de TV Digital desde que se unió a la norma GEM.

Ginga-J incluye soporte para la comunicación con dispositivos que utilizan tecnologías como son Bluetooth, Wi-Fi, infrarrojo, Línea de Poder, Ethernet o cualquier tecnología de red utilizada. Las aplicaciones pueden tener soporte de interacción por múltiples usuarios. El contexto sobre el cual se ejecuta la pila de software de Ginga-J es el que se muestra en la Figura 2.1. En ésta, la pila del software Ginga-J reside sobre el dispositivo Ginga, que puede ser por ejemplo un SetTopBox, un teléfono celular, etc [2].

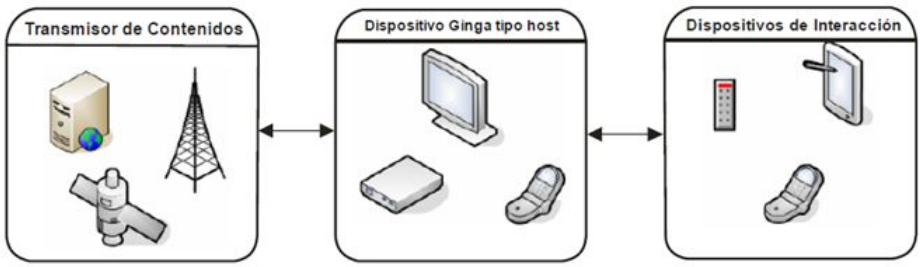


Figura 2.1: Contexto de Ginga-J
Fuente: http://comunidad.ginga.org.ar

[1]El midlleware Ginga debe tener acceso a flujos de video, audio, datos y otros recursos multimedia, pudiendo ser transmitidos a través del aire, cable, satélite o través de redes IP. Las informaciones deben ser procesadas y presentadas a los espectadores, los mismos que interactúan con la aplicación a través de los dispositivos de interacción de entrada y salida adjuntos o asociados al dispositivo

[1]ABNT NBR 15606-4. Televisão digital terrestre-Codicação de dados e especicações de transmissão para radiodifusão digital-Parte 4: Ginga-JAmbiente para a execução de aplicações procedurais. Asociación Brasilera de Normas Técnicas, 4, 2010. URL: http://www.dtv.org.br.[11]

Ginga. El dispositivo Ginga recibirá acciones por parte de los televidentes a través del dispositivo de interacción como el control remoto o teclado [11].

El dispositivo Ginga en respuesta a la acción que realice el espectador, presentará una respuesta visual, así como salidas de audio utilizando su propia pantalla y altavoces o pantallas y altavoces de los dispositivos de interacción. Un solo dispositivo puede tener la capacidad de entrada y salida simultáneamente.

La plataforma Ginga permite que varios espectadores puedan interactuar a la vez, para lo cual cada uno dispondrá de un dispositivo de interacción, y la plataforma debe distinguir los comandos enviados por y para cada dispositivo de interacción.

2.2. Arquitectura de Ginga-J.

El modelo de arquitectura y ambiente de ejecución de Ginga-J distingue entre entidades de hardware o de recursos, software del sistema, aplicaciones como las que se muestran en la Figura 2.2.

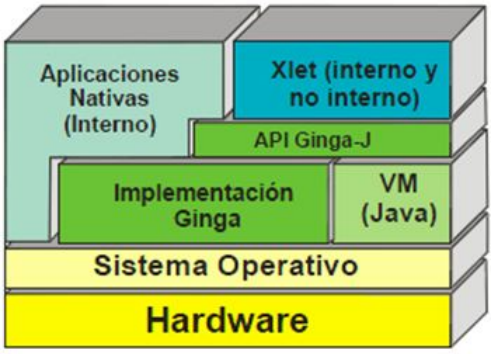


Figura 2.2: Arquitectura y ambiente de ejecución de Ginga-J
Fuente: http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html

Las aplicaciones nativas pueden ser implementadas usando funciones no estandarizadas, provistas por el sistema operativo del dispositivo Ginga, o por una implementación particular de Ginga. Las aplicaciones nativas también pueden incorporar funcionalidades provistas por las APIs estandarizadas Ginga-J. Las aplicaciones Xlets siempre deben utilizar APIs estandarizadas provistas por el Ginga-J [2].

2.3. El API de Ginga-J

Debido a que los middlewares existentes hasta ese entonces no cumplían con los requisitos identificados en el contexto brasileño sobre TV Digital, Ginga se basa en un conjunto de APIs que fueron creados para satisfacer las necesidades específicas de Brasil y al mismo tiempo mantener una compatibilidad internacional con el API de GEM.

Ginga se basa en tres grupos de APIs que permiten cumplir con toda la funcionalidad necesaria para implementar aplicaciones de TV Digital, estos grupos de APIs se han denominado: Verde, Amarillo y Azul (Figura 2.3). Los APIs Verde

de Ginga son los APIs compatibles con GEM. Los APIs Amarillo son aplicaciones compuestas para satisfacer las necesidades específicas de Brasil que pueden ser implementadas mediante el uso de un software de adaptación utilizando los APIs Verde. Los APIs Azul no son compatibles con los APIs de GEM . De esta forma las aplicaciones que utilizan sólo los APIS Verde pueden ser ejecutadas en los midlewares Ginga, MHP, OCAP, ACAP y ARIB STD-23. Las aplicaciones que utilizan los APIs Verde y amarillo solamente podrán ser ejecutadas en MHP, OCAP, ACAP y ARIB STD-23, si el software de adaptación es transmitido y ejecutado conjuntamente a la aplicación. Y finalmente las aplicaciones que utilicen los APIs Azul solo podrán ser ejecutadas en ambientes del midleware Ginga.

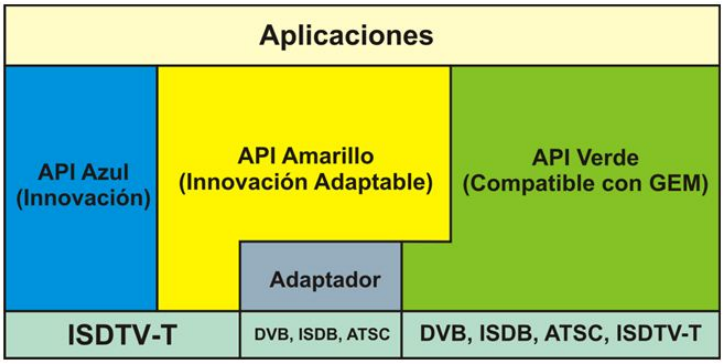


Figura 2.3: APIs Ginga-J
Fuente: http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html

A continuación definiremos los APIs de Ginga-J:

- Los APIs verde, están compuestos por los paquetes Sun JavaTV, DAVIC, HAVI y DVB, todos incluidos en el marco de especificaciones GEM.
- Los APIs Amarillo están conformados por el API JMF2.1, que es necesario para el desarrollo de aplicaciones avanzadas como captura de sonido por citar un ejemplo; una extensión de la API de Presentación de GEM, con funcionalidades para soportar las especificaciones de flujo de video definidas en el estándar Ginga; una extensión para la API del canal de retorno de GEM, que permite el envío de mensajes asíncronos; y una extensión de la API de Servicios de información del ISDB ARIB SDT-23 [9].
- Los APIs Azul están compuestos por un API de integración de dispositivos, que permite al receptor de TV Digital la comunicación con cualquier dispositivo con una interfaz compatible (Conexión por cable, como Ethernet o PLC, o redes inalámbricas como infrarrojo o Bluetooth), que puede ser utilizado como un dispositivo de entrada o de salida; una API multiusuario, que utiliza la API de integración de dispositivos para permitir que varios usuarios puedan interactuar simultáneamente con aplicaciones de TV Digital; una API puente a NCL, que permite el desarrollo de aplicaciones Java que contengan aplicaciones NCL .

A continuación describiremos los APIs más usados:

2.3.1. API JavaTV.

Es una extensión de la plataforma Java, ayuda en la producción de contenidos interactivos de manera procedural para la TV digital. El objetivo primordial de

este API es proporcionar un conjunto de métodos, clases e interfaces facilitando la creación de aplicaciones desarrolladas ejecutables en diversas plataformas para la recepción de TV Digital independientemente de las tecnologías utilizadas en la red de transmisión [5].

JavaTV soporta un nivel alto de interactividad, calidad gráfica y de procesamiento para ser ejecutado dentro de un set top box, siempre y cuando este equipada con la máquina virtual java que permita interpretar los bytecodes generados.

El API JavaTV realiza funciones como:

Streaming de audio y video: A más de la streaming de audio y video procedente de la estación, es factible generar otras aplicaciones con otros flujos.

Acceso a datos en el canal de transmisión: JavaTV puede recibir datos para las aplicaciones.

Aplicaciones con interactividad: Procesa datos y los reenvía a través de un canal de retorno.

Gestión del ciclo de vida de las aplicaciones: Permite que las aplicaciones coexistan con el contenido de TV convencional facilitando el intercambio de canal sin que la aplicación deje de existir.

2.3.1.1. Librerías del API JavaTV

- *javax.tvcarousel:* Proporciona acceso a archivos broadcast y directorio de datos a través de APIs que trabajan con el paquete java.io.
- *javax.tv.graphics:* Permite que los Xlets, puedan obtener su repositorio principal.
- *javax.tv.locator:* Proporciona una forma para referenciar datos en programas accesibles por la API JavaTV.
- *javax.tv.media:* Define una extensión para JMF (Java Media Framework) con la finalidad de gestionar los medios de comunicación en tiempo real.
- *javax.tv.media.protocol:* Proporciona acceso a un flujo de datos broadcast genérico.
- *javax.tv.net:* Permite acceso a datagramas IP (Internet Protocol) transmitidos en un stream broadcast.
- *javax.tv.service:* Proporciona mecanismos para accesar a la base de datos.
- *javax.tv.util:* Soporta la creación y gestión de eventos del temporizador.
- *javax.tv.xlet:* Proporciona interfaces para el desarrollo de aplicaciones y la comunicación entre las aplicaciones y el administrador.

2.3.2. API DAVIC (Digital Audio Visual Council)

Son especificaciones que presentan requisitos de sistemas audiovisuales que proporcionan interactividad de extremo a extremo. Se los puede utilizar en TV Digital con el fin de facilitar contenido al usuario final, también permite la interactividad con el mismo usuario.

A continuación listamos los paquetes que forman parte del API DAVIC:

- org.davic.media
- org.davic.resources
- org.davic.mpeg
- org.davic.mpeg.sections
- org.davic.net
- org.davic.net.dvb
- org.davic.net.tuning

2.3.3. API HAVi (Home Audio Video Interoperability).

Crea elementos, como la interfaz de usuario, cuyo objetivo es proporcionar un entorno fácil de usar para el espectador. Provee una extensión del paquete java.awt, permitiendo, asimismo, soporte de control remoto, transparencia entre otros.

Los paquetes que forman parte de la API HAVi incluidos en Ginga- J son:

- org.havi.ui
- org.havi.ui.event

2.3.4. API DVB.

En el desarrollo del middleware patrón MHP, el DVB incluye algunos paquetes para extender la funcionalidad ofrecida por JavaTV, HAVi y DAVIC. Estas características incluyen API de información de servicio, de intercomunicación entre Xlets, etc. Los paquetes que forman parte del API DVB, se mencionan a continuación.

- org.dvb.application
- org.dvb.dsmcc
- org.dvb.event
- org.dvb.io.ixc
- org.dvb.io.persistent
- org.dvb.lang
- org.dvb.media
- org.dvb.net

- org.dvb.net.tuning
- org.dvb.net.rc
- org.dvb.test
- org.dvb.ui

2.4. Ambientes de emulación para la programación de Ginga

Las aplicaciones pueden ser desarrolladas por los canales de televisión como por los televidentes. En caso de ser desarrollada por una emisora de televisión, la aplicación será enviada al SetTopBox a través de un canal de transmisión. En el caso de ser desarrollado por un usuario ésta tendrá que ser enviado al SetTopBox a través de una entrada externa como un USB portable, puerta de red, tarjeta de memoria, etc.

Para un desarrollador de aplicaciones de televisión digital interactiva es difícil disponer de una red experimental para realizar las pruebas correspondientes de las mismas, en la mayoría de los casos el medio ambiente es simulado con el uso de estaciones de prueba o con emuladores de software. Para el desarrollo de aplicaciones interactivas existen varios emuladores, que simulan el papel del middleware; los más utilizados para el ambiente Ginga-J son: el XleTView y el OpenGinga y para el ambiente Ginga-NCL el Virtual SetTopBox y el Emulador.

2.4.1. Emulador Ginga-J: XleTView.

Este emulador es de código abierto y está bajo la licencia de software libre GLP (General Public License), es usado para ejecutar Xlets en una PC. Tiene una implementación de referencia a la API JavaTV, pero además trae consigo implementaciones de otras APIs especificadas en el estándar MHP, como HAVI, DAVIC e implementaciones especificaciones por la propia DVB, además de las bibliotecas de PersonalJava. La pantalla de ejecución del emulador se puede ver en la Figura 2.4.

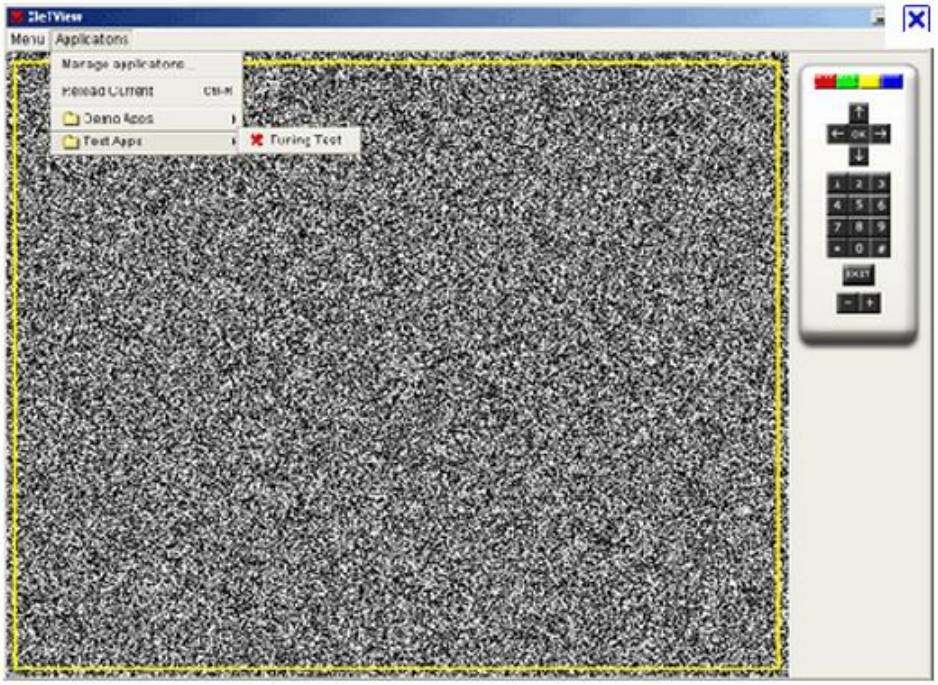


Figura 2.4: Interface del emulador XletView
Fuente:
http://www.interactivetvweb.org/tutorials/getting_started/xletview/running_applications

XleTView, está desarrollado en Java y para su ejecución independientemente del sistema operativo, es necesario utilizar el Java 2 Estándar Development Kit para compilar Xlets y ejecutar el XletView.

2.4.2. Emulador Ginga-J: OPENGINGA.

Openginga es un proyecto *open source* que está siendo desarrollado por el laboratorio LAVID de la Universidad Federal de Paraiba.

En la actualidad existe la máquina virtual con el sistema operativo Ubuntu 10.04 el cual tiene la implementación del emulador Open Ginga. La pantalla de ejecución del emulador OpenGinga se muestra en la Figura OpenGinga.

Figura 2.5: Pantalla del emulador OpenGinga
Fuente: http://b4dtv.blogspot.com/2009/08/tutorial-de-instalacao-da-maquina.html

2.5. Implementación de referencia de Ginga-J.

Se está desarrollando una implementación de referencia para validar las APIs y especificaciones de Ginga-J; hasta el momento la implementación tiene 800 mil líneas de código fuente y utiliza plataformas Intel-Kalaheo como banco de pruebas [2].

La implementación de referencia de Ginga-J se diseñó con la finalidad que sea el middleware de procedimiento para un receptor universal de TV Digital, ya que incorpora tres características importantes:

1. *Es multi-red:* Define un API de sintonización compatible con todas las redes utilizadas en la actualidad para la transmisión de TV Digital (terrestre, cable, satélite e IPTV).

2. *Es multi-sistema:* La especificación Ginga-J define un API de Servicio de Información compatible con ARIB B.23, sin embargo la implementación de referencia de Ginga-J incluye la tabla de procesamiento del Servicio de Información de DVB, ATSC y ARIB, utilizando esta API de Servicio de Información como la salida de datos.

3. *Es de aplicación compatible:* Ya que define el grupo de APIs Verde; la implementación de referencia Ginga-J es capaz de ejecutar la mayoría de aplicaciones diseñadas para ejecutarse sobre los middlewares de referencia compatibles con GEM.

2.5.1. Componentes de Software de middleware Ginga-J.

La implementación de referencia de Ginga-J utiliza un enfoque basado en componentes de software, este enfoque facilita la evolución funcional temporal del middleware, permitiendo la incorporación de nuevas funcionalidades a través de la adición de nuevos componentes y además facilita la reutilización de algunos de los componentes en otras implementaciones de middlewares, por ejemplo, para

PDAs, teléfonos móviles y receptores de TV de alta calidad (con mayor recursos, más costosos), receptores de baja calidad (con menor número de recursos, más económicos), etc. Aquellos componentes pueden ser reunidos sobre diferentes perfiles, en función de las características de la plataforma.

Los elementos de la arquitectura Ginga-J pueden ser agrupados de acuerdo a sus funcionalidades en las siguientes siete denominaciones:

1. *Componentes del acceso de flujo de bajo nivel:* Aquí se sitúan los elementos responsables para acceder a los flujos de transporte, para su procesamiento y demultiplexación en los distintos flujos elementales que construyen.
2. *Componentes del procesamiento de flujos elementales:* Contiene los elementos responsables del tratamiento de los flujos elementales, su decodificación y facilitación para otros componentes.
3. *Componentes de interfaz de usuario:* Lleva los elementos que permiten la interactividad con el usuario a través de la presentación de elementos audiovisuales, así como la administración de eventos generados de usuario (puede incluir dispositivos externos).
4. *Componentes de comunicación:* Permiten la comunicación entre aplicaciones que se ejecutan sobre el middleware.
5. *Componentes de gestión:* Gestión del middleware (gestión de contextos, actualizaciones del middleware, etc) y gestión aplicaciones (control del ciclo de vida, control de fallos, etc).
6. *Componentes de persistencia:* Son responsables del almacenamiento de datos persistentes.
7. *Componente de acceso condicional:* Responsables de la seguridad del acceso restringido de los contenidos transmitidos por proveedor de contenidos.

2.5.1.1. Componentes de acceso de flujo de bajo nivel.

El sintonizador: Selecciona el canal físico, así como de uno de los flujos de transporte que está siendo transmitido por el canal seleccionado.

Servidor de información de flujo: Identifica cuales son los flujos elementales presentes en el flujo de transporte seleccionado por el sintonizador y proporciona información relacionada (programación, opciones disponibles de audio, subtítulos, etc) de los otros componentes del middleware.

Demultiplexor: Proporciona los flujos elementales en el flujo de transporte de los otros componentes del middleware. Estos flujos primarios pueden estar disponibles para cualquier componente Ginga-J, por una aplicación que se ejecuta sobre el middleware o algún componente de hardware (por ejemplo un decodificador de audio y video).

2.5.1.2. Componentes del procesamiento de flujos elementales.

Controlador de procesamiento multimedia: Controla el procesamiento de los elementos multimedia para proporcionarlos a los otros componentes del middleware. A través de esta API se hace posible seleccionar que flujo de audio y video debe ser decodificado, que subtitulo se debe utilizar para iniciar y detener el proceso de decodificación de los archivos multimedia, etc. Contiene analizadores y procesadores multimedia específicos para cada tipo de multimedia soportado.

Procesador de Flujo de Datos: Accede, procesa y proporciona a los otros componentes del middleware flujos de datos elementales como: el carrusel de datos, paquetes IP transmitidos a través de broadcast, etc. También es responsable de notificar a los otros componentes de eventos como la llegada de aplicaciones, eventos síncronos y asíncronos, etc.

2.5.1.3. Componentes de interfaz de usuario.

Controlador de presentación multimedia: Permite la presentación de los flujos de audio, video, imágenes, etc.

Administrador de eventos de usuario: Maneja los eventos creados por el espectador, como la manipulación de una tecla del control remoto, los comandos de un dispositivo de interacción (PDA) y pasa estos eventos a las aplicaciones registradas y/o componentes del middleware.

Elementos gráficos: Compuestos por elementos gráficos principales utilizados para crear una aplicación, estos incluyen botones, cajas de texto, etc.

2.5.1.4. Componentes de comunicación.

Canal de interacción: Soporta múltiples redes tales como PTSN, Ethernet, GSM, Wimax, etc. Este componente es responsable de brindar interfaces que puedan ser utilizadas por otros componentes del middleware para acceder al canal de interacción, que es una canal bidireccional de datos que puede utilizarse por las aplicaciones locales para comunicarse con las aplicaciones remotas.

Comunicación entre aplicaciones: Las aplicaciones que se ejecutan sobre Ginga-J pueden comunicarse entre sí por la utilización de las APIs proporcionadas por este componente.

2.5.1.5. Componentes de Gestión.

Gestor de aplicaciones: Es un elemento de software que: carga, configura, instala y ejecuta las aplicaciones sobre Ginga-J proporcionando una API para controlar el ciclo de vida de la aplicación; identifica y previene el fallo de la aplicación, además gestiona la utilización de recursos y el control de acceso.

Gestor de Middleware: Actualiza el código del middleware en tiempo de ejecución, esto permite a los componentes del middleware ser sustituidos por la corrección de errores debido al cambio o mejora de la funcionalidad. Además este componente proporciona información relativa al contexto actual del receptor de

TV Digital que hace referencia a la capacidad, uso y disponibilidad de los recursos del CPU, memoria, etc.

2.5.1.6. Componentes de persistencia.

Gestor de perfil: Proporciona a todos los componentes del middleware o a las aplicaciones que se ejecutan sobre Ginga-J, un conjunto de datos definidos por el receptor de TV digital (preferencias definidas por el usuario).

Persistencia: Permite a un objeto almacenarse después de terminado el proceso que lo creó.

2.5.1.7. Componente de acceso condicional.

Módulo de acceso condicional: Es el módulo del middleware responsable de controlar el acceso a contenidos restringidos transmitidos por el receptor. Estos contenidos pueden ser recibidos tanto desde el canal de broadcast como desde el canal de retorno (IPTV).

2.6. Lista de paquetes mínimos de Ginga-J.

2.6.1. Paquetes de la plataforma Java.

Los paquetes de la plataforma Java se listan a continuación:

- java.awt
- java.awt.color
- java.awt.event
- java.awt.font
- java.awt.im
- java.awt.image
- java.beans
- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.rmi
- java.rmi.registry
- java.security
- java.security.acl

- java.security.cert
- java.security.interfaces
- java.security.spec
- java.text
- java.util
- java.util.jar
- java.util.zip
- javax.microedition.io
- javax.microedition.pki
- javax.microedition.xlet
- javax.microedition.xlet.ixc
- javax.security.auth.x500

2.6.2. Paquetes de la especificación JavaTV 1.1 y JMF 1.0

Los siguientes paquetes son incluidos por esta parte de la ABNT NBR 15606:

- javax.media
- javax.media.protocol
- javax.tv.graphics
- javax.tv.locator
- javax.tv.media
- javax.tv.net
- javax.tv.service
- javax.tv.service.guide
- javax.tv.service.navigation
- javax.tv.service.selection
- javax.tv.service.transport
- javax.tv.util
- javax.tv.xlet

2.6.3. Paquetes de la especificación JavaDTV

Los siguientes paquetes son incluidos por esta parte de la ABNT NBR 15606:

- com.sun.dtv.application
- com.sun.dtv.broadcast
- com.sun.dtv.broadcast.event
- com.sun.dtv.filtering
- com.sun.dtv.io
- com.sun.dtv.locator
- com.sun.dtv.lwuit
- com.sun.dtv.lwuit.animations
- com.sun.dtv.lwuit.events
- com.sun.dtv.lwuit.geom
- com.sun.dtv.lwuit.layouts
- com.sun.dtv.lwuit.list
- com.sun.dtv.lwuit.painter
- com.sun.dtv.lwuit.plaf
- com.sun.dtv.lwuit.util
- com.sun.dtv.media
- com.sun.dtv.media.audio
- com.sun.dtv.media.control
- com.sun.dtv.media.dripfeed
- com.sun.dtv.media.format
- com.sun.dtv.media.language
- com.sun.dtv.media.text
- com.sun.dtv.media.timeline
- com.sun.dtv.net
- com.sun.dtv.platform
- com.sun.dtv.resources
- com.sun.dtv.security
- com.sun.dtv.service
- com.sun.dtv.smartcard
- com.sun.dtv.test
- com.sun.dtv.transport

- com.sun.dtv.tuner
- com.sun.dtv.ui
- com.sun.dtv.ui.event.

2.6.4. Paquetes de la especificiación JSSE 1.0.1

Los siguientes paquetes son incluidos por esta parte de la ABNT NBR 15606:

- com.sun.net.ssl
- javax.net
- javax.net.ssl
- javax.security.cert

2.6.5. Paquetes de la especificación JCE 1.0.1

Los siguientes paquetes son incluidos por esta parte de la ABNT NBR 15606:

- javax.crypto
- javax.crypto.interfaces
- javax.crypto.spec.

2.6.6. Paquetes de la especificación SATSA 1.0.1

El siguiente paquete es incluido por esta parte de la ABNT NBR 15606:

- javax.microedition.apdu

2.6.7. Paquetes específicos de Ginga-J

Los siguientes forman parte de esta plataforma:

- br.org.sbtvd.net
- br.org.sbtvd.net.si
- br.org.sbtvd.net.tuning
- br.org.sbtvd.bridge
- br.org.sbtvd.ui

2.7. Núcleo Común de Ginga (Ginga Common - Core)

Este subsistema es la interfaz directa con el sistema operativo, haciendo un puente estrecho con el hardware. El núcleo común de Ginga ofrece servicios tanto para el ambiente de presentación (declarativo) como para el de ejecución (procedimiento). Aquí es donde se accede al sintonizador de canales, sistema de archivos, entre otros [2] [12].

Está formado por los decodificadores de contenido común, que sirven tanto a las aplicaciones de procedimiento como a las declarativas que necesiten decodificar

y presentar tipos comunes de contenidos como PNG, JPEG, MPEG , etc; y por procedimientos para obtener contenidos transportados en flujo de transporte MPEG-2 y a través del canal de interactividad.

Los componentes básicos del Núcleo Común se describen a continuación:

El sintonizador: Sintoniza un canal, seleccionando un canal físico y los flujos de transporte que están siendo enviados por este canal.

Filtros de Selección: Una vez sintonizado el canal, el middleware debe ser capaz de acceder a partes específicas del flujo de transporte. Para esto se utilizan los filtros de selección, que buscan en el flujo, la parte exacta que las APIs necesitan para su ejecución. Funciona como un filtro, sólo deja pasar la información que es requerida por la API.

Procesador de Datos: Accede, procesa y transfiere los datos recibidos por la capa física, además notifica a los otros componentes sobre cualquier evento que se ha recibido.

Persistencia: Ginga es capaz de guardar archivos, incluso luego que haya finalizado el proceso que los creó, para que pueda ser abierto en otra ocasión.

Administrador de Aplicaciones: Carga, configura, inicializa y ejecuta cualquier aplicación ya sea de un entorno declarativo o de procedimiento. También controla el ciclo de vida de las aplicaciones, eliminarlas cuando sea necesario, además de controlar los recursos utilizados por esas APIs.

Adaptador Principal de A/V: Con éste las aplicaciones pueden ver el flujo de audio y video. Esto es necesario cuando una aplicación necesita controlar sus acciones, de acuerdo con lo que se está transmitiendo.

Administrador de Gráficos: Las normas del middleware definen como se presentan al usuario las imágenes, videos, datos, etc.

Administrador de Actualizaciones: Gestiona las actualizaciones del sistema, controlando, descargando las actualizaciones del middleware siempre que sea necesario para corregir los errores que puedan existir en versiones anteriores.

Reproductor de archivos multimedia: Presentan los archivos multimedia recibidos, como por ejemplo archivos de tipo MPEG, JPEG, TXT, GIF, etc..

Interface de usuario: Capta e interpreta los eventos generados por los usuarios, como por ejemplo comandos de control remoto; y notifica a los otros módulos interesados.

Administrador de Contextos: Capta las preferencias del usuario, notificando a los otros componentes interesados esas preferencias, como por ejemplo horario en el que el usuario mira la TV o el bloquear o desbloquear canales.

Canal de Retorno: Proporciona la interfaz de las capas superiores con el canal de interacción (o canal de retorno). Además, debe gestionar el canal de retorno de modo que los datos sean transmitidos cuando el canal esté disponible o forzar la transmisión en caso de que el usuario o una aplicación tengan definido un horario exacto.

Acceso Condicional: Restringe contenidos inapropiados recibidos por los canales de programación, brindando así seguridad para el middleware.

Capítulo 3

Aplicaciones Declarativas (Ginga-NCL)

3.1. Modelo de contexto anidado (NCM)

NCM es un modelo conceptual, se centra en la representación y manipulación de documentos hipermedia. El modelo también debe definir las reglas que estructuran y las operaciones en los datos de la manipulación y la actualización de las estructuras [13].

Un documento hipermedia está compuesta por nodos y enlaces (links), donde cada nodo representa una media y cada enlace representa una relación entre medias (ver Figura 3.1), sin embargo, los enlaces no son la única entidad disponible para la definición de las relaciones. Un ejemplo para la mejor compresión es que en un determinado momento de la representación de una media otra debe ser inicializada, es decir los enlaces hacen que el espacio de sincronización de tiempo entre los puntos que componen el documento[14].

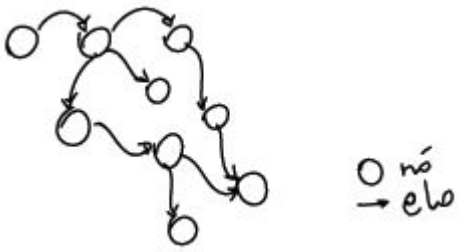


Figura 3.1: Nodos y enlaces de un documento hipermedia común.
Fuente: http://www.ncl.org.br

NCM va más allá de un documento hipermedia tradicional, ya que los gráficos se pueden anidar, lo que permite segmentación y la estructuración hipermedia como sea necesario o deseado [13], por lo que se extiende los conceptos sobre el aumento de la potencia y flexibilidad de un documento hipermedia [15].

Esto se hace a través de nodos de composición o también llamados de contexto, todo nodo media está definido dentro de un contexto, un ejemplo de estos se muestran en la Figura 3.2.

NCM es el modelo subyacente al NCL, un idioma de aplicación de XML para la creación de documentos hipermedia.

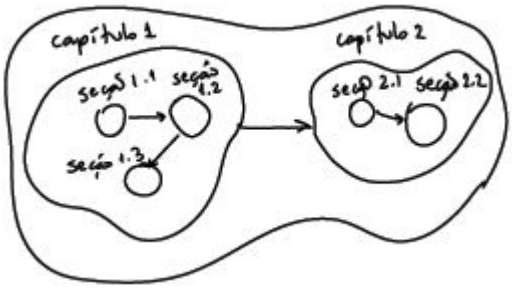


Figura 3.2: Nodos, enlaces y nodos de composición (contexto).
Fuente: http://www.ncl.org.br

En NCM se extiende la definición de nodos en dos tipos, de contenido y nodos de composición.

- Un nodo de contenido aporta información sobre una media usada por el documento, este es asociado a un elemento de media tales como vídeo, audio, imagen, texto, aplicación, etc.
- Nodo de composición contiene otros nodos de composición y un conjunto de ellos, siendo utilizados para dar estructura y organización de un documento hipermedia.

Para facilitar la elaboración de un documento hipermedia siguiendo el modelo NCM se desarrolló el lenguaje NCL, que es el lenguaje de programación sobre el que se basa el presente estudio.

3.2. Estructura de un documento Hipermedia.

Sobre la construcción de un documento hipermedia, algunas informaciones básicas son necesarias como se muestra en la Figura3.3 [16].

Figura 3.3: Estructura de un documento hipermedia
Fuente: http://tvd.lifia.info.unlp.edu.ar

3.2.1. ¿Qué vamos a mostrar? - Objetos Media

Al comenzar a diseñar un programa es el contenido audiovisual interactivo lo primero que se escoge; que vídeos, imágenes, textos y otros medios será presentado

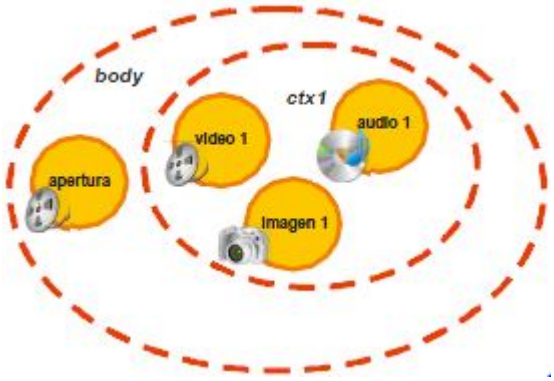


Figura 3.4: Representación de nodos multimedia y su composición
Fuente: http://ginga.softwarelibre.org.bo

por el programa, este contenido esta representado por los nodos de media. Una media representa cada nodo de un documento que informa el descriptor cual está relacionado.

De acuerdo con el NCM, una media debe estar necesariamente dentro de un nodo de composición, llamada contexto, que es usado para representar un documento o parte de él. En NCL, el elemento body es el contexto que contiene todos los nodos del documento hipermedia, sean estos de media o de contextos.

La Figura 3.4 muestra el concepto de media y contextos, con cuatro nodos multimedia, tres de los cuales están dentro de un contexto (ctx1) anidado al cuerpo (body).

3.2.2. ¿Dónde los vamos a mostrar? - Regiones

Después de definir el contenido multimedia del programa, se debe definir el área en donde se va a mostrar cada elemento en la pantalla, por medio de elementos llamados regiones. Una región representa la posición y tamaño de la zona donde ciertos objetos media serán visualizados, es decir una región sirve para inicializar la posición de los medios en una ubicación específica [17].

Aunque una región representa el lugar en donde se podría presentar un nodo media, ésta no indica que nodo media será presentado en dicha región, esta asociación se realiza por medio de un descriptor.

3.2.3. ¿Cómo los vamos a mostrar? - Descriptores.

La definición de una región debe ser complementada con otra información que indique cómo cada nodo será presentado. Esta descripción de las características de cada nodo se realiza a través de los elementos llamados descriptores. Un descriptor puede detallar parámetros de representación de los nodos, incluyendo a la región donde tendrá lugar la presentación de su volumen, su transparencia, y el tiempo de duración, entre otros. La Figura3.6 muestra un descriptor [14].

Cuando se define un descriptor, es necesario definir la región a la que se asocia. Todos los medios que utilizan éste son asociados a la región correspondiente[13].

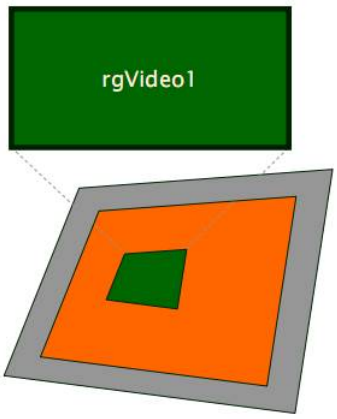


Figura 3.5: Representación de una región
Fuente: http://ginga.softwarelibre.org.bo

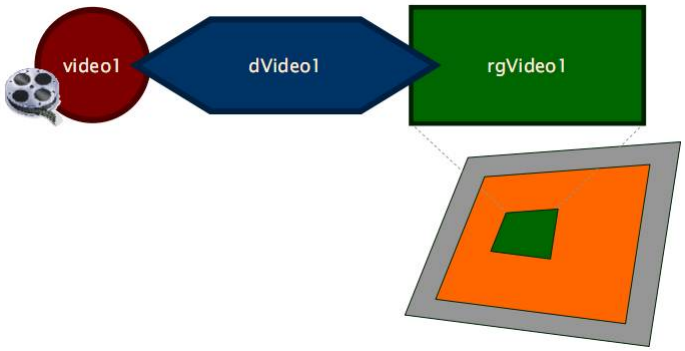


Figura 3.6: Representación de un descriptor
Fuente: http://ginga.softwarelibre.org.bo

3.2.4. ¿Cuándo los vamos a mostrar? - Links y Conectores

Una vez que se hayan seleccionado los nodos que formaran parte del documento hipermedia, se hace necesario definir cuál será el primer nodo en ser presentado y el orden de ejecución de los demás. Está definición se hace con el elemento llamado puerto, estos definen los nodos que serán presentados cuando un nodo de contexto iniciara. Si hay más de un puerto en el contexto cuerpo, se abren todos en paralelo.

Los puertos son necesarios para dar acceso a los nodos (sean de media o de contexto) internos para cualquier contexto y no sólo al cuerpo. En la Figura3.7, el nodo video1 del contexto ctx1 sólo puede ser accedido fuera del contexto ctx1, a través de la puerta pVideo1, mientras que los nodos audio1 e imagen1 no pueden ser ingresados fuera del contexto ctx1.

Los links no definen todas las relaciones de sincronización entre los nodos y la interactividad del programa, para eso requiere el uso de conectores.

3.3. Lenguaje de marcado extensible (XML).

Creado por el W3C a mediados de la década de 1990, el idioma XML [18] es un formato textual simple y bastante flexible diseñado para estructurar, almace-

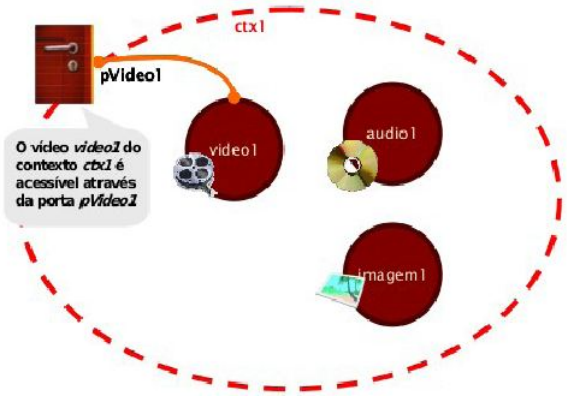


Figura 3.7: Puertos de un nodo de composición
Fuente: http://ginga.softwarelibre.org.bo

nar y representar información. Cómo XML no fue diseñado para una aplicación específica, puede ser utilizado como una base (metalenguaje) para el desarrollo de las lenguas de marca .

En XML los documentos están organizados jerárquicamente en forma de árbol, donde cada elemento tiene un elemento principal y elementos secundarios. Esta estructura se asemeja a la distribución de un árbol genealógico de una familia donde cada uno sería un elemento XML, es decir posee una identificación y atributos.

En la definición del patrón textual XML de un elemento se inicia con el símbolo "<" y termina por medio de símbolos "/>", más la repetición del nombre entre los símbolos de terminación. Entre estos dos símbolos se definen el nombre del elemento y los atributos del mismo. Los atributos de un elemento XML se definen por un par (nombre, valor), el valor de cada atributo se indica después del símbolo "=" y entre comillas. Cabe señalar que el nombre del elemento, así como sus atributos, no tiene letras mayúsculas o acentos, esta es una buena práctica para evitar la aparición de errores en el uso de documento, ya que XML es sensitivo entre mayúsculas y minúsculas.
Uno de los elementos, puede poseer otros elementos como hijos.

3.3.1. NCL

NCL es un lenguaje declarativo XML basado en el modelo conceptual NCM en la Figura 3.8 se muestran las entidades básicas del mismo. Además NCL tiene la facilidad para especificar los aspectos de la interactividad, en tiempo-espacio entre los objetos de las medias, posee adaptabilidad, soporte a múltiples dispositivos y producción en vivo de programas interactivos no lineales. Tiene la ventaja adicional en el uso del lenguaje de *script* Lua, para la manipulación de sus variables, mediante la adopción del paradigma imperativo, ser eficiente, rápido y ligero y diseñados para extender las aplicaciones.
Cada nodo posee un identificador, un contenido y un conjunto de anclas (subconjunto de unidades de información de un nodo).

NCL a diferencia de XHTML o HTML tiene una separación estricta entre el contenido y la estructura de un documento, por lo tanto es una aplicación de TVD que puede ser generada o modificada en vivo, permitiendo definir objetos de

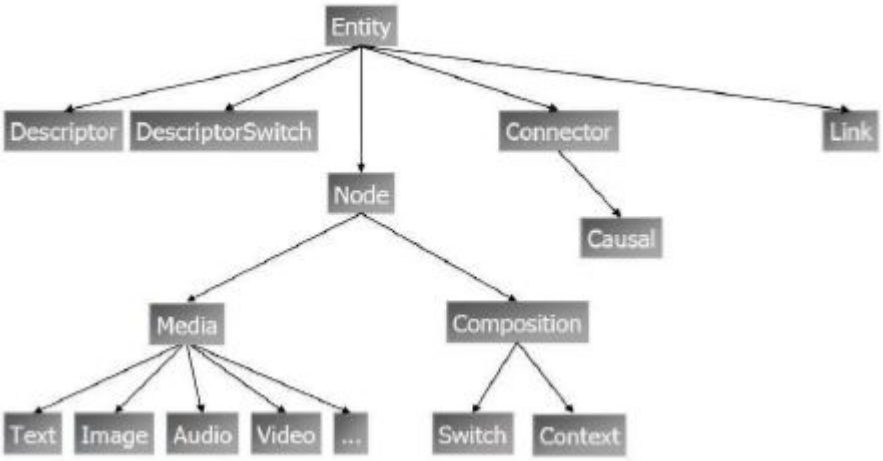


Figura 3.8: Entidades básicas del modelo NCM
Fuente: http://www.softwarepublico.gov.br

media estructurados y relacionados tanto en tiempo y espacio. Los componentes de este subsistema se muestran en la Figura 3.9.

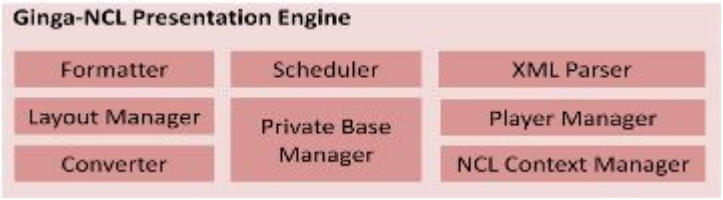


Figura 3.9: Subsistema Ginga-NCL
Fuente: Investigación del Estudio del Middleware GINGA y Guía de usuario del Middleware GINGA

A continuación se define los elementos principales de Ginga-NCL

- ***Formateador (Formatter):*** recibe y controla las aplicaciones multimedias escritas en NCL, siendo éstas entregadas por el Ginga-CC.
- ***Analizador de XML (XML Parser), Convertidor (Converter):*** traduce la aplicación NCL en la estructura interna de datos de Ginga-NCL para controlar la misma. Estos componentes son solicitados por el Formateador.
- ***Programador (Scheduler):*** organiza el orden de la presentación del documento NCL (antes que inicie los objetos de media, se evalúan las condiciones de los enlaces y la programación correspondiente a las relaciones de las acciones que guiaran el flujo de la presentación). El componente Programador es responsable para dar la orden al componente Administrador de la Reproducción (Player Manager) para iniciar la reproducción apropiada del tipo de contenido de media para exhibirlo en el momento indicado.[19].
- ***Base Privada (Private Base):*** el Motor de Presentación (Presentation Engine) lidia con un conjunto de aplicaciones NCL que están dentro de una estructura conocida como Base Privada.
- ***Administrador de la Base Privada (Private Base Manager):*** este componente está a cargo de recibir los comandos de edición de los documen-

tos NCL y el darle mantenimiento a los documentos NCL presentados. Estos comandos de edición están divididos en tres subgrupos:

- 1er Grupo de Comandos, responsable por la activación y desactivación de una base privada, o sea, la habilitación de una determinada aplicación NCL.
- 2do Grupo de Comandos, responsable de iniciar, pausar, resumir, detener, remover las aplicaciones NCL.
- 3er Grupo de Comandos, responsable de la actualización de aplicaciones en tiempo real, permitiendo el agregar o remover elementos NCL y permite que se asignen valores a las propiedades de los objetos de media.

- ***Administrador del Diseño (Layout Manager):*** el Motor de Presentación soporta múltiples dispositivos de presentaciones a través del componente Administrador del Diseño, el cual es responsable de mapear todas las regiones definidas en una aplicación NCL

NCL no define ninguna de las medias en sí, por el contrario, especifica la ligadura que mantiene juntos las medias como en una presentación multimedia. Por lo tanto, un documento NCL sólo puntualiza cómo los objetos de las medias están estructurados y relacionados en tiempo y espacio. Como un lenguaje de pegamento, no restringe o prescribe el contenido de los objetos media. Entre los tipos de media usuales que soporta están los siguientes [10][20]:

- Vídeo (MPEG, MOV, etc.)
- Audio (MP3, WMA, etc.)
- Imagen (GIF, JPEG, etc.)
- Texto (TXT, PDF, etc.)
- HTML
- scripts LUA.

Ginga-NCL, debe ofrecer soporte a dos lenguajes procedurales, como son LUA y JAVA. Lua es el lenguaje script de NCL, y Java debe seguir las especificaciones de Ginga-J.

Además de los formatos antes mencionados, Ginga-NCL también ofrece apoyo a los objetos de medias basados en XHTML, donde es tratado como un caso especial. De esta manera, NCL no reemplaza el XHTML, sino que los complementa. Los objetos basados en XHTML que apoyará la NCL varían dependiendo de la aplicación y el navegador incrustado en el formateador NCL. Según la ABNTNBR 15606-2 y ABNT NBR 15606 -5, se define como obligatoria sólo un conjunto de funcionalidades para XHTML y sus tecnologías resultantes, el resto de las funciones son opcionales [21].

El ambiente declarativo es en sí muy limitado. Las aplicaciones que utilicen éste deben tener su enfoque sobre el sincronismo, siendo el foco del lenguaje NCL exactamente eso, no la interactividad, ya que la interacción es tratada como resultado de la sincronización.

3.3.2. Lua.

Desde sus inicios Lua fue diseñado para ser utilizado en conjunto con otros lenguajes, no es muy común que un programa sea escrito puramente en éste. Este lenguaje permite que una aplicación principal pueda ser ampliada o adaptada a través de scripts. Lua es un lenguaje que combina sintaxis procedural con declarativa, con pocos comandos primitivos. Por lo tanto, comparado con otros lenguajes, posee implementación ligera y extensible. Otra de las características es que posee un Garbage-collection, un sistema dinámico de tipos y un alto grado de portabilidad, pudiendo ser ejecutada en diversas plataformas, tales como computadores personales, celulares, consolas de videojuego, etc . El nombre del lenguaje, Lua, se remite a la idea de un lenguaje satélite.

[1]Siendo un lenguaje de extensión, Lua no tiene noción del programa principal (main): sólo funciona como una extensión en un cliente anfitrión, denominado programa contenedor o simplemente anfitrión (host) que invoca funciones para ejecutar un segmento de código Lua, puede escribir y leer variables de Lua y puede registrar funciones C para que sean llamadas por el código Lua[22].

Las características de Lua aparte de su alto rendimiento, bajo consumo de recursos, simplicidad, eficiencia, portabilidad, además de su licencia libre de ***royalties*** que reduce el costo a cero de la adopción del interpretador por unidad producida, combinan a la perfección con el escenario de la TV Digital. La portabilidad es importante cuando el middleware sea desarrollado para dispositivos con características contradictorias como la telefonía celular y SetTopBoxes.

3.3.2.1. Extensiones de NCLua

Lua es el lenguaje de script adoptado por el módulo Ginga-NCL para implementar objetos imperativos en documentos NCL que son puramente declarativos. La definición completa do Lua puede ser vista en .

Para adecuar al ambiente de la televisión digital y que se integre a NCL, el lenguaje Lua se fue ampliando con nuevas funcionalidades generando así el plugin NCLua. Por ejemplo, este plugin necesita comunicarse con el documento NCL para saber cuándo su objeto <media> correspondiente es iniciado por un enlace. Un NCLua también puede responder a las claves en el control remoto, y realizar las operaciones dibujo o escritura libremente dentro de la región NCL que le está destinado. Estas características no son específicas del idioma NCL, por lo que no son parte de la biblioteca patrón. Lo que diferencia un NCLua de un programa Lua puro es el hecho de que es controlada por el documento NCL en la cual se inserta, y utilizar las extensiones descritas a continuación [17].

Además de la biblioteca estándar de Lua, cinco nuevos módulos están disponibles para los scripts NCLua:

1. ***Módulo NCLEdit:*** permite que scripts Lua manipulen objetos declarativos de documentos NCL, adicionando, modificando e removiendo informaciones;

2. ***Módulo event:*** permite que objetos NCLua puedan comunicarse con el documento NCL y otras entidades externas (tales como control remoto y el canal de interactividad), a través de eventos de una forma asíncrona.

[1]Pagina Ocial de Lua. Manual de referencia de lua 5.1. 20072008. URL: http://www.lua.org/manual/5.1/es/manual.html.

3. ***Módulo canvas:*** ofrece elementos (API) para diseñar objetos gráficos en la región de NCLua.

4. ***Módulo settings:*** exporta una tabla con variables definidas por el autor del documento NCL en variables de ambiente reservadas, contenidas en el nodo application/x-ginga-settings.

5. ***Módulo persistent:*** exporta un cuadro con las variables persistentes entre ejecuciones de objetos imperativos, estos datos están guardados en un área restringida del middleware.

La norma ABNT NBR 15606 -2:2007 [16]y H. 761 lista en detalle todas las funciones soportadas por cada módulo.

Las siguientes funciones de la biblioteca de Lua son dependientes de la plataforma y por lo tanto no están disponibles para los scripts NCLua:

- En el módulo paquete: la función loadlib.
- En el módulo io: todas las funciones.
- En el módulo os: las funciones reloj, ejecutar, salida, getenv, quitar, cambiar tmpname y setlocale.
- En el módulo debug: todas las funciones.

3.4. Estructura de un documento NCL.

Todo contenido de un documento NCL está definido dentro del elemento <ncl> siendo su estructura dividida en dos grandes partes, la cabecera (head) y el cuerpo del texto (body).

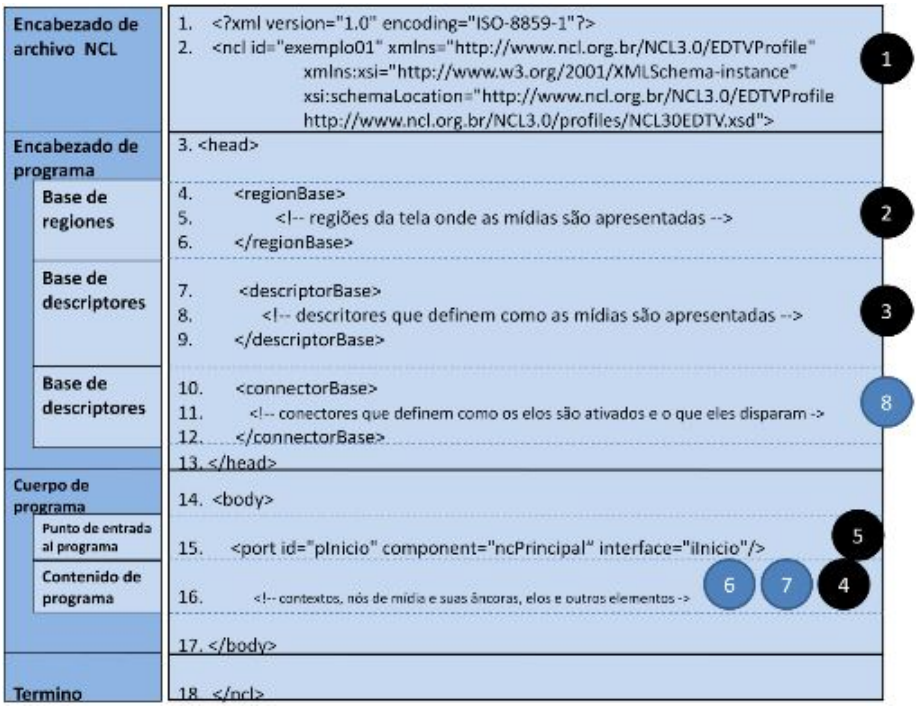
- Archivo de encabezado NCL (líneas 1 y 2);
- Una sección del encabezado (sección head, las líneas 3 a 13), que define las regiones, los descriptores, los conectores y las reglas utilizadas por el programa;
- El cuerpo del programa (sección body, líneas 14 a 17), donde se definen los contextos, en los nodos de media, enlaces y otros elementos que describen el contenido y la estructura del programa;
- Al menos una de las puertas que indica dónde el programa comienza a ser exhibido (puerto pInicio, línea 15); y
- La terminación del documento (línea 18).

El Cuadro 3.1presenta la estructura básica de un documento NCL.

En un documento NCL se debe incluir obligatoriamente las instrucciones de procesamiento, es decir el encabezado básico del programa. Éstas identifican documentos NCL que contengan sólo los elementos definidos en esta Norma, y la versión NCL con la cual el documento está de acuerdo.

<?xml versión="1.0" encoding="ISO-8859-1 "?>

<ncl id="qualquer string" xmlns="http://www.ncl.org.br/NCL3.0/profileName">



Cuadro 3.1: Estructura Básica de un documento NCL

El atributo *id* del elemento <ncl> puede recibir cualquier cadena de caracteres como valor. El número de versión de una especificación NCL consiste en un número principal y otro secundario separados por un punto. Los números son representados como una cadena de caracteres formada por números decimales, en la cual los ceros a la izquierda se suprimen. El número de versión inicial del estándar es 3.0.

3.4.1. Elementos de NCL y atributos

[2]Como se indicó en la sección anterior los elementos básicos de NCL; en un documento completo de un aplicativo NCL, se definen los hijos que puede contener un elemento <head>, como se listan a continuación:

- <importedDocumentBase>
- <ruleBase>
- <transitionBase>
- <regionBase>
- <descriptorBase>
- <connectorBase>
- <meta>
- <metadata>

[2]ABNT NBR 15606-2. Televisión digital terrestre-codicación de datos y especicaciones de transmisión para radiodifusión digital-parte 2: Ginga-ncl para receptores jos y móviles-lenguaje de aplicación xml para codicación de aplicacione. Asociación Brasilera de Normas Técnicas, 2, 2007. URL: http://www.dtv.org.br.

El elemento <body> se trata como un elemento de contexto NCM y tiene los siguientes elementos como hijos:

- **<port> :** es un punto de interface de un contexto, que ofrece axeso externo a contenidos internos (nodos internos) de un contexto.
- **<property> :** es una propiedad de la media que es utilizado cuando algún atributo de una media es necesario.
- **<media> :** este elemento define los tipos de los objetos multimedia especificando su tipo y ubicación de contenido.
- **<context>:** este elemento es responsable de la definición de los nodos de contextos. Un nodo de contexto NCM es un tipo particular de nodo compuesto de NCM y está definido como un contenedor de nodos y de enlaces.
- **<switch>:** permite la definición de nodos de documentos alternativos (representados por los elementos de <media>, <context> y <switch>) para ser elegidos durante el tiempo de presentación. Las normas de pruebas utilizadas en la elección del componente a ser presentados se definen por el elemento <rule> o <compositeRule> que están agrupados en el elemento <ruleBase>, definido como hijo del elemento <head>.
- **<link> :** este elemento se une (a través del elemento <bind>) a una interface de nodo con un rol de conector, definiendo la relación espacio temporal entre los objetos (representados por los elementos <media>, <context>, <body> o <switch>).

La funcionalidad de las interfaces NCL permite la definición de nodos interfaces que son usados en las relaciones que se hacen con otros nodos interfaces. El elemento <área> permite la definición del ancho del contenido representado en porciones espaciales, porciones temporales o porciones temporales espaciales del contenido de objetos de media (elemento <media>).

El elemento <port> especifica el puerto de un nodo compuesto (elemento <context>, <body> o <switch>) con su respectivo mapeo a una interface de uno de los componentes hijo. El elemento <property> es usado para definir una propiedad o un grupo de propiedades de un nodo como una interface del mismo.

El elemento <switchPort> permite la creación de interfaces de elementos <switch> que son mapeados, es un conjunto de interfaces alternativas del nodo interno del interruptor que se asignan a un conjunto de interfaces alternativos de los nodos internos del interruptor.

El elemento <descriptor> especifica la necesidad de presentación de la información en tiempo y espacio en cada componente del documento. Se debe referir a un elemento <región> para definir la posición inicial de la presentación del elemento <media> (que está asociado a un elemento <descriptor>) en cualquier dispositivo de salida. La definición del <descriptor> debe incluirse en la cabecera del documento, dentro del <descriptorBase>, que especifica el conjunto de descriptores de un documento.

También dentro del elemento <head>, el componente <regionBase> define un grupo de elementos <region>, donde cada uno puede contener un conjunto de elementos de <region> anidados, y así sucesivamente, de forma recursiva, las regiones definen aéreas y son referenciados por el <descriptor>.

Un elemento <causalConnector> representa una relación que puede ser utilizado para la creación de un elemento <link> en los documentos, esta condición debe cumplirse con el fin de desencadenar una acción. Un elemento <link> une (a través de sus elementos <bind>) una interfaz de nodo con conector roles, la definición de una relación espacio-temporal entre los objetos NCL (representado por <media>, <context>, <body> o elementos <switch>).

Para que se pueda comprender de mejor manera el motor de presentación de Ginga este se basa en una adaptación que realizo Soares L. F. G. e Rodrigues R. F., donde el subconjunto de elementos NCL que deberían ser implementadas en TVD y dividida por las funcionalidades donde cada uno se divide en uno o más módulos.

Primero para describir los módulos del NCL es necesario especificar que este es un conjunto de elementos relacionados que representan una unidad utilizable.

3.4.1.1. Área funcional Estructure.

Contiene un módulo llamado Structure que presenta el formato básico que debe tener un documento NCL para poder ser correctamente exhibido, además de tener una mejor uniformidad y claridad de los documentos en general.

Este formato básico (Estructura de un documento NCL), está compuesto por las etiquetas principales que se encuentran definidas dentro de este módulo y son: <ncl>, <head> e <body>, cada una con una funcionalidad de identificar un documento como válido.

Elementos:	**Atributos:**	**Contenido:**
ncl	*id, title, xmlns*	(head?, body?)
head		(importedDocumentBase?, ruleBase?, transitionBase?, regionBase*, descriptorBase?, connectorBase?, meta*, metadata*)
body	*id*	(port| property| media| context| switch| link| meta| metadata)*

Cuadro 3.2: Modulo de Estructura Extendida.

3.4.1.2. Área funcional de Diseño.

En el módulo Layout (Diseño) se especifican los parámetros básicos de las medias que tienen alguna presentación gráfica y cómo estos parámetros se presentan al principio en un documento, estos parámetros son de suma importancia, porque están directamente ligadas a la calidad de los contenidos que el usuario recibirá, puesto que no es posible aplicar escalas o alineaciones utilizando los atributos de este módulo.

La principal etiqueta definido en este módulo es la etiqueta <regionBase>, que describe un conjunto de elementos de la <region>, es decir aquí se asignan los valores al subconjunto {“left”,”top”,”bottom”,”right”,”width”,”height”} que especifican parámetros de presentación, y que van a ser vistos más claramente en el siguiente capítulo.

Elementos:	Atributos:	Contenido:
regionBase	*id, device, region*	(importBase|region)+
region	*id, title, left, right, top, bottom, height, width, zIndex*	(region)*

Cuadro 3.3: Modulo de Diseño Extendida.

3.4.1.3. Área funcional Componente.

Esta funcionalidad está compuesta por dos módulos, denominados Media y Context. En general este módulo define componentes de un documento NCL, llamando componente a todo elemento que pueda ser iniciado para mostrar algún tipo de contenido ya sea éste audio-visual (en el caso del módulo de Media), o lógico (en el caso del módulo de Context)

Media: El módulo se define utilizando la etiqueta <media> y sus atributos para representar de alguna manera algún contenido físico de media.

Elementos:	Atributos:	Contenido:
media	*id, src, refer, instance, type, descriptor*	(area|property)*

Cuadro 3.4: Modulo de Media Extendida

Context: El módulo especifica la etiqueta <context> que es el responsable por la definición de contextos internos en documentos, una etiqueta <context> especifica un conjunto de links y medias que no serán accesibles en el cuerpo global del documento, permitiendo el acceso solamente cuando el contexto fuera inicializado.

Elementos:	Atributos:	Contenido:
context	*id, refer*	(port|property|media|context|link|switch|meta|metadata)*

Cuadro 3.5: Modulo de Contexto Extendido

3.4.1.4. Área funcional de Interfaces

Permite la definición de interfaces de nodos (objetos de media o nodos de composición) que serán utilizadas en relaciones con otras interfaces de nodos. Esta área funcional se divide en cuatro módulos:

1. ***MediaContentAnchor,*** que permite definiciones de anclas de contenido (o área) para nudos de media (elementos <media>);
2. ***CompositeNodeInterface,*** que permite definiciones de puertos para nudos de composición (elementos <context> y <switch>);
3. ***PropertyAnchor,*** que permite la definición de propiedades de nudos como interfaces de nudos; y
4. ***SwitchInterface,*** que permite la definición de interfaces especiales para elementos <switch>.

MediaContentAnchor: Define la etiqueta <area> que es adicionada a las hijas de la etiqueta <media>. La etiqueta <area> define que en este módulo el elemento media genere eventos temporales, tal como se muestran o como el usuario interactúa con el mismo. Otro de los atributos principales de la etiqueta <area> es que ésta se divide entre espacial y temporal, las temporales son "begin" y "end" que determinan los acontecimientos que se están iniciando un cierto tiempo después que las medias han comenzado su presentación en pantalla, la espaciales "coords" que definen que una área de la media ira generando un evento si alguien interactuando con la misma, además de "texto" y "position" que especifican los eventos contenidos en sub-cadenas de medias de texto.

Elementos:	Atributos:	Contenido:
area	*id, coords, begin, end, text, position, first, last, label*	

Cuadro 3.6: Modulo de MediaContentAnchor Extendido

CompositeNodeInterface: define la etiqueta <ports>, ésta etiqueta es de gran importancia para documentos en general, una vez que se especifica como hija directa de la etiqueta <body> se indicaran las medias que iniciaran en la presentación del documento. La etiqueta <ports> básicamente contienen un atributo "component" que debe contener un identificador id de un medio o un contexto existente en el documento, el atributo "id" que es opcional y el atributo "interface" especifican que un evento interno de un componente se ha inicializado.

Elementos:	Atributos:	Contenido:
port	*id, component, interface*	

Cuadro 3.7: Modulo de CompositeNodeInterface Extendido

PropertyAnchor: especifica la etiqueta <property>, que se puede utilizar para definir una propiedad o grupo de propiedades de un nodo, como una de sus interfaces (ancla). El elemento <property> define el atributo name, que indica el nombre de la propiedad o grupo de propiedades, y el atributo value, atributo opcional que define un valor inicial para la propiedad name. El elemento padre no puede tener elementos <property> con los mismos valores para el atributo name.

Elementos:	Atributos:	Contenido:
property	*name, value*	

Cuadro 3.8: Modulo de PropertyAnchor Extendido

SwitchInterface: permite la creación de interfaces de elementos <switch>, que se pueden mapear a un conjunto de interfaces alternativas de nodos internos, permitiendo a un eslabón anclar en el componente elegido cuando el <switch> es procesado. Este módulo introduce el elemento <switchPort>, que contiene un conjunto de elementos de mapeo. Un elemento de mapeo define un camino desde el <switchPort> para una interfaz (atributo interfaz) de uno de los componentes del <switch> (especificados por su atributo component).

Elementos:	Atributos:	Contenido:
switchPort	*id*	mapping+
mapping	*component, interface*	

Cuadro 3.9: Modulo de SwitchInterface Extendido

3.4.1.5. Área funcional de Especificación de Presentación.

Esta funcionalidad específica el módulo Descriptor. Este módulo es importante para la visualización de contenido en los documentos NCL, ya que especifica la etiqueta <descriptor> que contiene toda la información necesaria para que las medias puedan ser correctamente exhibidas. Para que la definición de la etiqueta <descriptor> sea hecha es necesaria la definición de la etiqueta <descriptorBase> que contiene un conjunto de <descriptor> y está definido dentro de la etiqueta <heat> de un documento NCL. La etiqueta tiene un atributo "id" que ha de ser único en un documento, por medio de este atributo y la etiqueta <media> se consigue referenciar la etiqueta <descriptor> a través de su atributo "descriptor".

Elementos:	Atributos:	Contenido:
descriptor	*id, player, explicitDur, region, freeze, moveLeft, moveRight, move Up, moveDown, focusIndex, focusBorderColor, focusBorderWidth, focusBorderTransparency, focusSrc,focusSelSrc, selBorderColor, transIn, transOut*	(descriptorParam)*
descriptorParam	*name, value*	
descriptorBase	*id*	(importBase|descriptor| descriptorSwitch)+

Cuadro 3.10: Modulo del Descriptor Extendido

3.4.1.6. Área fuancional Linking.

El área funcional Linking define el módulo Linking, responsable de la definición de los eslabones, que utilizan conectores. Un elemento <enlace> puede tener un atributo id, que es identificado dentro del documento y debe tener obligatoriamente un atributo xconnector, que se refiere al URI de un conector hipermedia. La referencia debe tener obligatoriamente el formato *alias#connector_id*, o *documentURI_value#connector_id*, para conectores definidos en un documento externo, o simplemente connector_id, para conectores definidos en el propio documento.

El elemento <link> contiene elementos-hijos denominados <bind>, que permiten asociar nodos a papeles (roles) del conector. Para hacer esta asociación, un elemento <bind> tiene cuatro atributos básicos.

- El primero se denomina *role*, que se utiliza para hacer referencia a un papel del conector.
- El segundo se denomina *component*, que se utiliza para identificar el nodo.
- El tercero es un atributo opcional denominado *interfaz*, usado para hacer referencia a una interfaz del nodo.

- El cuarto es un atributo opcional denominado *descriptor*, usado para hacer referencia a un descriptor a ser asociado con el nudo, como se definio en el módulo Descriptor.

Los elementos del módulo Linking, sus atributos y sus elementos-hijos deben estar de acuerdo obligatoriamente con la Tabla 3.11.

Elementos:	**Atributos:**	**Contenido:**
bind	*role, component, interface, descriptor*	(bindParam)*
bindParam	*name, value*	
linkParam	*name, value*	
link	*id, xconnector*	(linkParam*, bind+)

Cuadro 3.11: Modulo Linking Extendido

3.4.1.7. Área funcional Conectors.

Esta es la funcionalidad que define el conjunto más grande de los módulos en todo el bloque NCL, puesto que no es el contenido completo del idioma NCL puede establecer hechos de sincronización y interacción con el contenido de la hipermedia. El módulo más simple pero no menos importante se llama Causal-Connector.

Las relaciones NCL se basaron en los hechos, que pueden clasificarse en tres tipos, que se muestran en el Cuadro 3.12.

Tipo de evento	**Descripción**
presentation	Este tipo de evento se desencadena por componentes NCL (media, switch y context), cuando se le da a estos elementos, cualquier instrucción que administra algunas acciones en sus componentes internos. Eventos de este tipo también son generados por elementos <area> definidos dentro de elementos <media>.
selection	Este tipo de evento se genera cuando la tecla ENTER es presiona y cuando sobre el selector de medios del formateador está sobre algún componente.
attribution	Este tipo de evento ocurrirá cada vez que el valor de algún atributo de cualquier elemento ncl es cambiado por algún tipo de vínculo (link).

Cuadro 3.12: Tabla de eventos generada por NCL

Los elementos del módulo CausalConnectorFunctionality, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con el Cuadro3.13

Elementos:	Atributos:	Contenido:
causalConnector	*id*	(connectorParam*, (simpleCondition | compoundCondition), (simpleAction | compoundAction))
connectorParam	*name, type*	
simpleCondition	*role, delay, eventType, key, transition, min, max, qualifier*	
compoundCondition	*operator, delay*	((simpleCondition | compoundCondition)+, (assessmentStatement | compoundStatement)*)
simpleAction	*role, delay, event Type, action Type, value, min, max, qualifier, repeat, repeatDelay, duration, by*	
compoundAction	operator, delay	(simpleAction | compoundAction)+
assessmentStatement	comparator	(attributeAssessment, (attributeAssessment | valueAssessment)
attributeAssessment	role, eventType, key, attributeType, offset	
valueAssessment	value	
compoundStatement	operator, isNegated	(assessmentStatement | compoundStatement)+

Cuadro 3.13: Modulo del CausalConnectorFunctionality Extendido

3.4.1.8. Área funcional de Control de Presentación.

Especifica alternativas de contenido y presentación para un documento. Esta área funcional se divide en cuatro módulos, denominados:

1. TestRule.
2. TestRuleUse.
3. ContentControl.
4. DescriptorControl.

TestRule: describe un conjunto de reglas en la etiqueta <ruleBase>, esta etiqueta debe ser definida como hija de la etiqueta <heat> en un documento, Esas reglas pueden ser simples definidas por el elemento <rule>, o compuestas definidas por el elemento <compositeRule>. Las reglas simples definen un identificador (atributo id), una variable (atributo var), un valor (atributo value) y un comparador (atributo comparator) relacionando la variable a un valor.

Elementos:	Atributos:	Contenido:
ruleBase	*id*	(importBase|rule|compositeRule)+
rule	*id, var, comparator, value*	
compositeRule	*id, operator*	(rule | compositeRule)+

Cuadro 3.14: Modulo del TestRule Extendido

TestRuleUse: define el elemento <bindRule>, que se utiliza para asociar reglas con componentes de un elemento <switch> o <descriptorSwitch>, a través de sus atributos rule y constituent, respectivamente.

Elementos:	Atributos:	Contenido:
bindRule	*constituent, rule*	

Cuadro 3.15: Módulo TestRuleUse extendido

ContentControl: explica la etiqueta <switch> que permite la definición de nodos alternativos a ser elegidos en tiempo de presentación del documento. Las reglas de prueba utilizadas para escoger el componente del switch a ser presentado se definen por el módulo TestRule, o son reglas de prueba específicamente definidas e incorporadas en una implementación del formateador NCL. El módulo ContentControl también puntualiza el elemento <defaultComponent>, cuyo atributo component (del tipo IDREF) identifica el elemento (default) que debe ser obligatoriamente seleccionado si ninguna de las reglas bindRule es evaluada como verdadera.

Elementos:	Atributos:	Contenido:
switch	*id, refer*	defaultComponent?, (switchPort | bindRule | media | context | switch)*)
defaultComponent	*component*	

Cuadro 3.16: Módulo ContentControl extendido

DescriptorControl: define el elemento <descriptorSwitch>, que contiene un conjunto de descriptores alternativos a ser asociado a un objeto. Análogamente al elemento <switch>, la elección <descriptorSwitch> se realiza en tiempo de presentación, utilizando reglas de prueba descritas por el módulo TestRule, o reglas de prueba específicamente detalladas e incorporadas en una implementación del formateador NCL. El módulo DescriptorControl también precisa el elemento <defaultDescriptor>, cuyo atributo descriptor (del tipo IDREF) identifica el elemento (default) que debe ser obligatoriamente seleccionado cuando ninguna de las reglas bindRule es evaluada como verdadera.

Elementos:	Atributos:	Contenido:
descriptorSwitch	*id*	defaultDescriptor?, (bindRule | descriptor)*)
defaultDescriptor	*descriptor*	

Cuadro 3.17: Módulo DescriptorControl extendido

3.4.1.9. Área funcional Timing

Define el módulo Timing que permite la descripción de atributos temporales para componentes de un documento, este módulo detalla atributos para especificar qué pasa con un objeto al final de su presentación (freeze) y la duración ideal de un objeto (explicitDur). Estos atributos pueden ser incorporados por los elementos <descriptor>.

3.4.1.10. Área funcional Reuse

NCL permite una gran reutilización de sus elementos mediante el área funcional Reuse que se divide en tres módulos:

1. Import.
2. EntityReuse.
3. ExtendedEntityReuse.

Import: para permitir que una base de entidades sea incorporada a otra ya existente, el módulo Import define el elemento <importBase>, que tiene dos atributos: DocumentURI y alias. El atributo documentURI se refiere a un URI correspondiente al documento NCL conteniendo la base a ser importada. El atributo alias específica un nombre a ser utilizado como código cuando sea necesario referirse a elementos de esa base importada. EL nombre del atributo alias debe ser obligatoriamente único en un documento y su alcance está supeditado al documento que lo definió.

Los elementos del módulo Import, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla 3.18

Elementos:	**Atributos:**	**Contenido:**
importBase	*alias, documentURI, región*	
imported DocumentBase	*id*	(importNCL)+
importNCL	*alias, documentURI*	

Cuadro 3.18: Módulo Import extendido

EntityReuse: permite que un elemento NCL sea reutilizado, aquí se define el atributo refer, que hace referencia a un elemento id que será reusado. Sólo <media>, <context>, <body> o <switch> se pueden reutilizar. Un elemento que hace referencia a otro elemento no puede ser reutilizado; es decir, su *id* no puede ser el valor de un atributo *refer*.

ExtendedEntityReuse: Otro atributo denominado "newinstance" es definido por este módulo para permitir que grupos de nodos sean referenciados como uno solo. Este atributo es exclusivo para elementos <medias> y puede tomar valores "true" o "false", en el caso de "true" cuando la media se inicializa se creará una nueva instancia de la presentación de la misma, si es "false" los eventos desencadenados por esa media serán compartidos tanto por el elemento <media> referenciado, y por los elementos que referencian aquel elemento <media> en el documento actual.

El atributo *instance* se define en el módulo ExtendedEntityReuse y tiene "new" como su valor default de string. El elemento referido y el elemento que le hace referencia deben ser obligatoriamente considerados el mismo, con relación a sus estructuras de datos.

3.4.1.11. Área funcional Animación

Una animación es una combinación de dos factores: soporte al dibujo y al movimiento del objeto, o más propiamente, soporte para la alteración del objeto

en función del tiempo. En NCL no se pueden crear objetos de media pero se puede utilizar como un formato de escalonamiento y orquestación. Eso significa que NCL no se puede utilizar para hacer dibujos animados, pero se puede utilizar para exhibir objetos de dibujo animado en el contexto de una presentación general y para alterar las propiedades de sincronización y exhibición de un objeto de un dibujo animado (o cualquier otro) globalmente, mientras el objeto esté siendo exhibido.

El área funcional Animación define el módulo Animation que suministra las extensiones necesarias para describir qué pasa cuando el valor de una propiedad de un nodo es alterado. Básicamente, el módulo describe atributos que pueden ser incorporados por los elementos <simpleAction> de un conector, si su valor eventType es "attribution". Dos nuevos atributos son definidos: *duration* y *by*

Al atribuir un nuevo valor para una propiedad, la alteración es instantánea por default (duration="0"), pero la alteración también se puede realizar durante un período explícitamente declarado, especificado por el atributo *duration.*

Además de ello, al atribuir un nuevo valor a una propiedad, la alteración del valor antiguo por el nuevo puede ser lineal por default (by="indefinite"), o hecha paso a paso, con el paso especificado por el atributo *by.* La combinación de las definiciones de los atributos *duration* y *by* ofrece la descripción de cómo (de forma discreta o lineal) la alteración se debe realizar obligatoriamente y su intervalo de transformación.

3.4.1.12. Área funcional SMIL Transition Effects

El área funcional Transition Effects se divide en dos módulos:

1. TransitionBase.
2. Transition.

TransitionBase: es definido por la NCL 3.0 y consiste en un elemento <transitionBase> que especifica un conjunto de efectos de transición y se debe definir obligatoriamente como elemento-hijo del elemento <head>. El elemento <transitionBase>, sus elementos-hijos y sus atributos deben estar de acuerdo obligatoriamente con la Tabla3.19.

Elementos:	Atributos:	Contenido:
transitionBase	*id*	(importBase, transition)+

Cuadro 3.19: Módulo TransitionBase extendido

Transition: está basado en las especificaciones SMIL 2.1 y solamente tiene un elemento llamado <transition>.

En el perfil NCL 3.0 TVD, el elemento <transition> es especificado en el elemento <transitionBase> y permite que sea definido un estándar (template) de transición. Cada elemento <transition> define un estándar único de transición y debe tener obligatoriamente un atributo id para que pueda ser referido dentro de un elemento <descriptor>.

Siete atributos del elemento <transition> son derivados de la especificación del módulo BasicTransitions de SMIL:

1. type;
2. subtype;
3. dur;
4. startProgress;
5. end-Progress;
6. direction;
7. fadeColor.

Las transiciones se clasifican de acuerdo a dos niveles: tipos y subtipos. Cada tipo describe un grupo de transiciones que están íntimamente relacionadas. Dentro de ese tipo, cada una de los cambios individuales se asocia a un subtipo que enfatiza las características distintas de las mismas.

El módulo Transition también define los atributos a ser utilizados en los elementos <descriptor>, para los estándares de transición definidos por los elementos <transition>: atributos transIn y transOut. Las especificadas con un atributo transIn empezarán en el comienzo de la duración activa de los elementos de media (cuando la presentación del objeto empieza). Los cambios detallados con un atributo transOut iniciaran cuando termine la duración activa de los elementos de media (cuando la presentación del objeto pasa del estado ocurriendo a terminado).

Los atributos transIn y transOut son agregados a los elementos <descriptor>. El valor default de ambos atributos es una string vacía, que indica que, obligatoriamente, ninguna transición se debe realizar.

3.4.1.13. Área funcional Metainformation

Una metainformación contiene informaciones sobre el contenido utilizado o exhibido. Esta área está compuesta por el módulo metainformation, derivado del módulo Metainformation SMIL. El elemento <meta> especifica un único par de propiedad/valor en los atributos name y content, respectivamente.

El elemento <metadata> contiene informaciones que también se relacionan con la metainformación del documento. Actúa como el elemento raíz del árbol RDF. El elemento <metadata> puede tener como elementos-hijos: Elementos RDF y sus subelementos.

Elementos:	**Atributos:**	**Contenido:**
meta	*name, content*	(importBase, transition)+
metadata	*empty*	RDF tree

Cuadro 3.20: Módulo Meta-Information extendido

3.5. Herramientas.

Todas las herramientas exploradas en esta sección son gratuitas y de código abierto que evolucionan constantemente. El objetivo de utilizar las mismas para el desarrollo de aplicaciones interactivas Ginga-NCL en una PC es poder disponer de un entorno similar al middleware Ginga, que se encontrará habitualmente ya

instalado en los receptores, pudiendo ser estos adaptadores SetTopBoxes o televisores integrados, obteniendo así una plataforma de simulación del funcionamiento real de dichas aplicaciones [23][17].

Hay dos formas de usar Ginga en una PC: realizando una instalación nativa o levantando una máquina virtual con Ginga pre-instalado. La diferencia entre las dos formas de ejecución se encuentra en que con una máquina virtual se puede simular un sistema mediante un software de virtualización dentro del sistema ya existente (Windows, Linux, MacOS), la ventaja de esta forma de ejecución es su facilidad para poner a funcionar el ambiente, pero presenta un inconveniente ya que es mucho más difícil manejar archivos entre dos sistemas operativos [24].

La instalación de forma nativa permite ejecutar Ginga en el sistema operativo existente, esta implementación mejora la velocidad de la PC, y ofrece una simplicidad en el manejo de archivos. La desventaja es que el proceso de instalación es mucho más complicado en relación al método de la máquina virtual ya que se requieren conocimientos de Linux y aún no posee automatización.

Para el desarrollo de aplicaciones interactivas Ginga se utilizan dos tipos de herramientas:

1. De desarrollo
2. De presentación.

3.5.1. Herramientas de desarrollo.

En la actualidad existe un gran número de herramientas para desarrollar aplicaciones interactivas en el entorno Ginga-NCL. Es posible crear programas interactivos en cualquier editor XML (puede usarse hasta el mismo bloc de notas). Esta sección se centrara en las herramientas de libre distribución, que soportan plataformas Windows y Linux, además de ser las más utilizadas para la generación de aplicaciones interactivas, diferenciándose entre sí por el nivel de conocimientos de programación que debe aplicarse para su manejo, además de la complejidad de la aplicación interactiva que se desea crear, estas herramientas son dos:

- Composer.
- Eclipse NCL.

3.5.1.1. Composer.

Es una herramienta de software libre diseñada para la autoría hipermedia que facilita y agiliza la construcción de documentos NCL para TV digital interactiva [6], fue desarrollada por el Laboratorio de Telemidia del departamento de informática de la PU-Rio. Es de edición gráfica, las abstracciones se definen en diversos tipos de visiones que permiten simular un tipo específico de edición, la versión actual de composer permite al usuario trabajar con 4 tipos de visiones [10]:

1. Visión Estructural.
2. Visión Temporal.
3. Visión de Diseño o Esquema.

4. Visión Textual

La Figura 3.10representa el entorno Composer e las diversas visiones en las que el usuario puede trabajar.

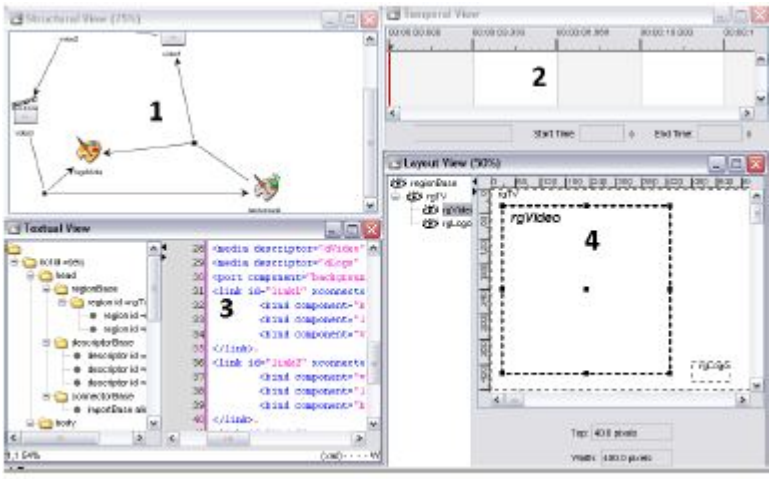


Figura 3.10: Herramientas de autoría Composer
Fuente: Captura de la pantalla del software composer

- ***Visión Estructural:*** Permite al usuario crear una estructura lógica de los aplicativos NCL, tiene sus objetos de multimedia conectadas entre sí por enlaces que responden a eventos y se asocia a los estados. El programador puede crear, editar y eliminar composiciones, objetos de medios y enlaces. En la Figura 3.11 3.11, se muestra la visión estructural, donde se crean los nodos de media, enlaces y sus propiedades.

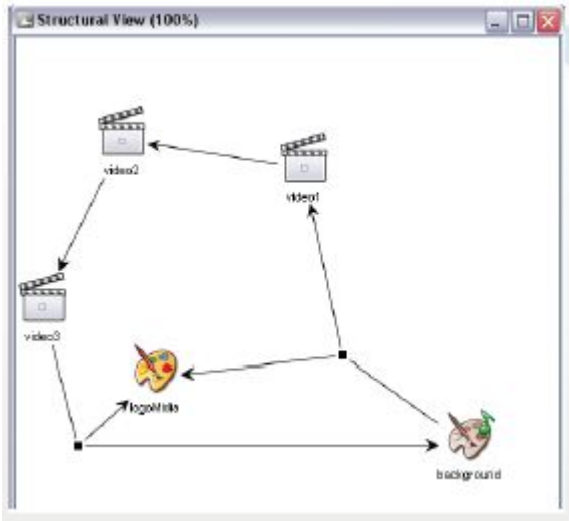


Figura 3.11: Visión Estructural
Fuente: Captura de la pantalla del software composer

- ***Visión Temporal:*** ilustra el sincronismo en el tiempo entre los nodos media y las oportunidades de interactividad. Esta visión establece relaciones temporales con anclas transitorias presentes en los medios (audio, vídeo e

imágenes), que son las entidades claves para la presentación del documento en la línea del tiempo. La Figura 3.12presenta la visión temporal de un documento hipermedia que exhibe una imagen iconeInte-racao durante un determinado segmento de la presentación de la media videoPrinc.

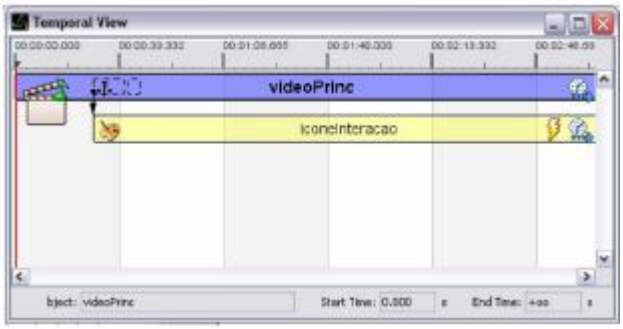


Figura 3.12: Visión Temporal
Fuente: Captura de la pantalla del software composer

- ***Visión de Diseño o Esquema:*** permite la creación y configuración de zonas donde los medios serán presentados, facilitándole al autor crear, editar y eliminar algunas de ellas. Además permite crear sub-regiones o regiones superpuestas. Las regiones pueden ser definidas con un tamaño de pixel o de tamaños relativos o porcentaje con respecto a la pantalla completa. En la Figura 3.13 se muestra un ejemplo de la visión con dos regiones (rgVideo, rgTexto) creadas donde se presentaran los vídeos correspondientes.

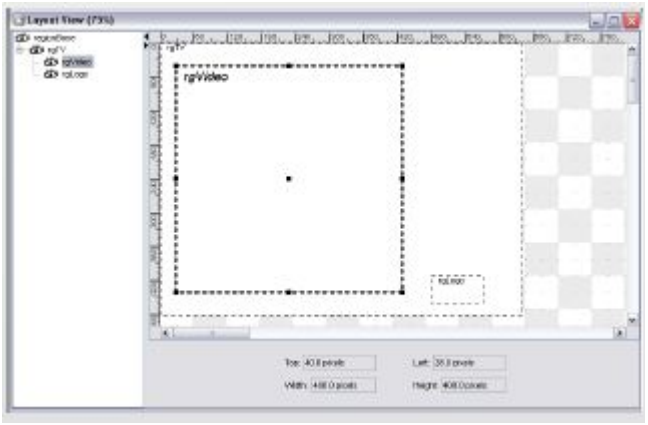


Figura 3.13: Visión de Diseño o Esquema
Fuente: Captura de la pantalla del software composer

- ***Visión Textual:*** La vista textual, presenta el código NCL en sí, aquí el usuario puede crear directamente el código NCL como un editor de texto común. En la Figura 3.14 se observa la estructura del código NCL que está basado en el formato XML y también es considerado como un documento XHTML.

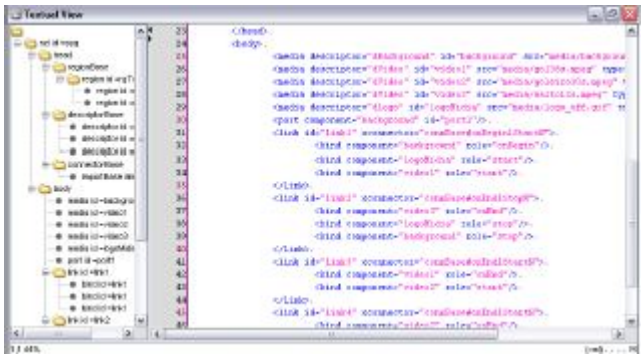


Figura 3.14: Visión Textual
Fuente: Captura de la pantalla software composer.

Estas visiones se sincronizan con el fin de ofrecer un entorno integrado de autoría [25], es decir todas las secciones pueden ser editadas con actualización instantánea de las demás, por ejemplo una alteración en el código NCL es reflejada instantáneamente en la visión temporal, espacial y estructural del documento.

La presentación de diversas visiones fomenta a que el autor tenga una mayor intuición en el desenvolvimiento de las aplicaciones. En resumen, el sistema está diseñado para facilitar y acelerar la creación de aplicaciones interactivas centradas en la televisión digital, o al menos reducir parte de la complejidad de la programación en NCL a través de este medio ambiente de autoría que requiere un nivel de conocimientos del lenguaje NCL mínimo.

El entorno composer se encuentra todavía en la etapa de desarrollo y, como tal, es una herramienta muy inestable. Cuando se utiliza ésta es recomendable guardar el archivo del proyecto frecuentemente, ya que se han reportado problemas de deterioro del mismo. Además de esto, el composer no tiene soporte para todas las características de la norma ABNT NBR 15606-2[16] y utiliza Ginga Emulador para mostrar los documentos NCL, por lo tanto está sujeto a las mismas restricciones.

Otra de las deficiencias fundamentales del entorno composer es la falta de soporte para el lenguaje Lua, problemas con las disposiciones de los componentes de interfaz gráfica, poca especificación de los errores encontrados, poco poder de manipulación de los archivos del proyecto y bajo rendimiento.

El entorno composer es limitado y cuando se precisa desarrollar aplicaciones más complejas es necesario utilizar la herramienta Eclipse sobre el que se centra la generación del manual.

3.5.1.2. Eclipse NCL.

Eclipse es un IDE de programación para desarrollar aplicaciones en varios lenguajes (C, java, php...). Esta plataforma es extremadamente modular ya que por medio de la adición de plugins se puede extender la funcionalidad, es decir provee la infraestructura para edición de productos a partir de estos. La característica fundamental que fomenta la adaptación de ésta a nuevos entornos de desarrollo es que posee una licencia de tipo EPL de código abierto que permite, usar, modificar, copiar y distribuir nuevas versiones del producto.

Para la creación del lenguaje NCL la Universidad Federal do Maranhão desarrollo NCL-Eclipse que es un editor XML/NCL creado como un *plugin* para la

plataforma Eclipse, permite que todo este ambiente sea reutilizado y facilita la integración con otras herramientas de desenvolvimiento para la TV Digital, como, por ejemplo, Lua Eclipse para la plataforma Eclipse, permite que todo ese ambiente se ha reutilizado y facilita la integración con otras herramientas de desenvolvimiento para la TV Digital, como, por ejemplo, Lua Eclipse.

Estos complementos auxilian y agilizan la creación de aplicaciones, permitiendo que facilidades extras de Eclipse sean reutilizadas e integradas con otras herramientas de desenvolvimiento. De entre ellas se pueden citar:

- Soporte para navegación hipertextual.
- Sugestion de parametros y auto-completar.
- Sugestión en pop-up para selecionar archivos.
- Cierre automático de los elementos.
- Validación de la sintaxis y marcado de los errores en el documento.
- Formateo e plantificación de código.

El NCL Eclipse reusa el NCL validator como un objetivo de idéntificar y marcar erros en documentos NCL, en tiempo de autoría. En la Figura 3.15 se muestra la pantalla de presentación del software Eclipse.

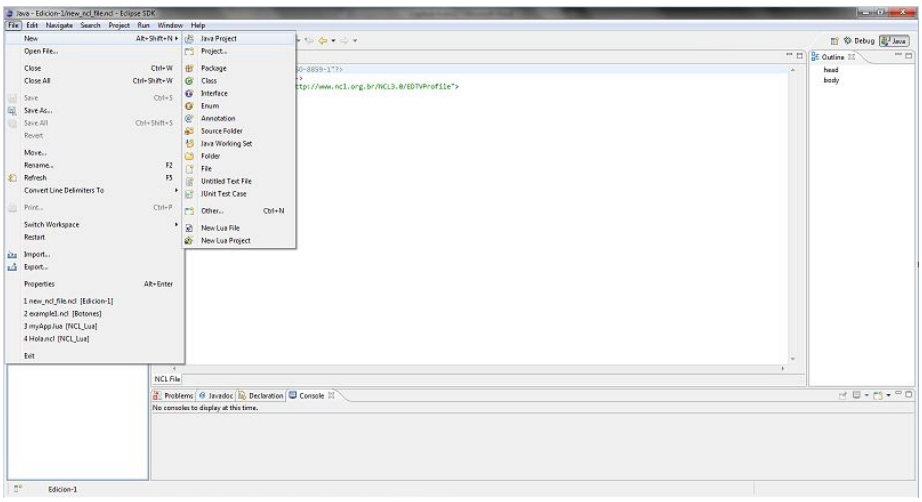


Figura 3.15: Pantalla de Eclipse
Fuente: Capturada de la pantalla software Eclipse.

El entorno LuaEclipse es una colección de plugins desarrollados para Eclipse, que juntos forman un IDE para el desarrollo de aplicaciones Lua. Este entorno de programación, posibilita la edición scripts Lua con sintaxis resaltada, complementación código automático, recopilación de errores, la ejecución del script viene pre configurada, además de las herramientas disponibles para la plataforma Eclipse.

El principal objetivo del LuaEclipse es que las nuevas herramientas pueden ser desarrolladas a partir de la extensión de la arquitectura que forma la plataforma Eclipse y LuaEclipse, permitiendo la ampliación de sus capacidades.

Debido a la gran cantidad de desarrollo de plugins para Eclipse esta herramienta usualmente es utilizada como base para las demás soluciones que buscan

facilitar el desenvolvimiento de aplicaciones declarativas, ya que permiten la integración de lenguaje procedural al lenguaje declarativo NCL, convirtiéndose así en la herramienta con más potencia para el desarrollo de contenido interactivo en la actualidad.

3.5.2. Herramientas de presentación

Para la visualización de las aplicaciones en entorno Ginga-NCL, existen dos aplicativos más usadas:

- Emulador Ginga NCL.
- Set Top Box Virtual.

3.5.2.1. Emulador Ginga-NCL

Este software es la herramienta más fácil de usar y más accesible, por medio de ésta es posible abrir un archivo NCL, ejecutar acciones mediante el control remoto interactivo que es activado con el mouse. Para que el emulador pueda funcionar es necesario tener instalada la Máquina Virtual de Java (JVM), ya que está desarrollado sobre la misma, sin embargo por estar basada en Java posee una serie de limitaciones. Las principales son [2]:

- La secuencia de los vídeos no es lineal. Cuando uno de estos se ve afectado del bucle o cuando varios de estos están vinculados, se ha producido una caída en la secuencia de los vídeos, ya que en la última parte del primero no se encuentra conectado perfectamente al primer fotograma del siguiente.
- No hay soporte para la transparencia. Algunas interfaces pueden ser desarrollados en software como Adobe Photoshop o GIMP y se guardan en el formato PNG, con niveles de transparencia. Estos parecen chapados, de color gris, en el emulador.
- Además, el emulador no posee soporte para el lenguaje Lua.

El emulador de Ginga NCL está incrustado en la herramienta composer y puede ser instalado como plugin de Eclipse IDE, que facilita el desarrollo y prueba de aplicaciones (ver Figura 3.16).

3.5.2.2. Virtual SetTopBox.

El Virtual SetTopBox (VSTB) es una implementación C++ del middleware Ginga emulada en una máquina virtual para VMWare que posee instalada una imagen de Fedora Linux. El VSTB simula fielmente el ambiente de presentación de aplicaciones declarativas, posee un mejor rendimiento y un entorno más parecido a una aplicación incrustado en las STB, que el que es producido por el emulador.

La máquina virtual de ubuntu-server10.10-ginga-i386 fue creado y configurado por el personal del Laboratorio de la PUC-Río de software TELEMIDIA utilizando VMWare Workstation 7. La instalación ha sido optimizada para incluir solamente los paquetes de software esenciales para el desarrollo del middleware Ginga y la ejecución de Ginga-NCL versión C++. Para ejecutar una aplicación sobre el VSTB, se debe abrir una conexión utilizando el protocolo SSH, por medio de un software de acceso remoto[3].

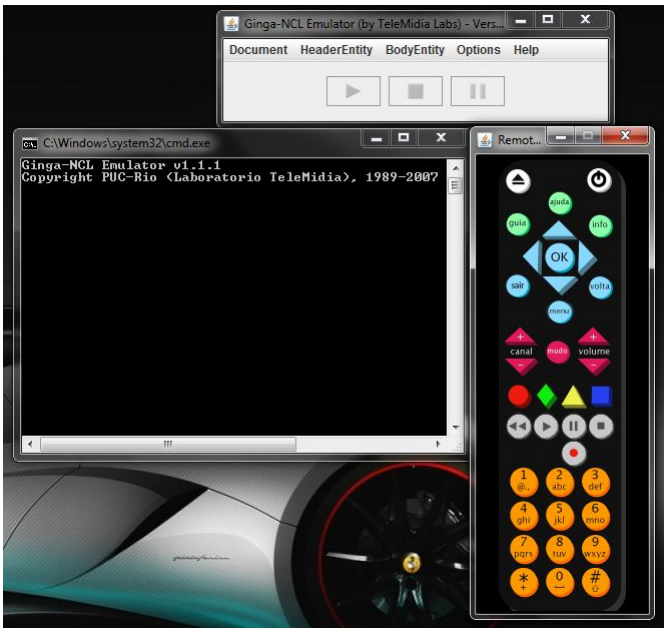


Figura 3.16: Ginga-NCL Emulator
Fuente: Captura de la pantalla del software emulator.

Como se indica en la Figura 3.17, después de cargar la imagen se tiene el VSTB listo para ser utilizado [26].

Los principales paquetes de software están instalados en la VSTB son:

- gingancl cpp-0.12.1
- Lua 5.1 / 2.0.2 luasocket
- kernel 2.6.35 cadena de herramientas GNU (gcc 4.4.4, glibc 2.12 a 1)
- directfb 1.4.11
- fusionsound 1.1.1 xine-lib 1.1.17

Esta herramienta posee suporte de ejecución de los script en LUA y soporte a la transparencia. La máquina virtual tradicional está limitado a la resolución 640x480, lo cual impide probar sus aplicaciones para la alta definición, pero la nueva version ofrece ahora la posibilidad de elegir la resolución de la pantalla entre 6 opciones preconfiguradas. La elección de la misma se hace en el momento del arranque de la máquina virtual y debe decidirse en el seguna la capacidad de procesamiento del CPU host de la estación receptora. La virtualización de las aplicaciones en VSTB es un poco mas demorada que en el emulador para Windows, pero su funcionamiento es muy superior.

Entre los requisitos principales para el funcionamiento de Ginga-NCL Virtual Set Top Box son:

Requisitos de Hardware:

- Arquitectura Intel.
- Promedio Core Duo.
- Memoria RAM recomendado de 2 GB.

Figura 3.17: Ginga-NCL VSTB
Fuente: Captura de la pantalla del software VSTB.

- Placa Aceleradora de Vídeo con 64Mb.
- Disco duro con 10 GB (depende de la cantidad de vídeos almacenados).

Requisitos de Software:

- Depende del Sistema Operativo en uso: Windows XP, Linux, Mac OS X.
- Software de virtualización: VMWare Player (Windows o Linux), VMWare Workstation (Windows/Linux), VMWare Fusion (Mac OS X).

Capítulo 4

Manual de programación del Middleware Ginga NCL

4.1. Creacion de contenido interactivo en Ginga-NCL.

Para la creación de contenido interactivo existe el lenguaje NCL, que se emplea en este manual para realizar ejemplos de documentos hipermedia. Las funciones del lenguaje se introducirán gradualmente, apoyándose en ejemplos. Éstos se basarán en la versión actual de la máquina de presentación Ginga-NCL, que interpreta un documento NCL y exhibe el programa audiovisual interactivo representado como en un receptor de TVD por medio de un VSTB. Las secciones siguientes proporcionan una introducción a varios elementos del modelo NCM.

4.1.1. Definición de la presentación.

En esta sección se presentan los elementos que componen el programa cuando empieza a ser creado, por medio de las regiones y los descriptores.

4.1.1.1. Regiones.

Las regiones establecen las áreas de la pantalla donde los elementos del programa (vídeo, texto, imagen, etc) podrán ser presentados. Este documento NCL define la posición inicial de un elemento. Para que un documento sea presentado, al menos una región debe ser definida, siendo esta región la que definirá la dimensión de la pantalla donde el documento NCL será visualizado.

La región se define en la cabecera del documento NCL en la base de las regiones, el elemento <*regionBase*> está definido por la etiqueta <*region*>. Una región puede contener también otras regiones que permitan una mayor organización del documento NCL, estas regiones hijas heredan por default los atributos de la región padre.

En el caso específico de las declaracion de regiones, tenemos:

- Region sin regiones anidada.

  <region ... atributos ... />

- Region com unas o mas regiones anidadas.

  <region ... atributos ... >
  <region ... atributos ... />

```
    <region ... atributos ... />
</region>
```

Toda región tiene un identificador único representada por el atributo ***id***. Este identificador será utilizado por otros elementos del documento NCL siempre que una referencia para la región sea necesaria. Una región puede tener los atributos enumerados a continuación y pueden ser vistos en la Figura 4.1.

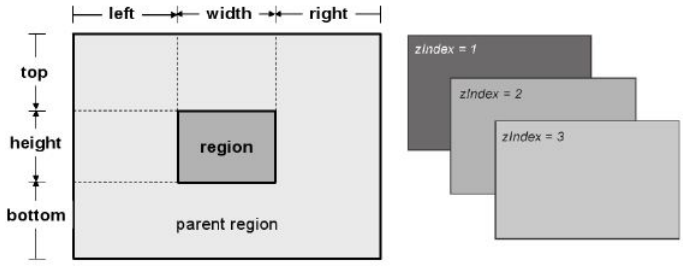


Figura 4.1: Atributos de posicionamiento y dimensionamiento de una region.
Fuente: http://www.softwarepublico.gov.br

- ***id:*** identificador de la región. Este valor, que debe ser único, se utilizará cada vez que sea necesario referirse a la región.
- ***title:*** es el título de una región, que en caso de ser exhibida como una moldura, éste será el que aparece como título de la ventana correspondiente.
- ***left:*** define la coordenada horizontal izquierda de la región. Ésta puede ser indicada en valores absolutos (sin la necesidad de señalar la unidad de medida utilizada) o por medio de porcentajes. Este valor tiene como referencia a la región padre, en el caso en que la región en cuestión no está contenida en otra pantalla del dispositivo de exhibición.
- ***top:*** describe la coordenada vertical superior de la región. Ésta se puede indicar en valores absolutos (sin la necesidad de describir la unidad de medida utilizada) o porcentual. Este valor tiene como referencia a la región padre, en el caso en que la región en cuestión no está contenida en otra pantalla del dispositivo de exhibición.
- ***right:*** define la coordenada horizontal derecha de la región. Se puede indicar en valores absolutos (sin la necesidad de indicar la unidad de medida utilizada) o porcentual. Este valor tiene como referencia a la región padre, en el caso en que la región en cuestión no está contenida en otra pantalla del dispositivo de exhibición.
- ***bottom:*** define la coordenada vertical inferior de la región. Se puede indicar en valores absolutos (sin la necesidad de indicar la unidad de medida utilizada) o porcentual. Este valor tiene como referencia a la región padre, en el caso en que la región en cuestión no está contenida en otra pantalla del dispositivo de exhibición.
- ***width:*** define la dimensión horizontal de la región. Cabe señalar que ésta también podría especificarse mediante los atributos de *left* y *right*. La elección de cómo declarar la dimensión de la región se deja al programador. Sin embargo, en el caso de definir los atributos de *left* y *width*, el valor del atributo *right* no se considera.

- ***height:*** establece la dimensión vertical de la región. Cabe señalar que ésta también podría definirse mediante los atributos *top* y *bottom*. La elección de la forma de establecer los tamaños de la región se deja al programador. Sin embargo, en el caso de especificar los atributos *top* y *height*, el valor del atributo *bottom* no es considera.

- ***zIndex:*** número entre 0 y 255 que define la superposición de las capas. De acuerdo con el valor contenido en este atributo, una región se presentará "arriba" de otras regiones con *zIndex* menor y "abajo" de otras regiones con *zIndex* mayor. En el caso que dos regiones posean el mismo valor *zIndex* definido, una midia se presenta en cada región, hay dos posibilidades para ordenar las regiones: las media presentada por ultimo será presentada "arriba" de la anterior, si ambas se presentan al mismo tiempo, el orden será elegido por el formateador (ejecutador)

4.1.1.2. Descriptores.

Especifican cómo los nodos media serán presentados en las áreas de la pantalla. Para ello, un descriptor debe indicar la región a la que está relacionado y explicar detalles de su presentación. Se especifican en la cabecera del documento NCL sobre la base de descriptores, elemento <*descriptorBase*> y definido por la etiqueta <*descriptor*>.

Un descriptor tiene un identificador único ***id***. . Además este tiene que indicar la región a la que está asociado a través del atributo ***región*** que es la regin en la que será presentado. Se ocupan solamente para relacionar un nodo media con una región en la pantalla. El descriptor puede ser utilizado para definir cómo será realizada la navegación en la pantalla, atravez de los botones en el control remoto y cambiar la forma en que un elemento es presentado, cambiando su tiempo de exhibición o su característica de transparencia.

Un descriptor tiene los siguientes atributos:

- ***id:*** identificador del descriptor. Este atributo contiene un valor único que será utilizado para referenciar al mismo.

- ***región:*** determina la región a la que el descriptor está relacionado, presentando el nodo en la región correspondiente. Si el elemento no posee contenido visual, no será necesario definir una región.

- ***player:*** identifica la herramienta de presentación que se empleara para mostrar el objeto multimedia asociado al descriptor. Este atributo es opcional. Cuando se omite, el programa interprete procura buscar esta herramienta por defecto en función al tipo de objeto multimedia.

- ***freeze:*** permite identificar lo que sucede al terminar la presentación de un objeto multimedia asociado al descriptor. En un video, el valor de "*true*" indica que el último valor debe permanecer congelado.

- ***explicitDur:*** determina la duración de la presentación del elemento asociado con el descriptor, es decir el tiempo de presentación del objeto multimedia. Cuando el atributo *explicitDur* no está especificado, se tendrá en cuenta el periodo por defecto de cada elemento. Si el componente es un texto o una imagen, el tiempo de presentación será infinito. En el caso que un elemento

posea una duración, será considerada su duración propia. EL valor definido en este atributo debe estar en segundos (considerando la unidad "*s*"), para describirlo se debe escribir el valor numérico seguido con la letra "*s*". Para que el programa pueda interpretar este atributo no se deberá considerar para archivos multimedia continuos, pero se puede utilizar este atributo para modificar la duración de un nodo.

- ***focusIndex:*** establece un índice a ser utilizado para la navegación entre los objetos multimedia presentados en la pantalla. Si un descriptor no define éste, el objeto multimedia no podrá recibir el foco de navegación. En el inicio de ejecución del programa, el foco pasa para el elemento asociado al descriptor de menor índice. Una observación, el valor del *focusIndex* puede no ser numérico, en cuyo caso será elegido el índice lexicográficamente menor.

- ***moveLeft:*** establece el descriptor, a través de su índice, que recibirá el foco cuando la "*Flecha izquierda*" del control remoto es presionado. Esta asignación sólo se realiza cuando el descriptor que lo define esté con el foco.

- ***moveRight:*** establece el descriptor, a través de su índice, que recibirá el foco cuando la "*Flecha derecha*" del control remoto es presionado. Esta asignación sólo se realiza cuando el descriptor que lo define esté con el foco.

- ***moveUp:*** establece el descriptor, a través de su índice, que recibirá el foco cuando la "*Flecha arriba*" del control remoto es presionado. Esta asignación sólo se realiza cuando el descriptor que lo define esté con el foco.

- ***moveDown:*** establece el descriptor, a través de su índice, que recibirá el foco cuando la "*Flecha abajo*" del control remoto es presionado. Esta asignación sólo se realiza cuando el elemento asociado a esté descriptor este con el foco.

- ***focusBorderColor:*** define el color del borde (rectángulo) de la región de este descriptor cuando el objeto a él asociado recibe el foco. El atributo puede tener uno de los siguientes valores: white, black, silver, gray, red, maroon, fuchsia, purple, lime, green, yellow, olive, blue, navy, aqua, ou teal (blanco, negro, plata, gris, rojo, fucsia, marrón, morado, verde lima, amarillo, aceite de oliva, azul, azul marino, azul turquesa, o verde azulado).

- ***focusBorderWidth:*** define el ancho del borde en píxeles del rectángulo cuando el elemento a él asociado recibe el foco. El espesor puede tomar valores positivos y negativos. En el caso de ser positivo se presentará fuera de la región, si no, el borde se presentará dentro de la región.

- ***focusBorderTransparency:*** define el porcentaje de transparencia que va a poseer un borde. Este recibe un valor real entre 0 y 1, donde 0 significa totalmente opaco y 1 indica totalmente transparente.

- ***focusSrc:*** define un archivo multimedia alternativo que debe ser presentado cuando el elemento asociado a este descriptor esté con el foco.

- ***focusSelSrc:*** define un archivo multimedia alternativo que debe ser presentado cuando es presionado el botón "Ok" o "Enter" mientras el elemento asociado a este descriptor esté con el foco.

- ***selBorderColor:*** define un color de borde que debe ser exhibido cuando sea presionado el botón "*Ok*" o "*Enter*" mientras el elemento asociado a este descriptor este con el foco.

En NCL se define aún el siguiente elemento opcional contenido en un elemento descriptor:

- ***descriptorParam:*** define un parámetro del descriptor como un par <*propiedad, valor*>. Las propiedades y sus respectivos valores dependen de programa de presentación del archivo multimedia asociado al descriptor.

Cada descriptor puede contener diversos elementos *descriptorParam*, definidos en el formato:

<*descriptorParam name="nombre_parametro"value="valor_parametro"/>*

Para indicar que archivo multimedia correspondiente debe ser reproducido con un volumen del 90 % del máximo:

```
<descriptor id="dVideo1" region="rgVideo1">
  <descriptorParam name="soundLevel" value="0.9" />
</descriptor>
```

El uso de parámetros de un descriptor promueve un alto grado de flexibilidad. Le corresponde a cada programa de presentación de objetos multimedia (*player*) interpretar esas propiedades de la forma adecuada. Actualmente, no se puede definir parámetros de un descriptor en el Composer. En NCL, están reservados los parámetros que se describen en la Tabla 4.1:

Transitions

Permiten mostrar objetos *MEDIA* con efectos de transición de entrada o salida. Una transición posee los siguientes atributos:

- ***id:*** identificador de la transición
- ***type:*** atributo obligatorio que indica el tipo de transición. Los valores posibles son : fade, barWipe, irisWipe, clockWipe, snakeWipe
- ***dur:*** duración de la transición en segundos. Por defecto es 1 segundo.

```
<transitionBase>
    <transition id="transicion" type="barWipe" dur="5s"/>
</transitionBase>
```

Las transiciones se asocian al descripor por:

- ***transIn:*** define la transición que será ejecutada al iniciar la presentación del objeto multimedia.
- ***transOut:*** define la transición que será ejecutada al terminar la presentación del objeto multimedia.

<descriptor id="descImagen" region="regImagen" transIn="transicion" explicitDur="10s" transOut="transicion"/>

Parámetros	Descripción
Objetos con Audio	
soundLevel, balanceLevel, trebleLevel, bassLevel	Valores entre 0 y 1. En el caso de soundLevel, 0 = mute; 0.5 = volumen a 50%; y 1 = volumen máximo
Objetos Visuales	
location	Posición del objeto multimedia. Se trata de dos números separados por una coma, en el orden <left, top> en uno de los siguientes formatos: a) números reales entre 0 y 100, seguidos del símbolo de porcentaje; o b) números enteros no negativos que especifiquen un valor en pixeles.
size	Dimensiones del objeto multimedia. Se trata de dos números separados por una coma, en el orden <width, height>, en uno de los siguientes formatos: a) números reales entre 0 y 100, seguidos del símbolo de porcentaje; o b) números enteros no negativos que especifiquen un valor en pixeles.
bounds	Posición y dimensiones del objeto multimedia. Se trata de cuatro números separados por una coma, en el orden <left, top, width, height>, en uno de los siguientes formatos: a) números reales entre 0 y 100, seguidos del símbolo de porcentaje; o b) números enteros no negativos que especifiquen un valor en pixeles.
background	Nombres de colores reservados: white, black, silver, gray, red, maroon, fuchsia, purple, lime, green, yellow, olive, blue, navy, aqua, o teal. También puede ser transparent, para el caso de imágenes con transparencia, como algunos GIFs
visible	true o false
transparency	número real entre 0 y 1 indicando transparencia: 0 significa totalmente opaco y 1 significa totalmente transparente
fit	toma uno de los siguientes valores: fill, hidden, meet, meetBest ou slice, donde: • **fill:** redimensiona el contenido del objeto multimedia para que toque todos los bordes de la región; • **hidden:** si la altura intrínseca del contenido del archivo multimedia es más pequeña que el atributo height, el objeto necesita ser renderizado a partir del tope y rellenar su altura restante con el color de background; si es mayor, el sobrante debe ser cortado. Idéntico para el ancho y la izquierda. • **meet:** redimensiona el contenido del objeto multimedia manteniendo sus proporciones hasta alcanzar uno de los bordes de la región. Si haya un espacio vacío a la derecha o en la parte de bajo, debe ser llenado con el color del background. • **meetBest:** semejante al meet, pero el objeto multimedia no es ampliado en más del doble de las dimensiones originales. • **slice:** redimensiona el contenido del objeto multimedia manteniendo sus proporciones hasta que toda la región sea llenada. Parte del contenido puede ser cortado a la derecha o en la parte inferior del contenido.
Objetos de media textual	
scroll	Toma uno de los siguientes valores: none, horizontal, vertical, both o automatic.
style	Localización de un archivo de hoja de estilo.

Cuadro 4.1: Parámetros que pueden ser utilizados en descriptor, de acuerdo al archivo multimedia..

4.1.2. Inserción de los elementos.

Para que un documento NCL tenga algo para mostrar, es necesario la declaración de contenidos que serán presentados, éstos son definidos por nodos media de NCL, como veremos a continuación. Los elementos descritos en esta sección pertenecen al cuerpo de NCL.

4.1.2.1. Medias.

Una media, es la representación del objeto que será mostrado por el documento, y definida por el elemento <*media*> de NCL, pudiendo ser un objeto multimedia como: video, audio, texto, html, lua, etc. Ésta también debe presentar además del archivo de origen, el descriptor que regula la presentación del nodo multimedia.

A continuación se muestra cómo se hizo el desarrollo de un elemento <media>.

<*Media Id="video" src="video.avi" descriptor="dpVideo"/*>

Una media posee los siguientes los atributos:

- ***id:*** identificador único del nodo multimedia, se emplea para hacer referencia al mismo.
- ***tipo:*** define el tipo de medios (audio, vídeo, texto, etc.) El valor de este atributo será uno de los tipos MIME definidos por IANA. La Tabla 4.2 muestra los tipos MIME utilizado por NCL.

Valor de **type**	Extensión de archivo de atributo src
Text/html	.htm, .html
Text/css	.css
Text/xml	.xml
Image/bmp	.bmp
Image/png	.png
Image/gif	.gif
Image/jpeg	.jpg
Audio/basic	.wav
Audio/mp3	.mp3
Audio/mp2	.mp2
Audio/mpeg4	.mp4, .mpg4
Video/mpeg	.mpeg, .mpg
Application/x-ginga-NCLua	.lua
Application/x-ginga-NCLet	.xlt, .xlet, .class
Application/x-ginga-settings	No tiene archivo asociado (no se defino el atributo src). Se trata de un nodo de atributos globales para ser utilizado en normas y switches.
Application/x-ginga-time	No tiene archivo asociado (no se define el atributo src)

Cuadro 4.2: Tipos de archivo multimedia

- ***descriptor:*** establece el descriptor que controla la presentación de objeto multimedia. El valor de este atributo deberá ser el identificador del descriptor.
- ***src:*** fuente del objeto multimedia. Se trata del camino para el archivo multimedia, que puede ser relativo, a partir del directorio donde se encuentra el archivo NCL, o absoluto, a través de una URI. Las URI válidas son las que se presentan en la Tabla 4.3.

Esquema	Formato	Uso
File:	///camino_archivo/#id_fragmento	Archivos locales
http:	//id_servidor/camino_archivo/# id_fragmento	Archivos remotos bajados del canal de retorno utilizando el protocolo http
Rstp:	//id_servidor/camino_archivo/# id_fragmento	Streams bajados del canal de retorno utilizando el protocolo rstp
Rtp:	//id_servidor/camino_archivo/# id_fragmento	Streams bajados del canal de retorno utilizando el protocolo rtp
Isdtv-ts	//id_programa	Streams recibidos del transport stream

Cuadro 4.3: URI válidas

- ***refer:*** referencia a otro nodo multimedia previamente definido, a manera de reutilización de nodo (utiliza los atributos del nodo multimedia referenciado, excepto el ***id***).
- ***instance:*** se utiliza solo cuando el atributo refer es definido, establece si un nodo que se refiere a otro genera una nueva instancia del objeto en el programa intérprete o se reutiliza la instancia previamente creada. Los valores posibles son:
  - **"new"**, cuando se trata de una copia;
  - **"instSame"**, cuando la media inmediatamente se hace referencia a si misma;
  - **"gradSame"**, cuando el medio es el mismo, desde que reciba explícitamente una acción de inicio (*start*).

4.1.2.2. Anclas de contenido.

Definen una parte de los contenidos de la ***media*** y son puntos de entrada para los nodos multimedia o de *contextos*. Por parte del contenido de una ***media***, se puede entenderse como un subconjunto de los elementos que conforman a ésta. Por ejemplo, digamos que queremos definir anclas de un vídeo. En este caso el vídeo en sí sería la media y las anclas serían los extractos del video. Las anclas pueden ser de tipo: de contenido (*content anchor*) o de propiedad (*property anchor*).

Un ancla de contenido define un segmento multimedia (intervalo de tiempo o región en el espacio) que pudiera ser utilizado como punto de activación de los enlaces. Un segmento de multimedia es una selección contigua de unidades de información (information units) de un nodo. La declaración de esas unidades de información depende del tipo de archivo multimedia presentado por el nodo.

Las anclas se utilizan para permitir la sincronización entre objetos multimedia (música, videos, textos, etc.). Imagínese el ejemplo de una película subtitulada. Esta película tiene varias escenas, cada escena tiene un diálogo. En este caso, la cinta debe tener un ancla para cada escena, permitiendo así que los subtítulos (diálogos) se presentan en el momento correcto.

Un ancla está definida por el elemento <*area*>, que es un elemento secundario de una media. El siguiente ejemplo demuestra la creación de un video con dos anclas.

```
<media id="video" src="video.avi" descriptor="dpVideo">
  <area id="ecena01" begin="2s" end="5s"/>
  <area id="ecena02" begin="5s" end="6s"/>
</ media>
```

En NCL se definen los siguientes atributos de ancla de contenido:

- ***id:*** identificador único de ancla. Este atributo es utilizado para referencia al elemento.
- ***begin:*** inicio del ancla. Es utilizado cuando se define un ancla de un objeto de media continuo. El valor de este atributo puede ser en el formato "segundos" o en "horas: minutos: segundos".
- ***end:*** atributo de final de ancla que es utilizado junto con el *begin*, de esta forma se define la terminación de ancla. Su valor tiene el mismo formato que el *begin*. En el caso que este atributo (***end***) no esté establecido, el final de ancla es considerado como el final de la media que la contiene
- ***label:*** define un identificador para el ancla. Una ***label*** es utilizada en objetos de procedimiento, pudiendo ser aplicado para identificar un conjunto de anclas.
- ***coords:*** coordenadas en pixeles del ancla espacial (atributo válido para archivos multimedia visuales), en el formato “X, Y, width, height”.
- ***dur:*** Duración del ancla, en segundos, en el formato “99.9s” (atributo válido para archivos multimedia continuos)
- ***first:*** cuadro/muestra del archivo multimedia definiendo el inicio del ancla (atributo válido para archivos multimedia continuos)
- ***last:*** cuadro/muestra del archivo multimedia definiendo el fin del ancla (atributo válido para archivos multimedia continuos)
- ***text:*** texto del ancla en el archivo de origen (atributo válido para archivos multimedia de texto)
- ***position:*** posición del texto del ancla en el archivo de origen (atributo válido para archivos multimedia de texto).
- ***anchorLabel:*** identificador del ancla en el archivo de origen, tal como es interpretado por la herramienta de exhibición.

Un ancla puede tener otros atributos para ser utilizado con otros tipos de medios, como medios no continuos.

4.1.2.3. Anclas de propiedades.

Hasta ahora las propiedades se definían en la región, descriptor, o parámetros del descriptor, una ancla de propiedades también se pueden declarar dentro del elemento ***media*** o ***context*** con la etiqueta ***property***, se utiliza una propiedad cuando es necesario la manipulación de algún atributo de un medio y es definida por el elemento <*property*> siendo este un elemento secundario de la media, que será manipulada por un enlace. El elemento <*property*> sólo contiene los atributos *name* y *value*. El atributo *name* indica el nombre de propiedad de media que será manipulado. El atributo *value*, a su vez, indica el valor de esta propiedad.

Una propiedad o conjunto será siempre definida con un valor, éste será el valor inicial referido al atributo y puede ser volumen del audio de un nodo de audio o video, coordenadas y dimensiones de exhibición de un nodo multimedia visual, entre otros. En el siguiente ejemplo se declaran cuatro anclas de propiedad para un nodo de video, además de un ancla de contenido:

```
<media id="video" src="video.avi" descriptor="dpVideo"/>
  <!—anclas de propiedad que seran manipuladas por los enlaces –>
  <property name=”top”/>
  <property name=”left”/>
  <property name=”width”/>
  <property name=”height”/>
  <!—anclas de contenido en el video que debe sincronizar la imagen – >
  <area id=”aVideo1Imagem1” begin=”3s” end=”8s”/>
</media>
```

Las propiedades que pueden ser declaradas para un nodo media se detallan en la Tabla 4.1.

4.1.3. Organización del documento.

Como se indicó en el capítulo 3 el modelo NCM define nodos contenidos y nodos de composición, que se utilizan para organizar y estructurar un documento según el modelo. En esta sección se muestra como estructurar un documento NCL usando nodos de composición.

4.1.3.1. Contexto.

El elemento ***body*** es un caso particular de ***contexto***, presentando el documento como un todo. En NCL, un nodo de composición o contexto está representado por el elemento <***context***> y se utiliza para estructurar un nodo hipermedia, los mismos que pueden ser anidados, con el objetivo de reflejar la estructura del documento y ayudarle al programador a organizar de mejor manera los segmentos del programa audiovisual interactivo. Éste puede tener otros elementos, incluyendo otros contextos, como elementos hijos.

Un contexto se mantiene activo mientras que uno de sus elementos hijos está activo y si una interfaz específica del contexto no se indica en un enlace, un contexto inicia todos los elementos secundarios cuando este inicia. El siguiente ejemplo muestra el desarrollo de un contexto.

```
<context id="context01">
```

```
<!– portas –>
<!– mídias e contextos –>
<!– elos –>
</ context>
```

Los atributos de un contexto son:

- ***id:*** identificador único de contexto
- ***refer:*** referencia a otro contexto previamente definido, utiliza los atributos del contexto referenciado, excepto el ***id*** que debe ser de la forma: alias#id_del_body_documento_importado.
  - El ***id*** del body importado debe ser difertente al del documento importador.

El contexto ***body*** de un documento NCL hereda el ***id*** del propio documento, y lo representa como un todo. Pero al utilizar el plugin para eclipse este no me ofrece un soporte para esta opción especificada por la norma de TVD.

4.1.3.2. Puertos.

Un puerto (***port***) define un punto de interfaz para un contexto. El modelo NCM no permite acceder a un elemento (nodos internos) dentro de un contexto directamente desde fuera del mismo, éste debe poseer una puerta que lo dirija hacia el nodo interno deseado. Todo documento debe poseer por lo menos una puerta de entrada indicando cual es el nodo multimedia que será presentado inicialmente (ver Figura 4.2). El puerto se encuentra en la sección *body*. En el caso que se desee iniciar más de un nodo media simultáneamente se deben crear más de un puerto y estos se ejecutaran al mismo tiempo.

```
<body>
  <port id="pVideoAbertura" component="videoAbertura"/>
  <port id="pImgPularIntro" component="imgPularIntro"/>
  ...
</body>
```

Un puerto contiene los siguientes atributos:

- ***id:*** Identificador único que permite que esta sea referencia por otro elemento del documento NCL.
- ***component:*** el componente que se asigna a través de la puerta, y que es el que será ejecutado por medio de la misma.
- ***interfaz:*** del componente que se está asignando. Puede ser ancla o propiedades, si el elemento asignado es una midia, u otros puertas, en el caso de que el elemento mapeado sea un *contexto*.

El cuerpo del documento NCL, elemento <*body*>, se considera un tipo especial de contexto.

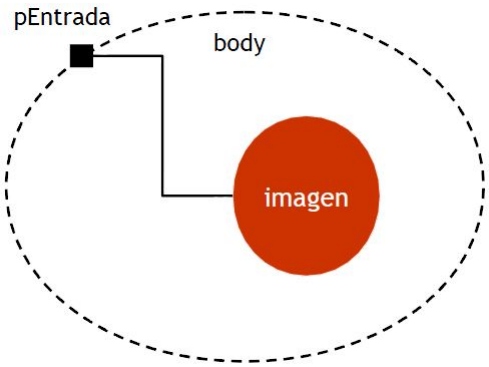


Figura 4.2: Puerta pEntrada como punto de entrada a un nodo interno de un contexto
Fuente: Referencia personal.

4.1.4. Sincronización de los elementos

Hasta ahora hemos visto como crear elementos, organizar el documento NCL y determinar qué elementos se mostrarán al inicio del documento. Además de lo mencionado se debe definir cómo los elementos se relacionan en la presentación y recepción de la interacción de los usuarios. Los conectores y enlaces son utilizados para este propósito.

4.1.4.1. Conectores

Los conectores establecen las relaciones genéricas que serán utilizadas por los elementos de un documento NCL, pero no específica los participantes de un relacionamiento individualmente. Por ejemplo, la relación "enseña a", define a alguien que enseña y alguien que aprende pero no indica quien enseña y quien aprende, es decir los participantes.

En NCM y en NCL, el sincronismo no está hecho por timestamps, pero si por mecanismos de causalidad y restricción definidos en los conectores (*connectors*). El conector establece las funciones (roles) que los nodos de origen y de destino ejercen en los enlaces que utiliza éste.

Todos los conectores se definen en la base de estos, en el elemnto <*connectorBase*> (que posee como único atributo un identificador ***id***), por la etiqueta <*causalConnector*>. Este elemento establece una relación de causa y efecto, como su nombre indica, a través de papeles de condición y de aplicación. Una función puede ser entendida como una interfaz de conector y que este indica cual partición tendrá un nodo como interfaz. Una base de conectores contiene los siguientes elementos hijos:

- <causalConnector>: define un conector propiamente.
- <importBase>: permite importar una base de conectores de algún otro archivo.

Una base de conectores puede ser definida conforme las siguientes estructuras:

```
<connectorBase id="menusConectores">
  <importBase .../>
  <importBase .../>
  <causalConnector id="onBeginStar">
```

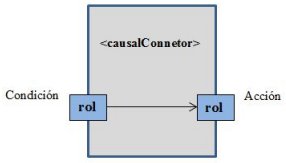


Figura 4.3: Ilustración de un conector causal (elemento <causalConnector>) con funciones (role) de condición y acción.
Fuente: Referencia personal.

```
...
</causalConnector>
<causalConnector id=" ">
... </causalConnector id=" ">
...
<causalConnector id=" ">
</connectorBase>
```

En el ejemplo citado anteriormente, la relacion "enseña a" posee dos papeles de quien enseña y de quien aprende. En el siguiente ejemplo se muestra el desarrollo de un conector.

```
<causalConnector id="onBeginStart">
  <simpleCondition role="onBegin"/>
  <simpleAction role="start" max="unbounded" qualifier="par"/>
</causalConnector>
```

En este ejemplo, se puede notar que la condición dada por el atributo "onBegin", indica la condición esperada al inicio de la presentación de un elemento, mientras que la condición dada por el papel de "start", indica que la presentación de un elemento será iniciada.

En NCL 3.0, existe solo un tipo de conector: o conector causal (*causal Connector*), éste define las condiciones (condition) bajo las cuales el enlace <*link*> puede ser activado y las acciones (action) que serán realizadas. Este conector debe poseer por lo menos una condición y una acción que están asociados a una funcion (*role*), punto de interfaz que participa de las asignaciones del enlace (Figura 4.3).

El elemento <*link*> hace referencia a un <*causalConnector*> y define un conjunto de asignaciones (elementos <*bind*> hijos del elemento <*link*>), que asocian cada extremo del enlace (interface de objeto) a un papel del conector utilizado, como ilustra la Figura 4.4:

Los elementos hijo de un <*causalConnector* > son:

- ***<connectorParam>***: define parámetros cuyos valores deberían ser establecidos por los enlaces que utilizan un conector.
- ***<simpleCondition> y <compoundCondition>***: definen las condiciones simples o compuestas de activación de un enlace que utiliza un conector;
- ***<simpleAction> y <compundAction>***: definen las acciones simples o compuestas que se realizaran cuando un enlace que utiliza un conector sea activado.

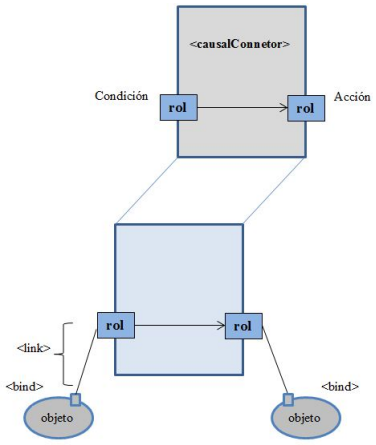


Figura 4.4: Ilustración de un enlace (elemento <link>)
Fuente: Referencia personal.

Tanto los papeles de condición "*onBegin*", "*onEnd*", "*onAbort*", "*onPause*" y "*onResume*", así como los papeles de acción "*start*", "*stop*", "*abort*", "*pause*" y "*resume*" están relacionados a posibles transiciones de estados de presentación de anclas de contenidos.

Un papel de condición "*onSelection*" está relacionado a eventos de selección y ligado a interactividad a través de dispositivos de entrada, como el control remoto de la TV (En el Apéndice C se muestran los conectores más utilizados para generar contenido interactivo)

Condiciones simples El elemento <*simpleCondition*> establece una condición que debe cumplirse para que el conector sea activado y además, define a través del atributo ***role*** el nombre del papel de la condición. NCL tiene un conjunto de nombres reservados para éstas funciones que identifican una condición, sin la necesidad de más información. Los nombres y sus significados están listados a continuación:

- ***onBegin:*** Se activa cuando la presentación de los elementos vinculados a esta función son iniciados.
- ***onEnd:*** Se activa cuando la presentación de los elementos vinculados a esta función son terminados.
- ***onAbort:*** Se activa cuando la presentación de los elementos vinculados aesta función son abortados.
- ***onPause:*** Se activa cuando la presentación de los elementos vinculados a esta función se detienen.
- ***onResume:*** Se activa cuando la presentación de los elementos vinculados a esta función retornan después de una pausa.
- ***onSelection:*** se activa cuando se pulsa una tecla que se especifique durante la presentación con el elemento vinculado a esta función o cuando se ejecuta la tecla *ENTER* y el elemento se encuentra con el foco.
- ***onBeginAttribution:*** Se activa inmediatamente antes de un valor a ser un atributo o una propiedad del elemento vinculado a este documento.

- ***onEndAttribution:*** Se activa inmediatamente después de un valor a ser un atributo o una propiedad del elemento vinculado a este documento.

Cuando la condición de "*onSelection*" es utilizada, es necesario establecer el atributo ***key*** del elemento <*simpleCondition*>. Este valor indica que tecla debe ser presionada para que el conector sea activado. Los valores posibles son: "0" al "9", "A" hasta la "Z", "*", "#", "MENU", "INFO", "GUIDE", "CURSOR_DOWN", "CURSOR_LEFT", "CURSOR_RIGHT", "CURSOR_ UP", "CHANNEL_DOWN", "CHANNEL_UP", "VOLUME_DOWN", "VOLUME_UP", "ENTER", "RED", "GREEN", "YELLOW", "BLUE", "BACK", "EXIT", "POWER", "REWIND", "STOP", "EJECT", "PLAY", "RECORD" y "PAUSE". Estos pueden ser establecidos con el uso de parámetros del conector como se verá más adelante. El elemento <*simpleCondition*> puede definir otros tipos de condiciones diferentes a estas. La correspondencia de los botones del control remoto con el teclado para la ejecución de las aplicaciones en el VSTB se muestra en la Tabla4.4.

Valores para la propiedad key:	Correspondencia con botones del control remoto:
RED	F1
GREEN	F2
YELLOW	F3
BLUE	F4
MENU	F5
INFO	F6
ENTER	
CURSOR_LEFT	
CURSOR_UP	
CURSOR_RIGHT	
CURSOR_DOWN	

Cuadro 4.4: Correspondencia de los botones del control remoto con VSTB

Condiciones compuestas Además de una condición simple, un conector puede definir una condición compuesta, las que están establecidas por la etiqueta <*compoundCondition*>. Este elemento posee dos o más condiciones simples como hijas. Cuando se utiliza, este elemento se debe declarar un atributo de la *ope-rator* que recibe los valores "*and*" y "*or*", indicando si todas o por lo menos una condición debe ser satisfecha para que el conector sea activado.

Acciones simple El elemento <*simpleAction*> define una acción a ser ejecutada cuando el conector es activado y establece a través del atributo ***role*** el nombre de la función de acción. Además del nombre del documento, este elemento define por medio del atributo ***max*** el número máximo de elementos para poder utilizar este documento. El valor "unbounded" especifica un número máximo ilimitado. Si el atributo *max* se declara, es necesaria la especificación de otro atributo llamado

qualifier, esto establece si la acción será ejecutada en paralelo o secuencialmente, valores de "*par*" y "*seq*" respectivamente. NCL también tiene un conjunto de nombres reservados para las funciones de acciones. Los nombres y su significado se muestran a continuación:

- ***start:*** inicia la presentación del elemento vinculado a esta función.
- ***stop:*** finaliza la presentación del elemento vinculado a esta función.
- ***abort:*** cancela la presentación de los elementos vinculados a esta función.
- ***pause:*** pausa la presentación de los elementos vinculados a esta función.
- ***resume:*** retoma la presentación de los elementos vinculados a esta función.
- ***set:*** establece un valor o una propiedad de un elemento asociado a esta función.

Cuando es utilizado el papel de "*set*", la condición que lo define también debe declarar el atributo ***value***, el que es el responsable de indicar el valor a ser recibido por la propiedad, que se puede establecer con el uso de parámetros de conectores.

Acciones compuestas Está definida por el elemento <*compoundAction*>. Una acción compuesta posee otras acciones simples como hijas. Cuando se utiliza éste, se debe definir un atributo *operator* que recibe los valores de "*par*" o "*seq*", indicando que las funciones deberán ser ejecutadas en paralelo o secuencialmente.

Parámetros Un conector también puede definir parámetros, éstos son descritos por el elemento <*connectorParam*> y se utiliza para que el valor sea validado o establecido y puede estar indicado en el momento de uso del conector. El siguiente ejemplo ilustra este concepto.

```
<causalConnector id="onKeySelectionStart">
  <connectorParam name="keyCode"/>
  <simpleCondition role="onSelection" key="$keyCode"/>
  <simpleAction role="start" max="unbounded" qualifier="par"/>
</causalConnector>
```

Se debe considerar que el parámetro solo define su nombre, dejando su valor para el momento de su uso. Los lugares donde el valor de los parámetros son utilizados indican esa funcionalidad a través del valor *"$nome_do_par^ametro.*

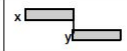
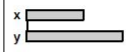
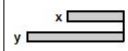
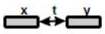
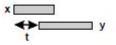
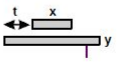
En la Tabla 4.5 se muestran algunos conectores y su funcionamiento representados por bloques.

Importando bases de arquivos externos

Para importar bases de datos de archivos externos se utiliza *importBase* como un elemento anidado al atributo que corresponde a la base a ser importada. Para importar una base de regiones, por ejemplo, se debe definir un *importBase* dentro del elemento *regionBase*:

```
<regionBase>
  <importBase alias="regioesDocumentario"
  documentURI="baseRegioesDocumentario.ncl" />
</regionBase>
```

El elemento ***importBase*** posee los seguientes atributos:

Relacion de iniciación	Ilustración	Conector Hipermídia
x inicia y ***y*** inicia cuando termina x		onEndStart
x y ***y*** inician simultaneamente		onBeginStart
x y ***y*** terminan simultaneamente		onEndStop
x termina y ***y*** inicia despues de un tiempo t		onEndStartDelay
x inicia y ***y*** inicia despues de un tiempo t		onBeginStartDelay
x es contenida durante la presentacion de ***y*** y su inicio es defasado por un tiempo t		onBeginStartDelay onEndStopDelay
x y ***y*** inician y terminan igualmente		onBeginStart onEndStop

Cuadro 4.5: Ejemplos de conectores

- ***alias:*** "apellido" del archivo importado. Este es el nombre que se utiliza como prefijo para referirse a los elementos importados, en el formato ***apellido#id_do_elemento_importado***. Para referirse al conector ***onEndStop*** dentro del archivo importado, se debe utilizar la cadena ***apellido#onEndStop***.
- ***documentURI:*** La ubicación y el nombre del archivo que contiene la base a ser importada.
- ***región:*** en el caso del archivo importado contenga la base de las regiones, se define qué región del programa contendrá las regiones importados.

Vale señalar que cuando una base de descriptores es importado, las regiones se importan automáticamente. En otras palabras, cuando hay un *importBase* en la sección *descriptorBase*, no es necesario hacer *importBase* de las regiones correspondientes en la sección *regionBase*, pudiendo estar vacía.

4.1.4.2. Enlaces

Un enlace (link) se utiliza para identificar los elementos que participan en una relación. Después del ejemplo de analogía "enseña a" presentado en la sección anterior, un enlace que utilice esta relación debería identificar los elementos "maestro" y "estudiante". La correspondencia completa especificada por el enlace, sería entonces "el maestro enseña al alumno". NCL establece los siguientes atributos:

- ***id:*** identificador único de enlace
- ***xconnector:*** identificador de conector asociado al enlace

NCL define los siguientes elementos contenidos en un elemento de tipo enlace:

- ***LinkParam:*** establece un parámetro del enlace como un par <*propiedad, valor*>. Las propiedades y sus respectivos valores dependen de la definición del conector al cual el enlace está asociado. Un enlace puede contener diversos elementos *linkParam*.

- ***Bind:*** indica un componente (nodo multimedia o de contexto) involucrado en el enlace, enseñando su papel (*role*) en el mismo, conforme la semántica del conector. En algunos casos se debe declarar también el punto de interfaz (*interface*) del nodo, al cual el enlace está ligado (puerta del contexto o ancla de un nodo multimedia). Un enlace puede contener diversos elementos *bind*, y debe contener por lo menos un *bind* para cada papel definido en el conector.

Un enlace está definido por el elemento <*link*>. Éste establece la relación que se utiliza a través de su atributo *xconnector*. Para la creación de las conexiones entre los elementos y los documentos, un enlace crea elementos secundarios <*bind*> como ya se indicó, para los atributos:

- ***role:*** identifica el papel que se utiliza
- ***component:*** identifica, a través de su ***id***, el elemento participando de la relacion.
- ***interface:*** identifica una ancla o la propiedad de un elemento en el caso que éste sea una media o un puerto de un elemento en el caso que éste sea una composición, a través de su ***id***.

El elemento bind puede contener una o más instancias del siguiente elemento como elementos hijos:

- ***bindParam:*** declara un parámetro específico del *bind* como un par <*propiedad, valor*>, el primero identifica el parámetro y el segundo su valor. Las propiedades y sus respectivos valores dependen de la definición del conector al cual el enlace está asociado.

El siguiente ejemplo demuestra a creación de un enlace.

```
<link id="link1" xconnector="onKeySelectionStart">
  <bind component="video" role="onSelection">
    <bindParam name="keyCode" value="YELLOW"/>
  </bind>
  <bind component="bkg" role="start"/>
  <bind component="screen" role="start"/>
</link>
```

En el ejemplo se puede notar la existencia del elemento <*bindParam*>, el mismo es utilizado en casos que se requiere el paso de parámetros.

4.1.5. Definiendo alternativas.

Se indicó cómo los elementos serán presentados, que elementos serán presentados y el orden en que serán presentados, creando un orden de presentación que se llama flujo temporal de programa NCL.

Además de las facilidades presentadas, NCL también permite que un programa pueda tener flujo temporal modificado durante su presentación. Esa modificación se efectúa a través da la definición de alternativas de contenido que especifican para un determinado momento de la presentación del programa NCL, caminos posibles a ser seguidos. La opción de un camino está dada por la evaluación de condiciones, llamadas ***reglas***.

4.1.5.1. Reglas

Representan condiciones que pueden ser usadas en un documento NCL, es decir básicamente, compara una propiedad de un elemento NCL con un valor retornando un resultado de esa comparación. Una regla está definida por el elemento *<rule>*.

Los siguientes atributos son definidas por las reglas:

- ***id:*** identificador de regla, es referenciado cuando el uso de la misma será necesario;
- ***var:*** indica que propiedad estará siendo testeada. El valor de ese atributo será un identificador de una propiedad;
- ***comparator:*** indica que tipo de comparación será hecha por la regla;
- ***value:*** indica el valor que será comparado con la propiedad.

La Tabla 4.6 demuestra los posıbles comparadores utilizados en una regla. El siguiente ejemplo demuestra la creación de una regla.

```
<ruleBase>
  <rule id="rEn" var="idioma" comparator="eq" value="en" />
  <rule id="rPt" var="idioma" comparator="eq" value="pt" />
</ruleBase>
```

Una regla es definida en base de reglas, representada por el elemento *<ruleBase>*, en la cabecera del documento NCL.

eq	igual a
ne	diferente de
gt	mayor que
ge	mayor o igual a
lt	menor que
le	menor o igual a

Cuadro 4.6: Comparadores utilizados en reglas NCL.

Una forma de almacenar las propiedades que serán utilizadas en las reglas es emplear el nodo settings. Se trata de un nodo de propiedades globales, como se especifica en el ejemplo siguiente:

```
<media type="application/x-ginga-settings" id="nodeSettings">
  <property name= "idioma" />
</media>
```

4.1.5.2. Switch

Es un elemento que presenta alternativas de contenido que un programa NCL puede tomar en un determinado momento. Evalúa una serie de reglas que si son verdaderas, activa uno de los elementos definidos dentro él.

Un switch es activado de la misma manera que un elemento cualquiera de NCL, su identificador puede ser utilizado en los enlaces tornándose posible que este sea activado, parando su misma participación de alguna condición de disparo.

El siguiente ejemplo detalla la creación de un switch.

```
<switch id="escolhe_audio">
  <bindRule rule="rule_en" constituent="audio_en"/>
  <bindRule rule="rule_pt" constituent="audio_pt"/>

  <media id="audio_en" ... />
  <media id="audio_pt" ... />
</switch>
```

Para que un switch pueda elegir un elemento y presentarlo, es necesaria la definición de asignaciones, éstas se definen por el elemento <*bindRule*> como se especifica en el ejemplo anterior. La regla que especifica la asignación debe ser evaluada y, en el caso que ésta sea verdadera, los elementos deben ser activados.

Se debe explicar que las reglas se evalúan de forma secuencial, siendo la primera regla evaluada como verdadera, la que definirá el elemento que será presentado, o hasta terminar el conjunto de asignaciones de aquel switch. Por lo tanto, incluso si más de una regla es cierta en un momento dado, sólo uno de los elementos se mostrará.

Si un switch tiene elementos de tipo contexto, también es necesario especificar el puerto destino de cada asignación de las reglas. Esta indicación es hecha por el elemento <*switchPort*> o por más de uno como se muestra en el siguiente ejemplo.

```
<switch id="escolhe_audio">
  <switchPort id="mapeamento">
    <mapping component="context_en" interface="port_en" />
    <mapping component="context_pt" interface="port_pt" />
  </switchPort>

  <bindRule rule="rule_en" constituent="context_en" />
  <bindRule rule="rule_pt" constituent="context_pt" />

  <context id="context_en">
    <port id="port_en" ... />
    ...
  </context>

  <context id="context_pt">
    <port id="port_pt" ... />
    ...
  </context>
</switch>
```

El ***<descriptorSwitch>*** define una lista de descriptores a utilizar según reglas que se definen. Un descriptor switch posee un identificador único ***id*** del switch descriptor. Se utiliza en la propiedad “descriptor” de la media.

```
<head>
  <descriptorBase>
    <descriptorSwitch id=“descTamañoPantalla”>
      <bindRule constituent="descriptorId" rule="rResolucion"/>
      <descriptor id=”descriptorId” region=”regionId”>
      ...
    </descriptorSwitch>
  </descriptorBase>
```

```
</head>
```

Cuando el ningún descriptor cumpla ninguna de las reglas definidas se selecciona un descriptor default ***<defaultDescriptor>***.

```
<descriptorSwitch id="descSubtitulos" >
  <bindRule constituent="descSubtituloEN" rule="rIngles"/>
  <bindRule constituent="descSubtituloES" rule="rEspañol"/>
  <defaultDescriptor descriptor ="descSubtituloES" />
  ...
</descriptorSwitch>
```

4.2. Ambiente de desembolvimiento de apliciones interactivas.

Un ambiente de desenvolvimiento hace referencia a un conjunto de herramientas para la edición, codificación y visualización de aplicaciones interactivas. Para la edición y codificación se utilizara un IDE muy utilizado llamado Eclipse.

El uso del IDE Eclipse es el más favorable ya que ofrece una cantidad de funcionalidades, como una serie de plugins para facilitar el desarrollo de las aplicaciones interactivas.

Para simular las aplicaciones de una TV con interactividad se utilizara VMware con una imagen de la máquina virtual con el middleware Ginga-NCL instalado. Esta imagen se llama Ginga-NCL VSTB y es la que se encarga de simular el papel de SetTopBox receptor. La instalación completa se encuentra detallada en el Apéndice A, esta debe ser realizada antes de comenzar a desarrollar las aplicaciones, ya que sin uno de los programas mencionados se vuelve imposible la generación de contenido interactivo para TVD.

4.3. Pasos para crear un documento NCL

Todos los programas NCL parten de un esqueleto básico:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<ncl id=“main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
      region, descriptor, connector
  </head>
  <body>
      media, port, switch, link
  </body>
</ncl>
```

Este esqueleto se genera de forma automática mediante el plugin Ginga-NCL que está instalado en el software Eclipse, cuando se crea un Documento NCL.

Por lo general, los pasos para construir un documento NCL son:

1. Añadir el encabezado básico;
2. Definir las regiones de la pantalla que aparecerán los elementos visuales (**regionBase**);

3. Definir cómo y donde los nodos de media serán mostrado, atravez de descriptores (**descriptorBase**);
4. Definir el contenido (nodos de **media**) y de la estructura (**contextos**) del documento (sección **body**), asociados con los descriptores;
5. Definir las puertas (**port**) de los contextos, destinadas a la construcción de los enlaces entre contextos y nodos de media e incluyendo la puerta de entrada al programa en el **body**, apuntando al primer nodo que se va a mostrar;
6. Definir anclas para los nodos media, con el objetivo de la construcción de los enlaces entre nodos de media (**area** y **property**);
7. Definir enlaces (**links**) con el sincronismo e interactividad entre los nodos de media y contextos.
8. Definir los conectores que especifican el comportamiento de los enlaces del documento (**connectorBase**);

4.4. Creación del primer proyecto Ginga-NCL en Eclipse.

La implementación del entorno de desarrollo (IDE) Eclipse permite crear y editar aplicaciones NCL. Una de las funcionalidades más importantes en el desenvolvimiento de programas con NCL Eclipse, es el identificador y marcador de errores en el momento de redacción de código del programa, permitiendo que el autor encuentre el error de forma rápida, eliminando la necesidad de ejecutar el documento para evidenciar los posibles erros de sintaxis o semánticos.

Para crear el primer proyecto se ejecuta la plataforma Eclipse, se selecciona en la barra de herramientas la opción *File* y se procede a escoger *New -> Project* como se muestra en la imagen.

En la siguiente ventana escogemos la opción de creación de un proyecto Java general *(General- >Project)*, y damos clic en el botón de siguiente.

Se nos abrirá otra ventana en la que asignamos un nombre (en este caso el nombre es Ejemplo1) al nuevo proyecto que se va a crear y la ubicación del mismo, y seleccionamos la opción de *Finalizar*.

Una vez creado el nuevo proyecto general se debe proceder a crear el documento para la edición del código NCL, el plugin Ginga-NCL que está instalado en Eclipse genera automáticamente la estructura básica del documento. Para la creación de la aplicación se da clic derecho sobre la carpeta que contiene el proyecto general Java, y escogemos la opción de ***Other***, debido a que la opción para crear el documento NCL no se presenta de forma directa (ver Figura 4.7).

En la imagen que se muestra se selecciona la opción *NCL Document* y se procede a continuar con la generación del mismo. En la siguiente ventana de creación que se despliega al escoger la opción de un documento NCL se asigna el nombre para el documento NCL en la reja de ***Id***, este nombre es el identificador del mismo y se escribe automáticamente en la cabecera de la estructura básica del documento NCL, además debe ser el mismo nombre *"File Name"* del documento con extension .ncl (ver Figura 4.8).

En la Figura 4.9 se presenta una visión general de un proyecto Ginga-NCL Eclipse con un documento NCL recién creado, aquí se puede observar como el plugin crea automáticamente el esqueleto del cual parten todos los programas.

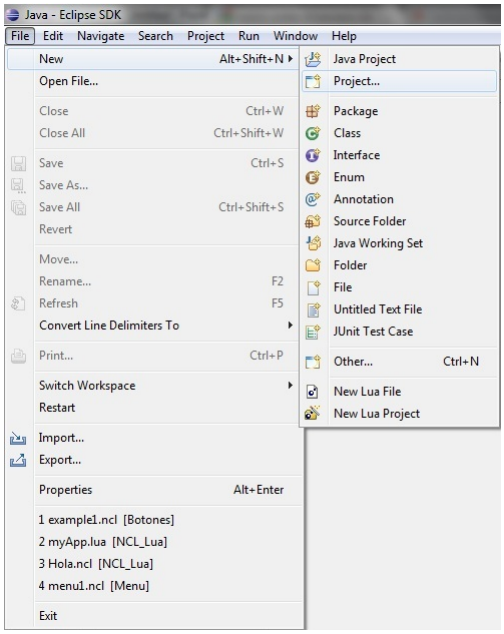


Figura 4.5: Creación del primer proyecto Ginga-NCL
Fuente: Captura de la pantalla del software Eclipse-NCL

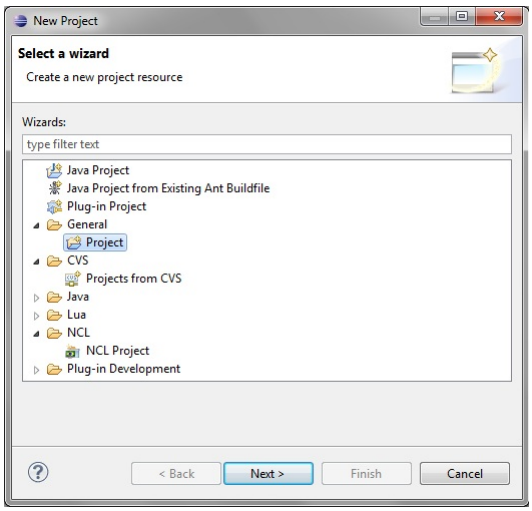


Figura 4.6: Proyecto Java General
Fuente: Captura de la pantalla del software Eclipse-NCL

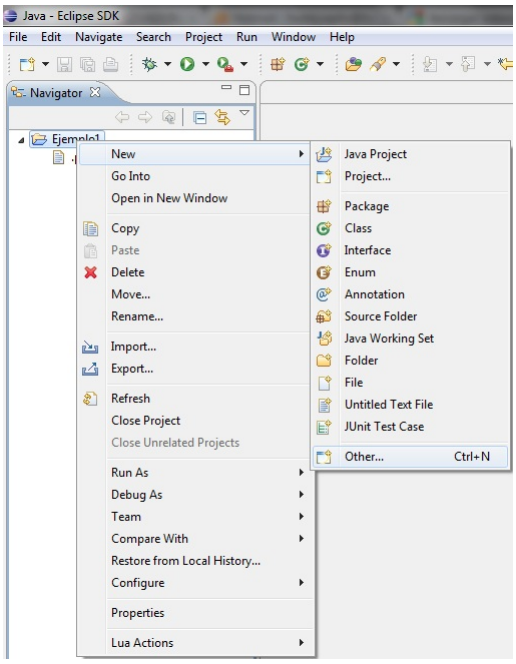


Figura 4.7: Primer ejemplo de creación de un proyecto Ginga-NCL
Fuente: Captura de la pantalla del software Eclipse-NCL

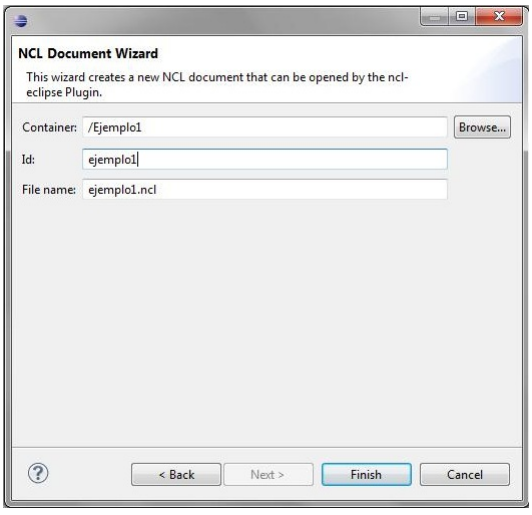


Figura 4.8: Ventana de asignacion del nombre del documento NCL
Fuente: Captura de la pantalla del software Eclipse-NCL

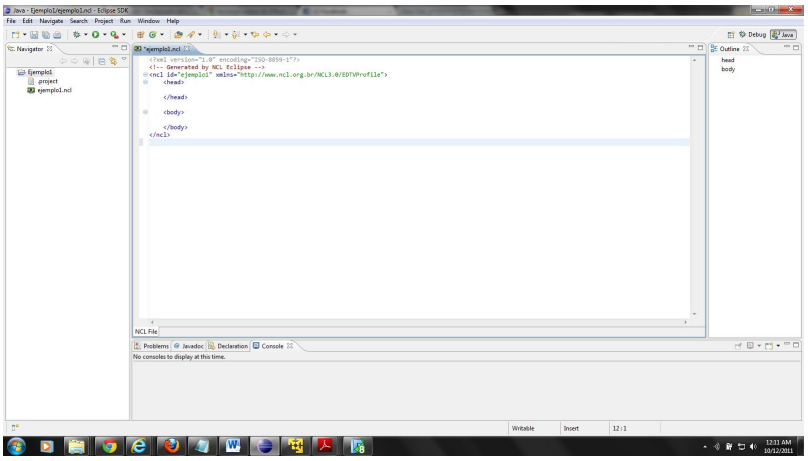


Figura 4.9: Vista general de un proyecto NCL Eclipse
Fuente: Captura de la pantalla del software Eclipse-NCL

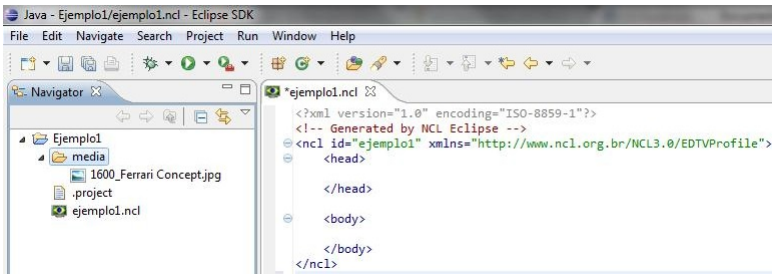


Figura 4.10: Nodo contenido dentro de la carpeta media.
Fuente: Captura de la pantalla del software Eclipse-NCL

Para que el proyecto Java NCL pueda presentar los diferentes tipos de objetos media, estos deben estar contenidos dentro de una carpeta o carpetas que pertenezcan al proyecto Java, para un mejor orden de estos y un manejo más sencillo, además estas carpetas son las direcciones "*src*" que contienen los nodos y que se debe especificar al definir la *media* para que estos puedan ser presentados, el proceso de declaración se detallara en la siguiente sección.

La creación de una carpeta dentro de un proyecto se realiza dando clic derecho sobre la carpeta general del proyecto NCL (*Ejercicio 1*) de la misma forma que se procede para crear un *Documento NCL*, es decir se selecciona *New->File->Folder,* aparecerá una ventana que permite nombrar la carpeta además de poder leer el proyecto al que pertenece la misma.

Para que puedan ser accedidos por el código del programa los diferentes objetos media que presentara el programa NCL deben estar contenidos dentro del proyecto, esto se lo realiza de forma sencilla, solamente se deben arrastrar de su ubicación de origen hacia la carpeta que lo contendrá para su muestra en el proyecto En la Figura 4.10 se muestra el objeto media contenido dentro de la carpeta.

4.5. Ejemplos de programacion en Ginga-NCL.

4.5.1. Ejemplos sin Interacción con el usuario:

4.5.1.1. Reproducir un video en la pantalla.

En esta parte presentamos la reproducción de un video en la pantalla completa del VSTB, a pesar de no tratarse de un ejemplo de un documento hipermedia, se utiliza para familiarizarse con los conceptos básicos de NCL. El orden que se sigue para el desarrollo del ejemplo es el que se planteó en la sección 4.4, definiendo primero las regiones, luego los descriptores, el nodo media con su contenido y por último la puerta que indica el inicio del programa. A continuación se detalla paso a paso la creación del ejemplo. (Para presentar contenido multimedia ya sea imágenes o texto se deben considerar las recomendaciones que se indican en el Apéndice B).

Paso 1: Definición de las regiones en la pantalla.

Las regiones se definen por el elemento $<region>$ en el encabezado del documento (entre las etiquetas $<head>$ y $</head>$), en la sección de la base de la región $<regionBase>$ (entre las etiquetas $<regionBase>$ y $</regionBase>$), en este ejemplo se utiliza solamente una región ("regVideo"), para presentar el video *video1.mpg* al 100 % de la pantalla, es decir el tamaño completo de la misma.

Paso 2: Definimos los descriptores que determinan como un video sera exhibido.

Después de especificar las regiones, se debe definir el descriptor ("*desVideo*") que determine como el video será presentado en la pantalla. En este ejemplo el descriptor será utilizado solamente para asociar una media a una región. Como ya se ha visto el descriptor se declara en el encabezado del documento (entre las etiquetas $<head>$ y $</head>$), en la sección de la base de los descriptores $<descriptorBase>$ (entre las etiquetas $<descriptorBase>$ e $</descriptorBase>$).

Paso 3: Definición de los nodos media.

En este parte se crea el nodo media (*video*) que compone el documento, esta definición del nodo se la realiza en el *body* del docuemnto NCL, como ya se indicó en la sección 4.4, ésta debe poseer un identificador ***id***, el tipo (*type*) de archivo media que se quiere presentar (puede ser opcional) y se debe indicar que descriptor será utilizado para presentar el video. Una observación, en esta definición es que el atributo ***src*** debe direccionar la ubicación exacta del archivo multimedia que creara el nodo media, este folder que lo contenga debe estar creado dentro del proyecto Java o en su defecto el archivo debe estar copiado de forma directa en el mismo.

Paso 4: Dfenición de la puerta del contexto body que determina el inicio del pregrama.

Por último en la creación del ejemplo, es necesario definir por lo menos una puerta ("*InVideo*") en el contexto $<body>$ que es la que da acceso al nodo media para que pueda ser presentado cuando el programa inicia.

Codigo del ejemplo 1.

<head>

```
    <regionBase>
        <region id="regVideo" width="100 %" height="100 %" />
    </regionBase>

    <descriptorBase>
        <descriptor id="desVideo" region="regVideo"/>
    </descriptorBase>
</head>

<body>
    <port id="InVideo" component="video"/>

    <media id="video" type="video/mpeg" src="media/video1.mpg"
     descriptor="desVideo">
    </media>
</body>
```

4.5.1.2. Reproduciendo un vídeo y una imagen con transparencia.

Partimos del ejemplo anterior para construir un programa que inicie una imagen con trasparencia del 60 % y que se presente durante 30s cuando comienza el mismo, tanto el video como la imagen se presentan de forma simultánea, ya que el programa posee dos puertas, una para iniciar a cada nodo multimedia, para este ejemplo adicionamos otros pasos con respecto al programa anterior y que se detallan a continuación.

Paso 1: Definición de la region en la pantalla para mostrar la imagen.
Definimos la región en la pantalla (*regImagen*) donde será mostrada la media de la imagen (*Logo*), de la misma forma que en el ejemplo anterior, para garantizar que la imagen sea presentada sobre el video se utiliza el atributo ***zIndex***.

Paso 2: Definimos el descriptor que determinan como la imagen seran exhibida.
Definir el descriptor desImagen, que determinara la región (*regImagen*) en que será presentada y la forma, es decir se indica por medio del atributo *explicitDur* la duración de la presentación de la media (*Logo*). Esta declaración se la realiza de forma similar que el descriptor en el ejemplo anterior, pero con la adición del atributo.

Paso 3: Definición el nodo media correspondiente a la imagen.
Para la creación de nodo media (imagen), se lo hace de similar al del ejemplo anterior, pero teniendo en cuenta que esta vez el nodo se refiere a una imagen (*Logo*), y que el descriptor debe ser indicado para saber cuál será utilizado para iniciar la presentación. Además aquí se especifica la transparencia de la imagen a través del atributo ***transparency*** que se define como una propiedad (*property*) del nodo.

Paso 4: Definición de la puerta adicional para la nueva imagen.
Por último se debe definir una nueva puerta ("*InImagen*"), ya que los dos nodos media (video e imagen) serán presentados simultáneamente. Esta declaración se realiza de la misma forma que en el contexto ***body*** del ejemplo anterior.

Codigo del ejemplo 2.

```
<head>
  <regionBase>
    <region id="regVideo" width="100 %" height="100 %"
    zIndex="1" />
    <region id="regImagen" left="86 %" top="5 %" width="10 %"
    height="10 %" zIndex="2" />
  </regionBase>

  <descriptorBase>
    <descriptor id="desVideo" region="regVideo"/>
    <descriptor id="desImagen" region="regImagen"
    explicitDur="30s"/>
  </descriptorBase>
</head>

<body>
  <port id="InVideo" component="video"/>
  <port id="InImagen" component="imagen"/>

  <media id="video" type="video/mpeg" src="media/video1.mpg"
   descriptor="desVideo"/>
  <media id="imagen" type="image/jpeg" src="media/Logo.jpg"
   descriptor="desImagen" >
      <property name="transparency" value="50 %"/>
  </media>
</body>
```

4.5.1.3. Iniciando y terminando dos objetos de medias simultaneamente.

En este ejemplo se presentan dos nodos media de manera simultánea, un video y una imagen. En NCL el inicio y finalización entre medias está dado por lo enlaces (links) entre las mismas. El comportamiento de los enlaces es establecido por conectores y enlaces asociados. El conector define uno o más papeles para la condición de activación del enlace y una o más acciones que se deben realizar cuando el mismo es activado, en este ejemplo se utilizan los conectores ***onBeginStart*** y ***onEndStop*** y partimos del ejercicio anterior y le realizamos algunas modificaciones.

El papel del conector ***onBeginStart*** es muy simple como se indicó en la subsubseccion 4.2.4.1, ya que solo posee un papel de condición (*onBegin*) y uno de activación (*start*). El conector ***onBeginStart*** comienza la presentación del nodo ligado al papel start causando que el nodo ligado a la función ***onBegin*** sea exhibido. En este ejemplo es necesario crear dos enlaces:

- Un enlace es utilizado por el conector ***onBeginStart*** para cuando el video inicie dé arranque a la imagen (*Logo*).
- Y un enlace que es utilizado por el conector ***onEndStop*** para cuando el video concluya también finalice la presentación de la imagen (*Logo*).

Un identificador de un enlace es necesario solamente cuando este va a ser manipulado por comandos de edición en vivo, pero si se los define son utilizados como un recurso del documento para facilitar la lectura del mismo.

En este ejemplo al iniciar el documento, es presentado el nodo media *video*, lo que inicia la ejecución del nodo media *imagen* (a través del enlace que utiliza el conector ***onBeginStart***). Cuando termina el *video*, finaliza también la presentación de *imagen* (a través del enlace que utiliza el conector ***onEndStop***).

Paso 1: Removiendo el atributo *explicitDur* del descritor de la imagem.

En la seccion <descriptorBase>, es necesario alterar la definicion del descriptor **desImagen**, eliminado del mismo el atributo **explicitDur="10s"**.

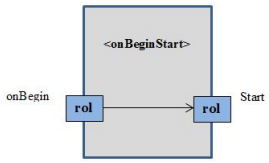
Paso 2: Removiendo la puerta *"InImagen"* que era la que presentaba la imagen.

Se debe remover la puerta **"*InImagen*"** ya que la presentacion de la imagen se realizara por conectores.

Paso 3: Definición de los conectores.

Para que los enlaces inicien o terminen la presentación de un objeto media cuando otro inicie o termina, es necesario crear conectores que describan ese comportamiento. La definición de la base de conectores también se la realiza en el encabezado del documento NCL. Aunque es posible importar conectores que son ofrecidos por la herramienta de autoría NCL y que pueden ser reutilizados en diferentes documentos, en este ejemplo se utilizara otra alternativa a la importación del conector, que es la de definirlos en el propio documento NCL.

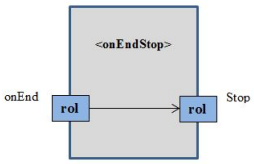
En la siguiente Tabla 4.7se define el funcionamiento del conector ***onBeginStart***.

Nombre:	**onBeginStart**
Condición:	Inicia la exhibición del nodo (ligado a la función **onBegin**)
Acción:	Inicia la exhibición del nodo (ligado a la función **start**)
Ilustración:	. 
Código NCL:	<causalConnector id="onBeginStart"> <simpleCondition role="onBegin"/> <simpleAction role="start"/> </causalConnector>
Lectura:	Cuando el nodo ligado a la función ***onBegin*** inicia, empieza el nodo ligado a la función ***start***. (Es el enlace a la vinculación con el conector, que nos permitirá sustituir los términos expuestos por los mismos.)

Cuadro 4.7: Definición del conector ***onBeginStart***

En la Tabla 4.8se describe el conector ***onEndStop***, tambien utilizado en el ejemplo.

Paso 4: Dfenición los enlaces de sincronización, para iniciar y terminar la exhibición de las medias.

Nombre:	**onBeginStart**
Condición:	Ternina la exhibición del nodo (ligado a la función **onEnd**)
Acción:	Cierra la exhibición del nodo (ligado a la función **stop**)
Ilustración:	. 
Código NCL:	<causalConnector id="onEndStop"> <simpleCondition role="onEnd"/> <simpleAction role="stop"/> </causalConnector>
Lectura:	Cuando el nodo ligado a la función ***onEnd*** inicia, termina el nodo ligado a la función***start***.

Cuadro 4.8: Definición del conector ***onBeginStart***

El primer enlace a ser creado, el que debe iniciar la presentación de la imagen cuando empieza a ser exhibido el nodo media video. En otras palabras, cuando video inicia, comienza imagen. Al examinar la definición de conectores en el paso anterior, se observa que esto se realiza a través del conector onBeginStar. El enlace que se declare debe estar ligado a la media video por el papel onBegin y a la media imagen por el papel start, como se muestra en la Figura 4.11. La definición de los enlaces se realiza en el cuerpo del documento, generalmente debajo de la declaración de las medias.

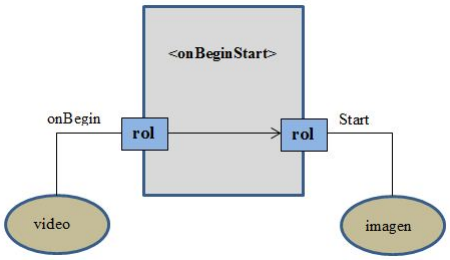


Figura 4.11: Ligación de medias con los enlaces que utiliza el conector ***onBeginStart***.
Fuente: Referencia personal.

Ya que el conector utilizado fue definido en el documento en sí, el atributo ***xconnector*** hace referencia sólo a la ***id*** del conector (***xconnector= "onBeginStart"***). Si el conector hubiera sido importado de un externo, el atributo ***xconnector*** tendría la forma *"alias_da_base_importada#id_do_conector"*.

Codigo del ejemplo 3.

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

```
        </causalConnector>
        <causalConnector id="onEndStop">
            <simpleCondition role="onEnd" />
            <simpleAction role="stop" />
        </causalConnector>
    </connectorBase>
</head>
<body>
    .....
    <link xconnector="onBeginStart">
        <bind role="onBegin" component="video" />
        <bind role="start" component="imagen" />
    </link>
    <link xconnector="onEndStop">
        <bind role="onEnd" component="video" />
        <bind role="stop" component="imagen" />
    </link>
</body>
```

4.5.1.4. Iniciando un objeto de media sincronizado a otro con retardo.

El objetivo de este ejemplo es modificar el anterior para que exhiba la media *imagen* "5s" después de que inicie la presentación del nodo *video*. Para esto se define el nuevo conector ***onBeginStartDelay*** que acepta un parámetro de duración de retardo.

Paso 1: Dfenición de los enlaces de sincronización, para iniciar y terminar la exhibición de las medias.

Para iniciar un vídeo con retardo, podemos hacer uso del atributo ***delay*** del elemento ***simpleAction*** de un conector. Para especificar un retardo de un tiempo de duración determinada, utilizada por todos los enlaces que utilice el conector, se puede definir el término directamente en el conector en el formato " *5s*", como se muestra a continuación.

```
<connectorBase>
  ...
  <causalConnector id="onBeginStartDelayFixo">
      <simpleCondition role="onBegin" />
      <simpleAction role="start" delay="2s" />
  </causalConnector>
</connectorBase>
```

Sin embargo, con el fin de hacer más flexible el conector, elegimos este ejemplo para definir un parámetro del conector para este retraso (aquí llamado ***oRetardo***), cuyo valor será dado por cada enlace que utilice el conector.

Paso 2: Modificando el elnlace para incluir un retardo.

Tomando como punto de partida el enlace definido en el ejemplo anterior. Modificamos el nombre del conector ***onBeginStart*** por ***onBeginStartDelay***, ademas es necesario definir el parámetro de retardo ***oRetardo*** que será utilizado

por el mismo para establecer la duración del retardo, por esta razón, la conexión de la función ***start*** debe modificarse, adicionando un elemento ***bindParam*** (parámetros de conexión) que especifica el valor de *5 segundos* para que el parámetro ***oRetardo***. Cabe señalar que ***bind*** es un *elemento atomico* , sera abierto (<***bind***) y cerrada por (/>) en la misma linea. Este ejemplo ademas contiene otro elemento (***bindParam***) y sólo "se cierra" después de (</***bind***):

```
<bind component="imgPularIntro" role="start">
  <bindParam name="oRetardo" value="5s" />
</bind>
```

Codigo del ejemplo 4.

```
<head>
    ....................
    <connectorBase>
        <causalConnector id="onBeginStartDelay">
            <connectorParam name="oRetardo"/>
            <simpleCondition role="onBegin" />
            <simpleAction role="start" delay="$oRetardo"/>
        </causalConnector>

        <causalConnector id="onEndStop">
            <simpleCondition role="onEnd" />
            <simpleAction role="stop" />
        </causalConnector>
    </connectorBase>
</head>

<body>
    ....................
    <link xconnector="onBeginStartDelay">
        <bind role="onBegin" component="video" />
        <bind role="start" component="imagen" >
            <bindParam name="oRetardo" value="5s"/>
        </bind>
    </link>

    <link xconnector="onEndStop">
        <bind role="onEnd" component="video" />
        <bind role="stop" component="imagen" />
    </link>
</body>
```

4.5.1.5. Iniciando un objeto de media cuando otro termina.

El objetivo de este ejemplo es sincronizar la terminación de una media con el inicio de otra, haciendo que, al final del video "*videoIntro*", inicie el video (*videoPrincipal*) en la misma región. Además de la creación del nuevo nodo media correspondiente al *video*, es necesario crear un enlace para conectar los dos videos, a través del conector ***onEndStart***. Este conector desencadena la visualización del nodo destino cuando la presentación del nodo origen se termina.

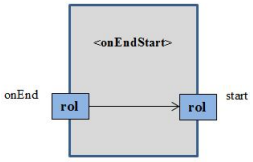
Al iniciar el documento, ***videoApertura*** empieza (puerta ***InVideo*** en el ***body***). Cuando se inicia ***videoIntro***, también lo hace ***"imagen"*** (enlace que utiliza el conector ***onBeginStart***). Cuando ***videoIntro*** termina inicia ***videoPrincipal*** (enlace que utiliza el conector ***onEndStart***), cuando termina la presentación de este segundo video finaliza la visualización de ***imagen*** (enlace que utiliza el conector ***onEndStop***). Los pasos para la creación de este ejemplo se detallan a continuación.

Paso 1: Modificando el elnlace para incluir un retardo.

Definir la media correspondiente al segundo video (*videoPrincipal*), en el contexto <*body*> del documento. En este ejemplo se puede observar que el nodo *videoPrincipal* se va a mostrar de manera similar y en la misma región, por lo que no es necesario crear una nueva región y otro descriptor para el nuevo nodo.

Paso 2: Definición del conector *onEndStart*.

El conector ***onEndStart*** debe ser definida conforme la Tabla 4.9.

Nombre:	**onEndStart**
Condición:	Ternino de exhibición de un nodo (ligado a la función **onEnd**)
Acción:	Inicia la exhibición del nodo (ligado a la función **start**)
Ilustración:	. 
Código NCL:	<causalConnector id="onEndStart"> <simpleCondition role="onEnd"/> <simpleAction role="start"/> </causalConnector>
Lectura:	Cuando el nodo ligado a la función ***onEnd*** inicia, inicia el nodo ligado a la función ***start***.

Cuadro 4.9: Definición del conector ***onEndStart***

De la misma manera que en los ejemplos anteriores, el conector ***onEndStart*** debe ser dclarado en la sección ***connectorBase***.

Paso 3: Definición de un enlace de sincronismo.

Éste se define de una manera similar a los enlaces anterior, con *videoIntro* en la función ***onEnd*** y *videoPrincipal* en la función ***start.***

Codigo del ejemplo 5.

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

```
        </causalConnector>

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

        <causalConnector id="onEndStop">
            <simpleCondition role="onEnd" />
            <simpleAction role="stop" />
        </causalConnector>
    </connectorBase>
</head>

<body>
    <port id="InVideo" component="videoIntro"/>

    <media id="videoIntro" type="video/mpeg" src="media/video1.mpg"
     descriptor="desVideo"/>
    <media id="videoPrincipal" type="video/mpeg"
        src="media/videoPrin.mpg"
     descriptor="desVideo"/>
    <media id="imagen" type="image/jpeg" src="media/Logo.jpg"
     descriptor="desImagen" >
          <property name="transparency" value="50 %"/>
    </media>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="videoIntro" />
        <bind role="start" component="imagen" />
    </link>

    <link xconnector="onEndStart">
        <bind role="onBegin" component="videoIntro" />
        <bind role="start" component="videoPrincipal" />
    </link>

    <link xconnector="onEndStop">
        <bind role="onEnd" component="videoPrincipal" />
        <bind role="stop" component="imagen" />
    </link>
```

4.5.2. Ejemplo con Interacción con el usuario:

4.5.2.1. Interrumpiendo un video e iniciando otro conforme la interacción con el usuario.

Modificaremos el ejemplo anterior para que sea el usuario el que inicie el segundo video (*videoPrincipal*) presionando la tecla verde del control remoto, sin tener que esperar a que finalice el video (*videoIntro*).

Para realizar la interacción del usuario con el control remoto se necesita de un nuevo conector, uno que utilice la función predeterminada ***onSelection***, y que recibe como parámetro el código de la tecla que se desea capturar. El comportamiento del enlace que use el conector debe ser: durante la exhibición de boton, cuando la tecla verde (*GREEN*) del control remoto sea presionada, y el vídeo de

Nombre:	***onKeySelectionStop***
Condición:	Tecla ***aTecla*** acionada (función ***onSelection***)
Acción:	Para el nodo ligado a la función ***stop***
Código NCL:	<causalConnector id="onKeySelectionStop"> <connectorParam name="keyCode" /> <simpleCondition role="onSelection" key="$keyCode"/> <simpleAction role="stop"/> </causalConnector>
Lectura:	Cuando el nodo ligado a la función***onSelection*** es presentado, cuando la tecla ***aTecla*** es presionada, para el nodo ligado a la función ***stop***.
Observaciones:	Los enlaces que utilice este conector deben identificar, como un parámetro adicional la conexión con el nodo de origen (***bindParam*** o ***linkParam***), el código virtual de la tecla del control remoto asociada con la selección, mediante el parámetro ***keyCode***. Por ejemplo: <bind component="imgPularIntro" role="onSelection"> <bindParam name="aTecla" value="GREEN" /> </bind>

Cuadro 4.10: Conector ***onKeySelectionStop***.

apertura (*videoIntro*) está siendo presentado. Al presionar la tecla verde el enlace debe terminar la presentación del vídeo de introducción y la exhibición de la imagen boton, además iniciara la presentación del vídeo principal (*videoPrincipal*) cuando es activado, resultando en comportamiento deseado.

Para que el espectador sepa que tecla puede presionar a cada instante, es importante que una media sea exhibida que indique las oportunidades para la interacción. Esto se hace a través de la media *boton*, que hasta el momento no tenía ningún comportamiento de interactividad asociada a la misma.

En este ejemplo se debe añadir un enlace que utiliza el conector ***onKeySelectionStop***, para capturar las pulsaciones de la tecla verde (*GREEN*) del control remoto y detener el vídeo de introducción y mostrar el video principal.

Paso 1: Definir el conector.

Los conectores que se utiliza en este ejemplo son: ***onBeginStart***, ***onEndStop***, ***onEndStart*** y ***onKeySelectionStop***. Los conectores ***onBeginStart***, ***onEndStop*** y ***onEndStart*** fueron explicados previamente. La siguiente Tabla 4.10 presenta la definición de conector ***onKeySelectionStop***:

El conector ***onKeySelectionStop*** deve ser definido en la sección ***connectorBase***:

Paso 2: Definición del enlace de interactividad.

El enlace debe ser especificado de forma semejante a los anteriores, con boton en el papel ***onSelection*** (con parámetro ***keyCode*** igual a *GREEN*, correspondiente a la tecla verde del control remoto) y *videoIntro* en el papel ***stop***. Se debe señalar que, como el papel ***onSelection*** está asociado a la media *boton*, y exige que el medio este siendo presentado para que el vínculo pueda ser activado. Mientras

que la exhibición de la media *boton* no sea iniciada, este enlace no puede ser activado.

Codigo del ejemplo 6.

```
<head>
    .....
    <connectorBase>
        .....
        <causalConnector id="onKeySlectionStop">
        <connectorParam name="keyCode"/>
            <simpleCondition role="onSelection" key="$keyCode" />
            <simpleAction role="stop" />
        </causalConnector>
    </connectorBase>
</head>

<body>
    <port id="InVideo" component="videoIntro"/>

    .....
    <media id="boton" type="image/jpeg" src="media/boton.jpg"
     descriptor="desImagen" >
          <property name="transparency" value="50 %"/>
    </media>
     .....
    <link xconnector="onKeySlectionStop">
        <bind role="onSelection" component="boton" >
            <bindParam name="keyCode" value="GREEN"/>
        </bind>
        <bind role="stop" component="videoIntro" />
    </link>
</body>
```

4.5.3. Ejemplo de importacion de bases de otros archivos:

4.5.3.1. Trabajando con bases de conectores en archivos separados.

Como se mencionó anteriormente existen dos formas para definir y utilizar conectores: la primera es definiéndolos en el propio hiperdocumento, que se utilizaron en los ejemplos anteriores y la segunda es creando un nuevo archivo de éstos e importando la sección ***connectorBase***. En el ejemplo anterior, se adoptó la primera estrategia mientras que en este ejemplo se adopta la segunda, separando el archivo de conectores e importando al programa principal .

Paso 1: Creando archivos de conectores.

Crear un archivo NCL con los conectores definidos en el ejemplo, éstos se declaran en el esqueleto básico de un documento NCL que fue explicado anteriormente. Como se ha creado una base de conectores, para cualquier media, no es necesario especificar la sección ***body***. Solo se debe crear un bloque ***connectorBase*** con los conectores que serán utilizados por el programa. A partir del esqueleto básico, se puede copiar los conectores que se utilizaron por ejemplo anterior, complementando el archivo de conectores, aquí llamado ***conectoresV1.ncl*** .

Paso 2: Importando la base de conectores.

Para importar los conectores al programa principal, se utiliza el elemento ***importBase*** dentro del elemento ***connectorBase***, en el documento NCL principal (***programa_principal***), en sustitución de la toda la configuración de los conectores que hubo en el ejemplo anterior.

Paso 3: Modificando los enlaces para utilizar los conectores importados.

Para utilizar los conectores definidos en el archivo importado, se debe alterar en cada enlace, la referencia al conector correspondiente para incluir el alias (*menuConectores*) de la base de conectores importados, en el formato:

alias_da_base#id_do_conector.

Este ejemplo solo tiene los conectores definidos en otro documento NCL, el resto de los atributos se establecen de la misma manera que en el anterior.

Codigo del ejemplo 7.

Codigo del archivo NCL principal ***"programa_principal"***:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!– Generated by NCL Eclipse –>
<ncl id="programa_principal".....>
  <head>
    ......................
    <connectorBase>
        <importBase documentURI="conectoresV1.ncl"
        alias="menuConectores"/>
    </connectorBase>
  </head>

  <body>
    ......................
    <link xconnector="menuConectores#onBeginStart">
        ......................
    </link>

    <link xconnector="menuConectores#onEndStart">
        ......................
    </link>

    <link xconnector="menuConectores#onKeySelectionStop">
        ......................
    </link>

    <link xconnector="menuConectores#onEndStop">
        ......................
    </link>
  </body>
```

Codigo del archivo NCL de conectores ***"conectoresV1"***:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!– Generated by NCL Eclipse –>
<ncl id="conectoresV1".....>
  <head>
```

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

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

    <causalConnector id="onBeginStartDelay">
        <connectorParam name="oRetardo"/>
        <simpleCondition role="onBegin" />
        <simpleAction role="start" delay="$oRetardo"/>
    </causalConnector>

    <causalConnector id="onKeySlectionStop">
    <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode" />
        <simpleAction role="stop" />
    </causalConnector>

    <causalConnector id="onEndStop">
        <simpleCondition role="onEnd" />
        <simpleAction role="stop" />
    </causalConnector>
</connectorBase>
</head>
```

4.5.4. Ejemplo de sincronización con un tramo de media:

4.5.4.1. Sincronización una imagen con un tramo de video.

Como continuación del ejemplo anterior, esta sección presenta un programa que en un determinado tramo de vídeo principal (*videoPrincipal*) presenta una imagen (*boton*). En el siguiente ejemplo, se asociará un comportamiento interactivo a esta imagen. En este ejemplo, sólo se mostrara la imagen pero sin interactividad con el control remoto.

Para definir el extracto del vídeo en el que la imagen debe aparecer, es necesario crear un ancla de contenido (elemento <*area*>) para el vídeo, estableciendo el intervalo de visualización de la imagen. Cuando el ancla empieza se debe exhibir la imagen, usando el conector ***onBeginStart*** y cuando termina debe finalizar la visualización de la imagen, usando el conector ***onEndStop***.

Paso 1: Definición un nodo de media para indicar oportunidad de interacción.
Se defiine en la sección **<body>** la media (*boton*) correspondiente a la imagen que indicará la posibilidad de interacción.

Paso 2: Definiendo el ancla en el video principal.

Para definir cuando la imagen debe iniciar y finalizar su exhibición, es necesario crear un ancla de contenido que delimita un tramo del vídeo. En NCL, un ancla de contenido se declara utilizando el elemento ***area*** anidado al elemento media correspondiente. En el caso de medios continuos, los atributos ***begin*** y ***end*** indican en que tiempo, en segundos, es el principio y el final del ancla. En este ejemplo la imagen (*boton*) sera exhibida entre 2 y 10 segundos del video principal.

Paso 3: Crear enlaces de sincronismo a la imagen

Crear enlaces a partir de anclas, es similar a crearlos a partir de nodos. La diferencia es que en la hora de la conexión, además de especificar el nodo (a través del atributo ***component*** del elemento ***bind***), se debe establecer también el ancla que debe estar conectada (atributo ***interface*** del elemento ***bind***) asociados a la función del conector.

Codigo del ejemplo 8.

```
<head>
  .....
  <connectorBase>
      <importBase documentURI="conectoresV1.ncl"
      alias="menuConectores"/>
  </connectorBase>
</head>

<body>
  .....
  <media id="videoPrincipal" type="video/mpeg"
      src="Videos/video1.mpg"
  descriptor="desVideo">
      <area id="area1" begin="2s" end="10s" />
  </media>

  <link xconnector="menuConectores#onEndStart">
      .....
  </link>

  <link xconnector="menuConectores#onBeginStart">
      <bind role="onBegin" component="videoPrincipal"
      interface="area1" />
      <bind role="start" component="boton" />
  </link>

  <link xconnector="menuConectores#onEndStop">
      <bind role="onEnd" component="videoPrincipal"
      interface="area1" />
      <bind role="stop" component="boton" />
  </link>
</body>
```

4.5.5. Ejemplo de la estructura de un programa NCL:

4.5.5.1. Exhibición de un contexto.

Cuando se presiona el botón rojo, el programa inicia las medias de un contexto (asignadas a las puertas del contexto) que representa un menú, cuyo comportamiento se define en los siguientes ejemplos. En este ejemplo, el contexto tiene cuatro medios de imagen, *menuItem1* a *menuItem4*. Para que todas las medias de imagen (items) se inicien cuando el contexto menú empieza, se declaran cuatro puertas, una para cada una de ellas. La visión del diseño se muestra en la Figura 4.12.

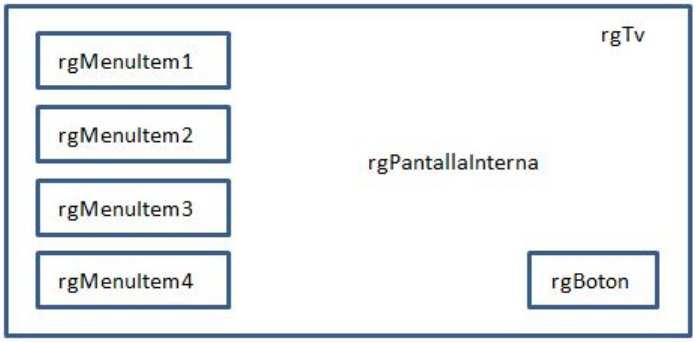


Figura 4.12: Visión de diseño de ejemplo.
Fuente: Referencia personal.

Paso 1: Definición de las regiones en donde los ítems del menú serán presentados.
Las regiones para la presentación de los nodos media de los ítems del menú, se definen de la misma manera que en los ejemplos anteriores en la sección <*regionBase*> en el encabezado del documento. Para asegurarse que las imágenes se presentarán en una capa por encima de las otras capas, se utiliza el atributo de región ***zIndex***.

Paso 2: Definición de los descriptores para los ítems del menú.
Se definen de forma similar que en los ejemplos anteriores en la región <*descriptorBase*> en el encabezado del documento, un descriptor para la presentación de cada uno de los nodos media de los ítems del menú.

Paso 3: Definición de los contextos con las medias correspondientes a los ítems del menú.
Para mejorar la estructurara de la sección <*body*> de un documento, se pueden utilizar contextos (elemento <*context*>) a fin de abarcar algunos medios fuertemente relacionados, como es el caso de los elementos del menú.

Paso 4: Definición de las puertas para las medias del contextos.
Para que un contexto esté completo es necesario crear las puertas de las medias, para hacerlos accesibles para los elementos que están fuera del mismo.

Paso 5: Definición de los enlaces que muestra el contexto de los medios.
Finalmente se debe definir un enlace para cuando el usuario presione el botón rojo del control remoto, se muestren los medios del contexto, pero no es necesario

iniciar a cada item. Iniciar el menú de contexto significa iniciar todos los medios asignados por sus puertas (*pMenuItem1* a *pMenuItem4*), es decir *menuItem1* a *menuitem4*. Para este ejemplo se utilizó el conector ***onKeySelectionStartStop*** que realiza dos acciones simultáneamente, detiene la exhibición de la imagen (*boton*) y muestra el menú en la pantalla. Además se creó un documento NCL ("*connectorBase.ncl*") que posee todos los conectores y puede ser importado para este y los demás ejemplos, este documento se encuentra detallado en el Apéndice D.

Codigo del ejemplo 9.

```
<head>
  <regionBase>
    .....
    <region id="rgMenu" left="7 %" top="30 %" width="200"
    height="70 %"
    zIndex="99">
      <region id="rgMenuItem1" top="0" height="50" />
      <region id="rgMenuItem2" top="60" height="50" />
      <region id="rgMenuItem3" top="120" height="50" />
      <region id="rgMenuItem4" top="180" height="50" />
</region>
  </regionBase>

  <descriptorBase>
    .....
    <descriptor id="dMenuItem1" region="rgMenuItem1" />
    <descriptor id="dMenuItem2" region="rgMenuItem2" />
    <descriptor id="dMenuItem3" region="rgMenuItem3" />
    <descriptor id="dMenuItem4" region="rgMenuItem4" />
  </descriptorBase>
  <connectorBase>
    <importBase documentURI="connectorBase.ncl"
    alias="menuConectores"/>
  </connectorBase>
 </head>

 <body>
  .....
  <context id="menu">
    <port id="pMenuItem1" component="menuItem1" />
    <port id="pMenuItem2" component="menuItem2" />
    <port id="pMenuItem3" component="menuItem3" />
    <port id="pMenuItem4" component="menuItem4" />

    <media id="menuItem1" src="Imagenes/Item1.png"
    descriptor="dMenuItem1" >
      <property name="transparency" value="20 %"/>
    </media>
    <media id="menuItem2" src="Imagenes/Item2.png"
    descriptor="dMenuItem2" >
      <property name="transparency" value="20 %"/>
    </media>
```

```
        <media id="menuItem3" src="Imagenes/Item3.png"
        descriptor="dMenuItem3" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem4" src="Imagenes/Item4.png"
        descriptor="dMenuItem4" >
            <property name="transparency" value="20 %"/>
        </media>
    </context>

    <link xconnector="menuConectores#onBeginStart">
        .....
    </link>

    <link xconnector="menuConectores#onEndStart">
        .....
    </link>

    <link xconnector="menuConectores#onKeySelectionStartStop">
        <bind component="boton" role="onSelection">
            <bindParam name="keyCode" value="RED"/>
        </bind>
        <bind component="boton" role="stop"/>
        <bind role="start" component="menu"/>
    </link>

    <link xconnector="menuConectores#onEndStop">
        <bind role="onEnd" component="videoPrincipal" />
        <bind role="stop" component="menu" />
    </link>
</body>
```

4.5.6. Ejemplo de Manipulación de propiedades de medias:

4.5.6.1. Redimensionamiento de video durante la exhibición del menú.

Este ejemplo se diferencia del anterior, en que cuando el menú es inicializado se redimensiona el video principal y se muestra una imagen de fondo.

Además tiene más de una región para el fondo, con ***zIndex*** inferior al ***zIndex*** de vídeo principal. Aunque el video cambia de tamaño, la visión de diseño sólo muestra su posición inicial y dimensiones. Cuando un video es presentado solo se puede manipular las propiedades de los medios, pero no de las regiones. En otras palabras, una región define la localización y las dimensiones en las que los medios serán presentados inicialmente. Estas propiedades afectan directamente a los nodos de media, donde luego puede ser manipulado por los enlaces. Los conectores que se emplean están en el documento "*connectorBase.ncl*" (Apéndice D).

Paso 1: Definición de la regione para la imagen de fondo.

La imagen de fondo debe ser mostrada por detrás del video principal. Por esta razón es necesario crear una nueva región (*regFondo*) para la presentación del nodo media (*Fondo*).

Paso 2: Definición del descriptor para la imagen de fondo.

Se definen de la misma manera que en ejemplos anteriores en la región <*descriptorBase*> en el encabezado del documento, un descriptor (*dFondo*) para la presentación del nodo media de la imagen de fondo.

Paso 3: Definición del nodo media para la imagen de fondo.

Se crea de manera similar que en ejemplos anteriores un nodo media (*fondo*) que contenga la imagen de fondo para ser mostrada.

Paso 4: Definición del ancla de propiedades del video.

Para manipular las propiedades ***top***, ***left***, ***width*** y ***height*** de la región a la que el vídeo esta asociado, se debe definir explícitamente tales propiedades como anclas de la media. En el caso concreto de estas características de posición y dimensiones, se puede especificar un ancla para la propiedad ***bounds*** para manipularlas en conjunto, en el orden de ***left***, ***top***, ***width***, ***height***:

```
<media id="videoPrincipal" src="media/video_P.mpg"
descriptor="desPantallaIn">
    <area id="area1" begin="2s" end="10s" />
    <property name="bounds" />
</media>
```

Paso 5: Definición del conector que permite manipular una propiedad.

Para manipular una propiedad, se puede definir un conector que utiliza la función de acción predeterminada set. Para este ejemplo, es necesario un conector que se lea: "cuando un nodo X inicia, alterando el valor de una propiedad P de un nodo Y". Como las funciones que se utilizan se llaman ***onBegin*** y set, el nuevo conector se llama ***onBeginSet***, especificado en la Tabla 4.11:

Nombre:	***onBeginSet***
Condición:	Inicio de exhibicion de la media con la función ***onBegin.***
Acción:	Altera la propiedad del nodo asociado a la función ***set***, conforme el valor pasa por la conexión (***bindParam***) para el parametro del conector (***connectorParam***) denominado ***var***.
Código NCL:	<causalConnector id="onBeginSetStart"> <connectorParam name="var" /> <simpleCondition role="onBegin"/> <simpleAction role="set" value="$var"/> </causalConnector>
Observaciones:	Los enlaces que utilice este conector deben identificar, como un parámetro adicional la conexión del nodo (***bindParam***), las nuevas coordenadas y dimenciones (x,y,w,h). Por ejemplo. <bind component="videoPrincipal" interface="bounds" role="set"> <bindParam name="oValor" value="45 %,25 %,40 %,40 %" /> </bind>

Cuadro 4.11: Conector **onBeginSetStart.**

Es importante tener en cuenta que los valores en el nombre del parámetro definido en el conector ***valor***, se utilizan por la conexión (elemento ***bindParam***), para el paso de los limites (propiedad ***bounds***) de la media *videoPrincipal*, especificando los atributos ***interface*** y ***component*** de la ligación (elemento ***bind***). Si se usa la propiedad ***"size"***, se deben pasar solamente dos parámetros ***width*** y ***height.***

Paso 6: Definición del enlace que manipula la propiedad.

La manipulación de esta propiedad se realiza a través de un enlace que conecta la propiedad ***bounds*** del nodo al papel ***set*** y utiliza el conector ***onBeginSet***, que se puede leer de la siguiente manera: cuando el nodo en el menú de inicio, se cambia el valor de la propiedad ***bounds*** del nodo *videoPrincipal* para " 45 %, 25 %,40 %,40 %". Una observación importante: cuando se utilizan porcentajes para modificar el tamaño y posición de una media, este es siempre relativa al tamaño y posición actual de la media.

Paso 7: Definición del enlace que exhibe la imagen de fondo.

Por último es necesario exhibir la imagen de fondo cuando menú es accionado, utilizando el conector ***onBeginStart.***

Codigo del ejemplo 10.

```
<head>
  <regionBase>
    <region id="rgTv">
      <region id="regFondo" zIndex="0"/>
      <region id="regPantallaIn" zIndex="1" />
      <region id="regImagen" left="89 %" top="5 %" width="10 %"
      height="10 %" zIndex="2" />
      <region id="regBoton_R" left="3 %" top="88 %" width="7 %"
      height="7 %" zIndex="2" />
      <region id="regBoton_V" left="90 %" top="88 %" width="7 %"
      height="7 %" zIndex="2" />

      <region id="rgMenu" left="7 %" top="30 %" width="200"
      height="70 %" zIndex="99">
        <region id="rgMenuItem1" top="0" height="50" />
        <region id="rgMenuItem2" top="60" height="50" />
        <region id="rgMenuItem3" top="120" height="50" />
        <region id="rgMenuItem4" top="180" height="50" />
      </region>
    </region>
  </regionBase>

  <descriptorBase>
    <descriptor id="desImagen" region="regImagen" />
    <descriptor id="desBoton_R" region="regBoton_R" />
    <descriptor id="desBoton_V" region="regBoton_V" />
    <descriptor id="desFondo" region="regFondo"/>
    <descriptor id="desPantallaIn" region="regPantallaIn" />

    <descriptor id="dMenuItem1" region="rgMenuItem1" />
```

```
        <descriptor id="dMenuItem2" region="rgMenuItem2" />
        <descriptor id="dMenuItem3" region="rgMenuItem3" />
        <descriptor id="dMenuItem4" region="rgMenuItem4" />
    </descriptorBase>
    <connectorBase>
        <importBase documentURI="connectorBase.ncl"
        alias="menuConectores"/>
    </connectorBase>
</head>

<body>
    <port id="InVideo" component="videoIntro"/>

    <media id="videoIntro" type="video/mpeg" src="Videos/video_I.mpg"
    descriptor="desPantallaIn"/>
    <media id="videoPrincipal" type="video/mpeg" src="Videos/video_P"
    descriptor="desPantallaIn">
        <area id="area1" begin="2s" end="15s"/>
        <property name="bounds"/>
    </media>
    <media id="boton_R" type="image/png" src="Imagenes/B-rojo.png"
    descriptor="desBoton_R" >
        <property name="transparency" value="20 %"/>
    </media>
    <media id="boton_V" type="image/png" src="Imagenes/B-verde.png"
    descriptor="desBoton_V" >
        <property name="transparency" value="20 %"/>
    </media>
    <media id="fondo" type="image/jpeg" src="Imagenes/ferrari1.jpg"
    descriptor="desFondo"/>

    <context id="menu">
        <port id="pMenuItem1" component="menuItem1" />
        <port id="pMenuItem2" component="menuItem2" />
        <port id="pMenuItem3" component="menuItem3" />
        <port id="pMenuItem4" component="menuItem4" />

        <media id="menuItem1" src="Imagenes/Item1.png"
        descriptor="dMenuItem1" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem2" src="Imagenes/Item2.png"
        descriptor="dMenuItem2" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem3" src="Imagenes/Item3.png"
        descriptor="dMenuItem3" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem4" src="Imagenes/Item4.png"
        descriptor="dMenuItem4" >
            <property name="transparency" value="20 %"/>
```

```
    </media>
</context>
<!– logo ferrari –>
<media id="imagen" type="image/png" src="Imagenes/Imagen2.png"
descriptor="desImagen" />
<link xconnector="menuConectores#onBeginStart">
    <bind role="onBegin" component="videoIntro" />
    <bind role="start" component="imagen"/>
</link>
<link xconnector="menuConectores#onEndStop">
    <bind role="onEnd" component="videoPrincipal"/>
    <bind role="stop" component="imagen"/>
</link>
<!–.....–>
<link xconnector="menuConectores#onBeginStartDelay">
    <bind role="onBegin" component="videoIntro" />
    <bind role="start" component="boton_R" >
        <bindParam name="oRetardo" value="3s"/>
    </bind>
</link>
<link xconnector="menuConectores#onBeginStart">
    <bind role="onBegin" component="videoIntro" />
    <bind role="start" component="boton_V"/>
</link>
<link xconnector="menuConectores#onEndStop">
    <bind role="onEnd" component="videoIntro"/>
    <bind role="stop" component="boton_V"/>
</link>
<link xconnector="menuConectores#onEndStart">
    <bind role="onEnd" component="videoIntro" />
    <bind role="start" component="videoPrincipal"/>
</link>
    <link xconnector="menuConectores#onKeySelectionStartStop">
    <bind component="boton_R" role="onSelection">
        <bindParam name="keyCode" value="RED"/>
    </bind>
    <bind component="boton_R" role="stop"/>
    <bind role="start" component="menu"/>
</link>
<link xconnector="menuConectores#onEndStop">
    <bind role="onEnd" component="videoPrincipal"/>
    <bind role="stop" component="menu"/>
</link>
<link xconnector="menuConectores#onBeginSet">
    <bind component="boton_R" role="onBegin"/>
    <bind component="videoPrincipal" interface="bounds" role="set">
        <bindParam name="aValue" value="38 %,30 %,50 %,50 %"/>
    </bind>
</link>>
<link xconnector="menuConectores#onBeginStart">
```

```
        <bind role="onBegin" component="videoPrincipal" />
        <bind role="start" component="fondo"/>
    </link>
    <link xconnector="menuConectores#onEndStop">
        <bind role="onEnd" component="videoPrincipal"/>
        <bind role="stop" component="fondo"/>
    </link>
    <link xconnector="menuConectores#onKeySelectionStop">
        <bind component="boton_V" role="onSelection">
            <bindParam name="keyCode" value="GREEN"/>
        </bind>
        <bind component="videoIntro" role="stop"/>
    </link>
    <link xconnector="menuConectores#onEndStop">
        <bind role="onEnd" component="videoPrincipal"/>
        <bind role="stop" component="boton_R"/>
    </link>
</body>
```

4.5.7. Ejemplo de Importacion de programas NCL:

4.5.7.1. Definición del menú en un programa NCL separado.

Este ejemplo difiere del anterior en el desarrollo del menú en un documento NCL separado e importándolo al documento principal. Basándose en el ejemplo anterior se deben adicionar los siguientes pasos.

Paso 1: Creando el archivo del menú.

El primer paso es crear un nuevo documento NCL (*menuV1.ncl*) que contenga el desarrollo del menú, incluyendo los descriptores, las regiones y los elementos de media correspondientes. Como se puede ver el documento *menuV1.ncl* es un programa completo por lo que podría ser ejecutado individualmente.

Paso 2: Importación del archivo del menú.

La importación del archivo NCL *menuV1.ncl* se la realiza en la cabecera del programa principal (sección <*head*>), por medio de la etiqueta <*importNCL*> dentro del elemento <*importDocumentBase*> y debe ser realizada antes de las regiones y de los descriptores del programa principal, si no se presentan conflictos al momento de la ejecución del programa. De la misma manera que se realizó cuando se importó las bases de conectores, se debe definir un valor para el atributo alias para identificar al archivo importado en el documento principal.

```
<importedDocumentBase>
    <importNCL documentURI="menuV1.ncl" alias="menuID" />
</importedDocumentBase>
```

Paso 3: Definir un contexto que haga referencia al contexto importado.

Para utilizar los elementos del documento importado, se debe crear un elemento en el programa principal (sección <*body*>) que haga referencia a ellos, a través del atributo ***refer***. En el ejemplo el contexto *menu* del documento importado

(*alias=menuID*) es referido en el programa principal, por el elemento también llamado *menu*.

documento_principal.ncl

```
<context id="menu" refer="menuID#menu" />
```

Paso 4: Modificar los enlaces que utilice el elemento importado.

En el caso de este ejemplo, no es necesario cambiar los enlaces que utilice el elemento importado, ya que el identificador id del contexto importados (*menu*) es el mismo ***id*** del contexto cuando se encontraba creado dentro del programa principal, en el ejemplo anterior.

Codigo del ejemplo 11.

Codigo del programa principal.

```
<ncl id="documento_principal" .....>
<head>
  <importedDocumentBase>
      <importNCL documentURI="menuV1.ncl" alias="menuID" />
  </importedDocumentBase>

  <regionBase>
      <region id="rgTv">
      ..................
      </region>
  </regionBase>

  <descriptorBase>
      ..................
  </descriptorBase>

  <connectorBase>
      <importBase documentURI="connectorBase.ncl"
      alias="menuConectores"/>
  </connectorBase>
</head>

<body>
  <port id="InVideo" component="videoIntro"/>
  <context id="menu" refer="menuID#menu" />
      ..................
</body>
</ncl>
```

Codigo del menuV1.

```
<ncl id="menuV1" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
  <regionBase>
      <region id="rgMenu" left="7%" top="30%" width="200"
      height="70%" zIndex="99">
          <region id="rgMenuItem1" top="0" height="50" />
          <region id="rgMenuItem2" top="60" height="50" />
          <region id="rgMenuItem3" top="120" height="50" />
          <region id="rgMenuItem4" top="180" height="50" />
```

```
    </region>
    </regionBase>

    <descriptorBase>
        <descriptor id="dMenuItem1" region="rgMenuItem1" />
        <descriptor id="dMenuItem2" region="rgMenuItem2" />
        <descriptor id="dMenuItem3" region="rgMenuItem3" />
        <descriptor id="dMenuItem4" region="rgMenuItem4" />
    </descriptorBase>
  </head>

  <body>
    <context id="menu">
        <port id="pMenuItem1" component="menuItem1" />
        <port id="pMenuItem2" component="menuItem2" />
        <port id="pMenuItem3" component="menuItem3" />
        <port id="pMenuItem4" component="menuItem4" />

        <media id="menuItem1" src="Imagenes/Item1.png"
        descriptor="dMenuItem1" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem2" src="Imagenes/Item2.png"
        descriptor="dMenuItem2" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem3" src="Imagenes/Item3.png"
        descriptor="dMenuItem3" >
            <property name="transparency" value="20 %"/>
        </media>
        <media id="menuItem4" src="Imagenes/Item4.png"
        descriptor="dMenuItem4" >
            <property name="transparency" value="20 %"/>
        </media>
    </context>
  </body>
</ncl>
```

4.5.8. Ejemplo de Navegación entre nodos de media:

4.5.8.1. Navegando con las flechas del control remoto.

Este ejemplo agrega un comportamiento de desplazamiento de los elementos del menú del ejemplo anterior. El ítem en foco tiene su imagen alterada y un marco añadido para reflejar cual sería el ítem activado al presionar el botón de activación (*OK*, entrar o equivalente). Todas las modificaciones se realizan en el documento NCL *menuV1*, sin alterar en nada el documento principal, ya que se mantiene el mismo nombre del documento importado.

Hasta ahora los ejemplos solo hicieron uso de las teclas de colores, que en la TV digital son generalmente cuatro (Apéndice C). Cuando hay situaciones en las que se les ofrece a los usuarios más de cuatro opciones, se hace necesario presentar otra estrategia de selección, con frecuencia se hace uso de las flechas del control

remoto para identificar una opción, seguido de la tecla de "*OK*" para activar el enlace correspondiente a la selección realizada.

Para lograr este tipo de menú, se utiliza una organización vertical, según el diagrama de la Figura4.13, que considera que las opciones están ordenadas. Comienza con la primera opción en foco. Si la tecla abajo es presionada, la siguiente opción recibe el foco. La tecla arriba se utiliza para mover el foco a la opción anterior en la lista. Se observa que este régimen no representa un menú circular, ya que al pulsar la flecha hacia abajo hasta la última opción no se selecciona la primera, así como la tecla de flecha hacia abajo en la primera opción no selecciona la última. Después de seleccionar la opción deseada y el usuario puede presionar el botón clave "*OK*" (código virtual Enter) para activar el enlace correspondiente.

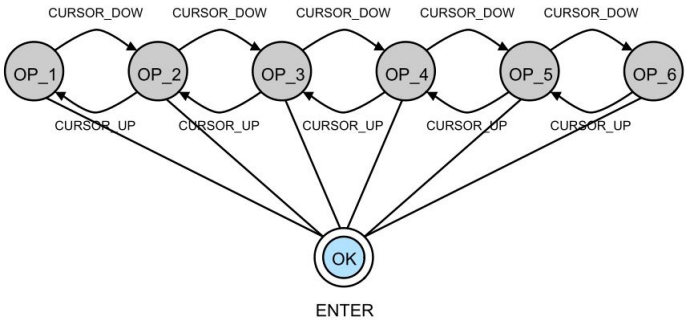


Figura 4.13: Esquema de selección de una de entre 6 opciones del menú.
Fuente: Referencia personal.

La navegación por teclas está establecida en los descriptores de diferentes botones, cada uno con sus atributos ***focusIndex*** definidos, que en este caso varían de 1 a 6. Los atributos ***moveDown*** y ***moveUp*** de cada descriptor indican a que opción debe cambiar el foco cuando las teclas *DOW* y *UP* fueron presionadas, respectivamente. En este tipo de menú, es esencial que el usuario este informado sobre la selección actual. Por esta razón, se utiliza el concepto de foco introducido en NCL 3,0. Se puede especificar un color y el ancho del elemento del marco cuando tenga el foco por los atributos ***focusBorderColor*** y ***focusBorderWidth***, como se muestra en la Figura 4.14, que indica que opción está siendo seleccionada en ese instante. La descripción de todas las características que posee un descriptor se detallaron en la su subsección 4.2.1.2. En los siguientes pasos se describen los cambios que se deben hacer al ejemplo anterior sobre todo en los descriptores del menú para mostrar un margen cuando esta con el foco uno de los items.

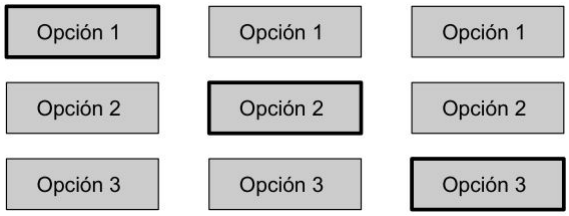


Figura 4.14: Ejemplo de indicación de opción actual de selección.
Fuente: Referencia personal.

Paso 1: Definición el foco de cada descriptor.

Cada descriptor correspondiente a un ítem del menú debe establecer un valor

para el atributo ***focusIndex***, para que pueda recibir el foco, siendo el de menor valor el primero en ser mostrado.

Paso 2: Definición de los descriptores de cada elemento que recibe el foco a cada pulsación de una de las teclas de la flecha.

Este ejemplo posee los ítems del menú colocados verticalmente, además utiliza solo los atributos ***moveUp*** y ***moveDown*** definidos para que la navegación sea efectuada por las teclas de hacia arriba y hacia abajo. La navegación que se emplea en el ejemplo es circular, es decir si el foco está en el primer ítem (con ***focusIndex***="1") y se presiona la tecla hacia arriba (***moveUp***) el foco se desplaza hacia el ultimo ítem (con ***focusIndex***="4").

Paso 3: Definición las medias y margenes de los descriptores de los elementos en foco.

Se define una imagen en el atributo ***focusSrc*** y un recuadro en los atributos ***focusBorderColor focusBorderWidth*** (color y ancho del marco respectivamente) para cada descriptor cuando estén con el foco.

Paso 4: Definición las medias y margenes de los descriptores de los elementos en foco.

En este ejemplo se establece un color de trama para cuando el usuario presiona la tecla de activación mientras el elemento asociado con el descriptor está con el foco, esto se define en el atributo ***selBorderColor***.

Codigo del ejemplo 12.

```
<ncl id="menuV1" .....>
 <head>
   <regionBase>
       .....
   </regionBase>
   <descriptorBase>
       <descriptor id="dMenuItem1" region="rgMenuItem1"
       focusIndex="1" moveUp="4" moveDown="2"
       focusBorderWidth="-3" focusBorderColor="yellow"
       focusSrc="Imagenes/FocoItem.png" selBorderColor="aqua"/>
       <descriptor id="dMenuItem2" region="rgMenuItem2"
       focusIndex="2" moveUp="1" moveDown="3"
       focusBorderWidth="-3" focusBorderColor="yellow"
       focusSrc="Imagenes/FocoItem.png" selBorderColor="aqua"/>
       <descriptor id="dMenuItem3" region="rgMenuItem3"
       focusIndex="3" moveUp="2" moveDown="4"
       focusBorderWidth="-3" focusBorderColor="yellow"
       focusSrc="Imagenes/FocoItem.png" selBorderColor="aqua"/>
       <descriptor id="dMenuItem4" region="rgMenuItem4"
       focusIndex="4" moveUp="3" moveDown="1"
       focusBorderWidth="-3" focusBorderColor="yellow"
       focusSrc="Imagenes/FocoItem.png" selBorderColor="aqua"/>
   </descriptorBase>
```

```
  </head>
  <body>
    <context id="menu">
        .....
    </context>
  </body>
</ncl>
```

4.5.8.2. Selección del elemento del menú utilizando teclas de activación.

Este ejemplo modifica el ejemplo anterior para iniciar un programa NCL definida en un archivo independiente cuando el usuario seleccione el primer elemento del menú. El programa importado consiste de un video en pantalla completa con 3 subtítulos sincronizados para ciertos tramos de video. El archivo importado se llama *programaV1.ncl*. Los siguientes pasos describen la modificación del ejemplo anterior para cumplir las especificaciones requeridas en esté.

Paso 1: Creando el archivo del programa importado.
El primer paso es crear un nuevo documento NCL (*prog1.ncl*) de forma semejante a ejemplos anteriores. Este programa contendrá tres anclas para los subtítulos.

Paso 2: Importación del programa leyendas.
La importación del archivo NCL se la realiza en la cabecera del programa principal de la misma manera que en el ejemplo 4.12. Igualmente se define un valor para el atributo ***alias*** (***prog***) para identificar al archivo importado.

Paso 3: Definir un contexto que reutiliza el programa importado.
Para utilizar los elementos del programa importado, se debe crear un elemento en el programa principal (sección <**body**>) que haga referencia a ellos, a través del atributo ***refer***. Este contexto *progV1* del documento importado es referenciado en el programa principal por un contexto llamado *progra01*, es decir de esta manera se referencia el <**contexto**> del documento importado.

Paso 4: Definir los enlaces que utilizan el programa importado.
Al seleccionar la primera opción del menú, debe también iniciar el contexto *progra01* y termina la presentación del menú. Cuando la ejecución del programa importado *progra01* termina, debe volver aparecer la presentación del menú. En el ejemplo a pesar que el conector ***onKeySelectionStartStop*** es capaz de recibir un valor de tecla ***keyCode*** como parámetro, si este valor no es pasado, el conector asume el comportamiento de ***onSelection*** sin una tecla definida, es decir, la selección sobre el elemento en foco a través de la tecla de activación (*OK*, ENTER o equivalente).

Codigo del ejemplo 13.
Codigo del programa con leyendas.

```
<ncl id="prog1" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
    <regionBase>
        <region id="regPantallaIn" zIndex="1" />
```

```
        <region id="regLeyenda" left="25%" width="50%" top="90%"
        zIndex="2" />
    </regionBase>

    <descriptorBase>
        <descriptor id="desLeyenda" region="regLeyenda">
            <descriptorParam name="border" value="none" />
        </descriptor>
        <descriptor id="desPantallaIn" region="regPantallaIn"
        explicitDur="40s"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
</head>

<body>
    <port id="InVi" component="progV1"/>
    <context id="progV1">
    <port id="pVideo1" component="video1"/>
    <media id="video1" src="Video/video.mpg"
    descriptor="desPantallaIn">
        <area id="a1" begin="2s" end="5s" />
        <area id="a2" begin="7s" end="9s" />
        <area id="a3" begin="11s" end="13s" />
    </media>

    <media id="leyenda1" src="Texto/leyenda1.htm"
        descriptor="desLeyenda" />
    <media id="leyenda2" src="Texto/leyenda2.htm"
        descriptor="desLeyenda" />
    <media id="leyenda3" src="Texto/leyenda3.htm"
        descriptor="desLeyenda" />

    <!- leyenda 1 ->
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a1" role="onBegin" />
        <bind component="leyenda1" role="start" />
    </link>

    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a1" role="onEnd" />
        <bind component="leyenda1" role="stop" />
    </link>

    <!- leyenda 2 ->
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a2" role="onBegin" />
        <bind component="leyenda2" role="start" />
    </link>

    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a2" role="onEnd" />
        <bind component="leyenda2" role="stop" />
```

```
    </link>

    <!– leyenda 3 –>
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a3" role="onBegin" />
        <bind component="leyenda3" role="start" />
    </link>

    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a3" role="onEnd" />
        <bind component="leyenda3" role="stop" />
    </link>

    </context>
  </body>
</ncl>
```

Codigo del programa principal.

```
  <ncl id="documento_principal" .....>
  <head>
    <importedDocumentBase>
        <importNCL documentURI="menuV1.ncl" alias="menu01"/>
        <importNCL documentURI="programa.ncl" alias="prog"/>
    </importedDocumentBase>

    <regionBase>
        <region id="rgTv">
        ..................
        </region>
    </regionBase>

    <descriptorBase>
        ..................
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="connectorBase.ncl"
        alias="menuConectores"/>
    </connectorBase>
  </head>

  <body>
    <port id="IniPre" component="main-contex"/>
    <context id="main-contex">
    <context id="menu" refer="menu01#menu"/>
    <context id="progra01" refer="prog#progV1"/>
    <port id="InVideo" component="videoIntro"/>
        ..................
    <!–*** Seleccion Item1 ***–>
    <link xconnector="con#onKeySelectionStartStop">
        <bind role="onSelection" component="menu"
        interface="pMenuItem1"/>
        <bind role="stop" component="menu"/>
        <bind role="start" component="progra01"/>
    </link>
```

```
    <link xconnector="con#onEndStart">
        <bind role="onEnd" component="progra01"/>
        <bind role="start" component="menu"/>
    </link>
    </context>
  </body>
</ncl>
```

4.5.9. Adaptación del comportamiento del programa.

4.5.9.1. Alteración de la visibilidad de la leyenda (I).

Este ejemplo permite la posibilidad al usuario que pueda cambiar la visibilidad de la leyenda presionando la tecla amarilla (*Yellow*) del control remoto.

En este ejemplo se guarda el valor de visibilidad de la leyenda. El recurso en NCL que permite guardar valores de propiedades definidas por el programa en un nodo media especial, es del tipo ***application/x-ginga-settings***, también llamado nodo ***settings.*** Se debe definir cada propiedad del nodo ***settings*** de manera análoga a las propiedades de las medias, que van a ser manipuladas, y además estas deben ser declaradas dentro de las mismas.

Para utilizar el valor de visibilidad de la leyenda, se modifican los enlaces que inician la presentación de cada una. Estas pasan a utilizar un conector que testea el valor de la propiedad ***leyenda*** del nodo ***settings.*** En el caso que la leyenda posea un valor relacionado esta es exhibida, de lo contrario el enlace no es activado y la leyenda no es presentada.

En este ejemplo se indica a los usuarios la oportunidad de interactuar a cada momento. Por lo tanto se utiliza dos medias de imagen: una para indicar que el usuario puede desconectar la leyenda (***imgLeyendaDesligar***), para cuando esta se relaciona (que es el valor por defecto) y otro para indicar que el usuario puede conectar la leyenda (***imgLeyendaLigar***), para cuando está desconectada, es decir es el usuario el que tiene la posibilidad de escoger si quiere o no que las leyendas sean mostradas mediante la presión del botón amarillo.

Paso 1: Modificación del archivo principal y definicion de propiedades del nodo media leyenda.

El primer paso es alterar el documento para que este pueda incluir las medias de las imágenes que indican los cambios de visibilidad de la leyenda. También es necesario definir la propiedad ***leyenda*** del nodo ***noSetings*** que inicializa con el valor relacionado ***"ligada"***.

```
<property name="leyenda" value="ligada" />
```

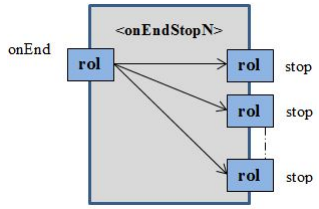
En este ejemplo se utilizó la definición de las regiones y los conectores en documentos separados al del programa principal para mantener una mejor estructura y sea más fácil las modificaciones para ejemplos futuros, la importación de estos documentos es de la misma manera que la que se realizó en ejemplos anteriores con la única diferencia que debe ser realizada dentro de cada elemento correspondiente, como en el caso de la importacion del conectores

Paso 2: Definición del conector que termina varios programas simultáneamente.

Para terminar la presentación de varios nodos simultáneamente cuando otra inicie utilizando el mismo conector que se ha usado hasta el momento se necesitaría definir varios enlaces que realicen esta función, no en tanto NCL ofrece la posibilidad de emplear otros recursos, que posibilitan de manejar un conector que acepta múltiples ligaciones para una misma función, mediante el empleo del atributo ***max= "unbounded"***, que no limita la cantidad de nodos que se pueden finalizar.

Paso 3: Definición del enlace que conecta y termina las imágenes de ligar y desligar la leyenda cuando finaliza el video.

El conector ***onEndStopN*** es utilizado para terminar la exhibición de las medias de imagen para ligar y desligar la leyenda. La tabla 4.12describe el conector ***onEndStopN***:

Nombre:	***onEndStopN***
Condición:	Termina la exhibición del nodo (ligado con la función ***onBegin***).
Acción:	Cierra la presentación de uno o más en nodos (relacionadas con la función ***stop***)
Ilustración:	. 
Código NCL:	<causalConnector id="onEndStopN"> <simpleCondition role="onEnd"/> <simpleAction role="stop" max="unbounded"/> </causalConnector>
Lectura:	Cuando el nodo ligado a la función ***onEnd*** inicia, terminan los nodos ligados a la función ***stop***.

Cuadro 4.12: Definición del conector ***onEndStopN***

Paso 4: Definición del conector que prueba el valor de la propiedad leyenda.

Para probar el valor de leyenda cuando el ancla comienza, es necesario crear un conector que considera el evento de inicio de un nodo (definido por la función ***onBegin***) y permite comparar el valor de la propiedad con un valor que se pasa como parámetro para el conector por el enlace.

La tabla 4.13presenta la definición del conector. La condición es compuesta y esta hecha por el elemento <***compoundCondition***>, ambas condiciones deben ser satisfechas por el enlace al ser activado, según se define por el atributo ***operator*** (con valor **"and"**). Fue definido un elemento <***assessmentStatement***> para hacer la comparación entre la propiedad de un nodo (atributos ***eventType="attribution" attributeType="nodeProperty"***), que deberá ser ligado por un elemento <***bind***> del enlace o función ***testaProp***, y el valor ***oValor*** pasado como parámetro de ligación del enlace (elemento <***bindParam***>), cuando

las dos condiciones son satisfechas y el enlace es activado, el nodo indicado en la función ***start*** es iniciado.

Nombre:	***onBeginPropertyTestStart***
Condición:	Inicia la exhibición de un nodo (función ***onBegin***). Valor de una propiedad igual al palor pasado por el enlace (función ***testaProp***)
Acción:	Exhibe el nodo ligado a la función ***start***
Código NCL:	<causalConnector id="onBeginPropertyTestStart"> <connectorParam name="oValor"/> <compoundCondition operator="and"> <simpleCondition role="onBegin" /> <assessmentStatement comparator="eq"> <attributeAssessment role="testaProp" eventType="attribution" attributeType="nodeProperty"/> <valueAssessment value="$oValor"/> </assessmentStatement> </compoundCondition> <simpleAction role="start" /> </causalConnector>
Lectura:	Cuando la media dela función ***onBegin*** inicia, y las propiedades ligadas a la función ***testaProp*** tienen valores iguales (***eq)*** el parametro ***oValor***, inicia el nodo ligado a la función ***start***.
Observación:	Los enlaces que utilizan este conector deben identificar, como parámetro adicional la ligación con un nodo de origen (<***bindParam***> o <***linkParam***>), el valor que debe compararse con las propiedades, por medio del parámetro ***oValor***.

Cuadro 4.13: Definición del conector ***onBeginPropertyTestStart***

Paso 5: Modificación de los enlaces de inicio de la leyenda.

Los enlaces que utilizaban en el ejemplo anterior el conector ***onBeginStart*** deben ser modificados por el nuevo conector ***onBeginPropertyTestStart***.

Paso 6: Definición del conector que altera el valor de la propiedad leyenda.

Para alterar el valor de la propiedad ***leyenda*** se define un conector que capture cuando la tecla es presionada y realice tres acciones: altera el valor de la propiedad ***leyenda***, que inicia y termina las medias correspondientes al conectarla o desconectarla, manteniendo al usuario informado de cuál será la próxima acción al presionar la tecla ***Yellow***.

La tabla 4.14 presenta la definición del conector ***onKeySelectionSetStartStopDelay***. Cuando el botón ***aTecla*** es presionada durante la exhibición del nodo ligado a la función ***onSelection***, se activa el enlace, realizándose tres acciones:

1. El valor de la propiedad ligado a la función ***set*** es alterado para el valor ***oValor*** pasado como parámetro de la ligación.
2. El nodo ligado a la función ***start*** es iniciado.
3. El nodo ligado a la función ***stop*** es terminado.

Finalmente para hacer el conector independiente de la implementación del formateador, se necesita introducir un retardo en la definición del valor de la propiedad leyenda. Con esto se evita por ejemplo que el enlace vinculado a la imagen ***imgLeyendaDesligar*** sea activado, alterando la ***leyenda*** y las imágenes, en el instante que el enlace vinculado a la imagen ***imgLeyendaLigar*** es activado, cambia la ***leyenda*** y las imágenes al estado anterior, produciendo comportamientos no deseados y evitan que el usuario cambie la visibilidad de la leyenda.

Paso 7: Definición de los enlaces que cambian el valor de la propiedad leyenda conforme es presionado el botón amarillo del control remoto.

El último paso consiste en crear los enlaces que utilizan el conector ***onKeySelectionSetStartStopDelay*** para que mediante la presión del botón amarillo del control remoto cambie el valor de la propiedad leyenda y alternar la media de la imagen que identifica el efecto de la interacción del usuario.

El retardo se define como un parámetro de enlace (***linkParam***), es decir el valor toma un valor default para el parámetro ***oRetardo*** en ese enlace, para todas las acciones definidas con ***delay***. Si es necesario, el valor puede ser redefinido por un parámetro de conexión (***bindParam***).

Codigo del ejemplo 14.

Codigo del las regiones.

```
<ncl id="regiones" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
<head>
    <regionBase>
        <region id="regPantallaIn" zIndex="1" />
        <region id="regLeyenda" left="25%" width="50%" top="90%"
        zIndex="2" />
        <region id="regBoton_A" right="5%" bottom="5%" width="10%"
        height="15%" zIndex="3" />
    </regionBase>
  </head>
</ncl>
```

Codigo del los descriptores.

```
<ncl id="descriptores" .....>
<head>
    <descriptorBase>
        <descriptor id="desLeyenda" region="regLeyenda">
            <descriptorParam name="border" value="none" />
        </descriptor>
        <descriptor id="desPantallaIn" region="regPantallaIn"
        explicitDur="40s"/>
        <descriptor id="desBoton_A" region="reg#regBoton_A" />
    </descriptorBase>
  </head>
</ncl>
```

Codigo del programa principal.

```
<ncl id="programa" .....>
<head>
```

```
    <regionBase>
        <importBase documentURI="regiones.ncl" alias="reg"/>
    </regionBase>

    <descriptorBase>
        <importBase documentURI="descriptores.ncl" alias="des"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="InVi" component="progV2"/>
    <context id="progV2">
    <port id="pVideo1" component="video1"/>
    <media id="video1" src="Video/video.mpg"
    descriptor="desPantallaIn">
        <area id="a1" begin="2s" end="5s" />
        <area id="a2" begin="7s" end="9s" />
        <area id="a3" begin="11s" end="13s" />
    </media>

    <media id="leyenda1" src="Texto/leyenda1.htm"
        descriptor="des#desLeyenda" />
    <media id="leyenda2" src="Texto/leyenda2.htm"
        descriptor="des#desLeyenda" />
    <media id="leyenda3" src="Texto/leyenda3.htm"
        descriptor="des#desLeyenda" />

    <media id="imgLeyendaLigar" src="Imagenes/B-amarillo.png"
    descriptor="des#desBoton_A" />
    <media id="imgLeyendaDesligar" src="Imagenes/Ksco-FA.png"
    descriptor="des#desBoton_A" />

    <!- Nodo midia especial para guardar la leyenda ->
    <media id="noSettings" type="application/x-ginga-settings">
        <property name="leyenda" value="ligada" />
    </media>

    <link xconnector="con#onBeginStart">
        <bind component="video1" role="onBegin" />
        <bind component="imgLeyendaDesligar" role="start" />
    </link>

    <!- leyenda 1 ->
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a1" role="onBegin" />
        <bind component="leyenda1" role="start" />
    </link>
    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a1" role="onEnd" />
        <bind component="leyenda1" role="stop" />
    </link>
```

```
    <!– leyenda 2 –>
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a2" role="onBegin" />
        <bind component="leyenda2" role="start" />
    </link>
    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a2" role="onEnd" />
        <bind component="leyenda2" role="stop" />
    </link>

    <!– leyenda 3 –>
    <link xconnector="con#onBeginStart">
        <bind component="video1" interface="a3" role="onBegin" />
        <bind component="leyenda3" role="start" />
    </link>
    <link xconnector="con#onEndStop">
        <bind component="video1" interface="a3" role="onEnd" />
        <bind component="leyenda3" role="stop" />
    </link>

    <!– Modificacion de la leyenda –>
    <link xconnector="con#onKeySelectionSetStartStopDelay">
        <linkParam name="oRetardo" value="0.5s" />
        <bind component="imgLeyendaDesligar" role="onSelection">
            <bindParam name="aTecla" value="YELLOW" />
        </bind>
        <bind component="noSettings" interface="leyenda" role="set">
            <bindParam name="oValor" value="desligada" />
        </bind>
        <bind component="imgLeyendaLigar" role="start" />
        <bind component="imgLeyendaDesligar" role="stop" />
    </link>

    <!– Modificacion de la leyenda –>
    <link xconnector="con#onKeySelectionSetStartStopDelay">
        <linkParam name="oRetardo" value="0.5s" />
        <bind component="imgLeyendaDesligar" role="onSelection">
            <bindParam name="aTecla" value="YELLOW" />
        </bind>
        <bind component="noSettings" interface="leyenda" role="set">
            <bindParam name="oValor" value="desligada" />
        </bind>
        <bind component="imgLeyendaDesligar" role="start" />
        <bind component="imgLeyendaLigar" role="stop" />
    </link>
    </context>
  </body>
</ncl>
```

4.5.9.2. Alteración de la visibilidad de la leyenda (II).

Este ejemplo posee una semejante al ejemplo anterior, pero en este se utiliza un recurso de NCL llamado <*descriptorSwitch*> (selección del descriptor) y ***rule*** (regla). En lugar de desidir el iniciar o no la leyenda de cada ancla el programa siempre inicia, la diferencia está en que las reglas <*descriptorSwitch*> *dLeyenda* deciden por que descriptor de leyenda debe empezar: *dLegendaLigada* (visible=true, o default) o *dLegendaDesligada* (visible=false).

Paso 1: Definición de las reglas que evalúan el valor de propiedad de la leyenda.
Para hacer posible determinar el valor de la propiedad leyenda, son definidas las siguientes reglas, de la sección <***ruleBase***> en <***head***>, se debe tener en cuenta que las reglas deben ser declaradas antes de las regiones y descriptores.

Paso 2: Configuración del descriptorSwitch y las reglas de conexión para el descriptor adecuado.
En lugar de definir solamente un descriptor ***dLeyenda*** para las leyendas, este ejemplo define un ***descriptorSwitch*** con dos reglas, una para las leyendas visibles y otra para las invisibles, cuya selección será realizada en base a la validación de reglas de conexión (***bindRule***) del ***descriptorSwitch***.

Las reglas se validan en el orden en que fueron declaradas. Si la regla ***rLeyendaLigada*** es comprobada como verdadera, el descriptor utilizado para la presentación del nodo será ***dLeyendaLigada***. En el caso que la regla ***rLeyendaDesligada*** es comprobada como verdadera, el descriptor utilizado para la presentación del nodo será ***dLeyendaDesligada***.

Paso 3: Modificación del descriptor utilizado por las medias de leyenda.
Como el nombre asignado al ***descriptorSwitch fue dLeyenda***, en este ejemplo no es necesario modificar el descriptor que se utilizó para medias de leyenda visible.

Para iniciar cada una de las medias de leyenda, las reglas de ***descriptorSwitch dLeyenda*** son evaluadas y el programa escoge el descriptor correspondiente para iniciar la presentación de la media: ***dLegendaLigada*** o ***dLegendaDesligada***.

Paso 4: Modificación de los enlaces que inician las leyendas.
Como la decisión de mostrar o no una leyenda pasa por el descriptorSwitch, los enlaces ahora vuelven a utilizar el conector *onBeginStart.*

Codigo del ejemplo 15.

Codigo del programa principal.

```
<ncl id="programa" .....>
  <head>
    <ruleBase>
        <rule id="rLeyendaLigada" var="leyenda" comparator="eq"
        value="ligada" />
        <rule id="rLeyendaDesligada" var="leyenda" comparator="eq"
        value="desligada" />
    </ruleBase>
    <regionBase>
```

```
<importBase documentURI="regiones.ncl" alias="reg"/>
    </regionBase>
    <descriptorBase>
        <descriptor id="desPantallaIn" region="reg#regPantallaIn"
        explicitDur="40s"/>
        <descriptor id="desBoton_A" region="reg#regBoton_A" />

        <descriptorSwitch id="desLeyenda">
            <bindRule rule="rLeyendaLigada"
            constituent="dLeyendaLigada" />
            <bindRule rule="rLeyendaDesligada"
            constituent="dLeyendaDesligada" />
            <descriptor id="dLeyendaLigada"
            region="reg#regLeyenda">
              <descriptorParam name="border" value="none" />
            </descriptor>
            <descriptor id="dLeyendaDesligada"
            region="reg#regLeyenda">
              <descriptorParam name="visible" value="false" />
            </descriptor>
        </descriptorSwitch>
    </descriptorBase>
  </head>

  <body>
    <port id="InVi" component="progV2"/>
    <context id="progV2">
    <port id="pVideo1" component="video1"/>
    <media id="video1" src="Video/video.mpg"
    descriptor="desPantallaIn">
        <area id="a1" begin="2s" end="5s" />
        <area id="a2" begin="7s" end="9s" />
        <area id="a3" begin="11s" end="13s" />
    </media>

    <media id="leyenda1" src="Texto/leyenda1.htm"
        descriptor="desLeyenda" />
    <media id="leyenda2" src="Texto/leyenda2.htm"
        descriptor="desLeyenda" />
    <media id="leyenda3" src="Texto/leyenda3.htm"
        descriptor="desLeyenda" />

    <media id="imgLeyendaLigar" src="Imagenes/B-amarillo.png"
    descriptor="des#desBoton_A" />
    <media id="imgLeyendaDesligar" src="Imagenes/Ksco-FA.png"
    descriptor="des#desBoton_A" />

    <!- Nodo midia especial para guardar la leyenda ->
    <media id="noSettings" type="application/x-ginga-settings">
        <property name="leyenda" value="ligada" />
    </media>

    <link xconnector="con#onBeginStart">
        <bind component="video1" role="onBegin" />
```

```
    <bind component="imgLeyendaDesligar" role="start" />
</link>

<!– leyenda 1 –>
<link xconnector="con#onBeginPropertyTestStart">
    <bind component="video1" interface="a1" role="onBegin" />
    <bind component="noSettings" interface="leyenda"
    role="testaProp">
        <bindParam name="oValor" value="ligada" />
    </bind>
    <bind component="leyenda1" role="start" />
</link>

<!– leyenda 2 –>
<link xconnector="con#onBeginPropertyTestStart">
    <bind component="video1" interface="a1" role="onBegin" />
    <bind component="noSettings" interface="leyenda"
    role="testaProp">
        <bindParam name="oValor" value="ligada" />
    </bind>
    <bind component="leyenda2" role="start" />
</link>

<!– leyenda 3 –>
<link xconnector="con#onBeginPropertyTestStart">
    <bind component="video1" interface="a1" role="onBegin" />
    <bind component="noSettings" interface="leyenda"
    role="testaProp">
        <bindParam name="oValor" value="ligada" />
    </bind>
    <bind component="leyenda3" role="start" />
</link>

<!– Modificacion de la leyenda –>
<link xconnector="con#onKeySelectionSetStartStopDelay">
    <linkParam name="oRetardo" value="0.5s" />
    <bind component="imgLeyendaDesligar" role="onSelection">
        <bindParam name="aTecla" value="YELLOW" />
    </bind>
    <bind component="noSettings" interface="leyenda" role="set">
        <bindParam name="oValor" value="desligada" />
    </bind>
    <bind component="imgLeyendaLigar" role="start" />
    <bind component="imgLeyendaDesligar" role="stop" />
</link>

<!– Modificacion de la leyenda –>
<link xconnector="con#onKeySelectionSetStartStopDelay">
    <linkParam name="oRetardo" value="0.5s" />
    <bind component="imgLeyendaDesligar" role="onSelection">
        <bindParam name="aTecla" value="YELLOW" />
    </bind>
    <bind component="noSettings" interface="leyenda" role="set">
```

```
            <bindParam name="oValor" value="desligada" />
        </bind>
        <bind component="imgLeyendaDesligar" role="start" />
        <bind component="imgLeyendaLigar" role="stop" />
    </link>
    </context>
  </body>
</ncl>
```

4.6. Interacción NCL-Lua.

Un script Lua maneja la misma abstracción para objetos media utilizados por imagines, videos u otro tipo de medias, algo que apenas lo hace NCL ya que este se refiere al objeto media y no a su contenido. El lenguaje Lua posee adaptación para funcionar contenido dentro del lenguaje NCL, pero debe ser escrito en un documento con extensión .lua separado del documento mismo, que apenas lo referencia como cualquier otro nodo media.

La principal característica de NCLua está en que su ciclo de vida es controlado por el documento NCL que lo referencia, y es activado en el momento en que el programa Lua es inicializado. La función puede recibir un valor de tiempo como un parámetro opcional que puede ser usado para especificar el momento exacto que el documento debe ser ejecutado.

4.6.1. Modulos NCLua.

Las bibliotecas disponibles para NCLua están divididas en cuatro módulos esenciales donde cada uno expresa un conjunto de funciones, estos módulos ya fueron indicados en el capítulo 3 y en esta sección se describen de forma completa.

4.6.1.1. Modulo event.

El middleware Ginga posee un modelo propio de ejecución y comunicación de objetos imperativos incrustados en documentos NCL. En el caso de objetos NCLua, los mecanismos de integración con un documento NCL se realizan a través del paradigma de la programación orientada a eventos.

El paradigma, es realizado por la difusión y recepción de eventos que se efectúan para comunicarse con el documento NCL y toda la interacción con entidades externas a la aplicación, tales como el canal de interactividad, control remoto y temporizadores. El módulo ***event*** de NCLua es utilizado para este propósito y su entendimiento es esencial para desarrollar cualquier aplicación que utilice objetos NCLua, este modulo sigue el diagrama de la Figura 4.15.

Para comunicarse con un NCLua, una entidad externa debe insertar un evento en la cola, que a continuación es redireccionado a las funciones tratadoras de eventos, definidas por el programador de scripts NCLua. Mientras cada tratador procesa un evento (uno a la vez), ningún otro evento de la cola es tratado. Lua recibe todos los eventos que ocurren en la aplicación NCL, si la media tiene foco (Figura 4.16).

Un evento se capta mediante una función manejadora de eventos. El manejador de eventos tiene como parámetro un evento. Ejemplo:

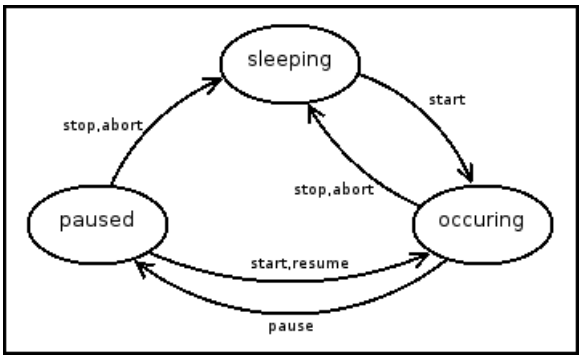


Figura 4.15: Diagrama de estado
Fuente: Referencia personal.

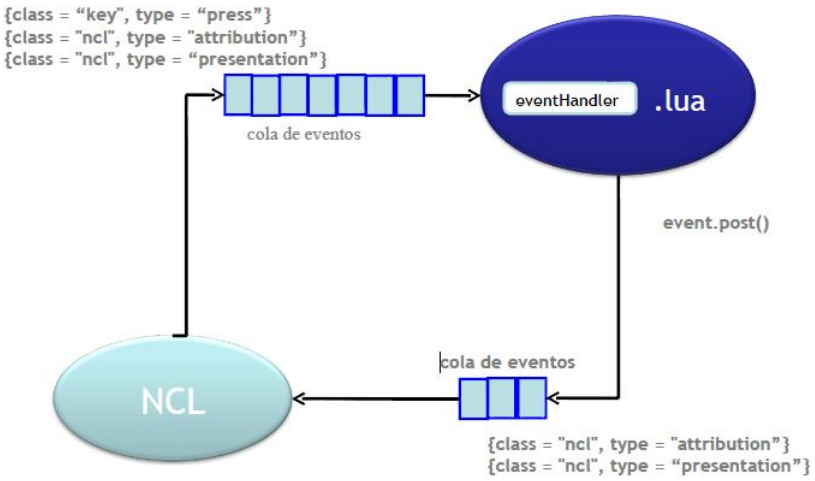


Figura 4.16: Programación orientada a eventos
Fuente: Referencia personal.

```
function handler(evt)
    ...
end
```

Además debemos indicar que el manejador de eventos reciba todos los eventos de la aplicación.

```
event.register(handler)
```

Para ser informado cuando eventos externos son recibidos, un NCLua debe registrar al menos una función de tratamiento en su cuerpo a través de una llamada a la función event.register. (El nombre de la función a ser registrada es cualquiera). El código de un NCLua sigue una estructura común a todos los scripts, tales como los siguientes:

```
...       – código de inicialización.
function handler (evt)
    ...       – codigo para tratar los eventos
end
event.register(handler)       – registro del tratador
```

El código de inicialización, la definición de los tratadores y su registro son ejecutados antes de que el documento NCL (o cualquier entidad externa a script)

señale cualquier evento a NCLua, incluyendo el inicio de la presentación del objeto. Después de que el proceso de carga del script, es efectuado por el sistema, solo el código del tratador es llamado cada vez que ocurre un evento externo.

Un NCLua también puede enviar eventos para comunicarse con la aplicación NCL, y por ejemplo, podria enviar datos por el canal de interactividad o señalar su estado a un documento NCL., para esto utiliza la función *event.post()* mostrado a continuación (Figura 4.16):

```
event.post (dst; evt)
```

Como NCLua (a través de sus manejadores) debe correr rápidamente, nunca la función de envío de eventos espera el regreso de un valor. Si el destino necesita devolver una información a un NCLua, debe hacerlo mediante el envío de un nuevo evento. En el siguiente ejemplo, el NCLua señala a un documento su fin natural.

```
event.post {
    class='ncl',
    type='presentation',
    action='stop',
}
```

Los eventos son tablas Lua simples siendo el campo ***class*** el responsable de identificar la clase de evento y separarlos por categorías. La clase identifica no solamente el origen de eventos pasados a los tratadores, sino también su destino, si el evento es generado y publicado por un script NCLua.

Las siguientes clases de eventos están definidas:

- ***Classe ncl:*** Utilizada en la comunicación entre un NCLua y el documento NCL que contiene el objeto de medios.
- ***Classe user:*** A través de esta clase, aplicaciones pueden extender su funcionalidad creando sus propios eventos. Como los eventos de esta clase son para uso interno, no tiene sentido publicar sus eventos con el destino igual a "out".
- ***Classe key:*** Representa la presión de teclas del control remoto por el usuario. Para esta clase no tiene sentido que el NCLua genere eventos, ya que el control remoto es un dispositivo únicamente de entrada.
- ***Classe tcp:***Permite acceso al canal de interactividad por medio del protocolo TCP.
- ***Classe sms:***Usada para envio y recibimiento de mensajes SMS en dispositivos móviles.
- ***Classe edit:*** Permite que los comandos de edición en vivo sean activados a partir de scripts NCLua.
- ***Classe si:*** Proporciona acceso a un conjunto de informaciones multiplexadas en un flujo de transporte y transmitidas periódicamente por difusión.

Como se puede observar hay eventos sólo de entrada, sólo de salida, y eventos que se utilizan en ambas direcciones.

Classe *ncl*:

Un documento NCLua se comunica con uno en el cual esta insertado atravez de esta clase de eventos. En un documento NCL, las relaciones entre los nodos media son descritas atravez de enlaces que relacionan condiciones de anclaje. Un documento NCLua interactúa con el documento únicamente por medio de sus enlaces que están asociados en sus objetos media. Por lo tanto no es posible que un NCLua interfiera directamente en el comportamiento de otras medias presentadas. Los enlaces que accionan el NCLua, con sus condiciones satisfechas hacen que el NCLua reciba un evento describiendo la acción a ser tomada. Por ejemplo el enlace que aparece a continuación:

```
<link xconnector="onBeginStart">
    <bind role="onBegin" component="videoId"/>
    <bind role="start" component="luaId"/>
</link>
```

Cuando el video inicia el NCLua recibe el evento:
{ *class='ncl', type='presentation', action='start'* }

Este evento será recibido por la función registrada durante la inicialización del *script*. Para los enlaces cuya condición depende de un NCLua, la acción será desencadenada cuando este señalice el evento que realiza la condición esperada. Por ejemplo el enlace que aparece a continuación:

```
<link xconnector="onBeginStart">
    <bind role="onEnd" component="luaId"/>
    <bind role="start" component="imageId"/>
</link>
```

Tan pronto como el NCLua publicar el evento:
event.post { class='ncl', type='presentation', action='stop' }

El enlace mostrara la imagen que forma parte del enlace. Existen dos tipos de eventos para la clase ncl soportados por NCLua: *presentación* y *atribución*. El tipo es identificado en el campo *type* del evento y podrá asumir, por tanto, solamente los valores '*presentation*' o '*attribution*'.

Tipo '*presentation*': Los eventos de presentación controlan la exhibición del nodo. Además pueden estar asociados con áreas (anclas de presentación) que definen a un nodo como un todo. Las áreas son especificadas por el campo *area* y equivalen a un nodo completo cuando no son especificadas se asume un valor '*nil*'.

El campo ***action*** indica la acción a ser realizada o señalizada por NCLua, dependiendo si este está recibiendo o generando un evento. En resumen, un evento de presentación tiene la siguiente estructura:

- ***class:*** 'ncl'
- ***type:*** 'presentation'
- ***area:*** [string] Nombre del ancla (label) asociada al evento.
- ***action:*** [string] Puede asumir los siguientes
- ***valores:*** 'start', 'stop', 'abort', 'pause' e 'resume'.

Tipo *'attribution'*: Eventos de atribucion controlan las propiedades de NCLua. El campo ***property*** del evento contiene el nombre de las propiedades que están afectadas. Los eventos de asignación son bastante similares a la de presentación, una vez que se rigen por el mismo tipo de máquina de estado. Por lo tanto, la acción ***start*** en un evento de atribución corresponde al ***role="set"*** en un enlace NCL. El campo ***value*** se declara como el valor a ser atribuido y siempre es una *string*, una vez que proviene de un atributo XML. La acción para que se inicie en un evento de asignación corresponde al ***role= "set"*** en un enlace NCL.

Las propiedades de NCLua no tienen ninguna relación directa con las variables declaradas en el *script*. Una NCLua que intenta cambiar el valor de una propiedad debe publicar un evento para este propósito. Las propiedades de los nodos están controladas por el propio documento NCL. Por ejemplo.

event.post { class = 'ncl', type = 'attribution', property = 'myProp', action = 'start', value = '10', }

Un evento de atribución posee la siguiente estructura.

- ***class:*** 'ncl'
- ***type:*** 'attribution'
- ***property:*** [string] Nombre de la propiedad (name) asociada al evento.
- ***action:*** [string] Puede asumir los seguinte valores: 'start', 'stop', 'abort', 'pause' e 'resume'.
- ***value:*** [string] Nuevo valor a ser atribuído a la propiedad.

Classe '*key*':

Representa la presión de teclas del control remoto por el usuario. Para esta clase no tiene sentido que el NCLua genere eventos, ya que el control remoto es un dispositivo únicamente de entrada. Ejemplo:
{ class='key', type='press', key='0' }

Eventos da classe key posee la siguiente estructura:

- ***class:*** 'key'
- ***type:*** [string] Puede asumir 'press' o 'release'.
- ***key:*** [string] Valor de la tecla en cuestión.

Classe '*user*':

Las aplicaciones pueden extender su funcionalidad creando sus propios eventos desde esta clase. Los campos de la tabla que representa el evento se define (además del, el campo *clase*). Como los eventos de esta clase son para uso interno, no tiene sentido publicar sus eventos con el destino igual a *"out"*. Ejemplo:
{ class='user', data='mydata' }

Classe '*tcp*':

El uso del canal de interactividad es llevada a cabo por medio de esta clase de eventos. Con el fin de enviar y recibir datos, la conexión debe ser preestablecida, registrando un evento como se indica a continuación.

{event.post
class = 'tcp',
type = 'connect', host = <addr>,
port = <number>,
[timeout = <number>,]
}

El resultado de la conexión es un tratador de eventos pre registrado.El evento de regreso tiene la siguiente estructura:

evt = {
class = 'tcp',
type = 'connect',
host = <addr>,
port = <number>,
connection = identifier,
error = <err_msg>,
}

Los campos *error* y *connection* son mutuamente exclusivos. Cuando se trata de un problema de conexión, un mensaje de error es devuelto en el campo *error*. Cuando la conexión se realiza correctamente, un identificador único para la conexión es devuelve en el campo *connection*. Una NCLua envía datos a través del canal de retorno publicando eventos de la siguiente manera: {event.post
class = 'tcp',
type = 'data',
connection = <identifier>,
value = <string>,
[timeout = number,]
}

De manera similar, un NCLua recibe datos del canal de retorno en eventos de la siguiente forma.

evt = {
class = 'tcp',
type = 'data',
value = <string>,
connection = <identifier>,
error = <err_msg>,
}

Una vez más, los campos *error* y *connection* son mutuamente excluyentes. Cuando se trata de un problema de conexión, un mensaje de error es devuelto en el campo de *error*. Cuando la conexión se realiza correctamente, un identificador único para la conexión se devuelve en el campo *connection*. Para cerrar la conexión, el siguiente suceso se debería publicar:

{event.post
class = 'tcp',
type = 'disconnect',
connection = <identifier>

}

Todas las funciones que pueden ser utilizadas por el modulo ***even*** pueden ser consultadas en [16]

4.6.1.2. Módulo Canvas

UNA NCLua tiene la posibilidad de realizar las operaciones gráficas durante la presentación de una aplicacion, tales como el dibujo de líneas, círculos, imágenes, etc. Esto se realiza por medio de la utilización del módulo canvas que ofrece un API para ser utilizado por NCLua.

Cuando un objeto NCLua es iniciado, la región del elemento <***media***> correspondiente (del tipo ***application/x-ginga-NCLua***) funciona como variable global ***canvas*** para el *script* Lua. Si la variable <***media***> no posee ninguna región especificada (propiedad left, right, top y bottom) este valor de canvas es establecido como "***nil***". Si la región está asociado con, por ejemplo:

<region id="luaRegion" width="300" height="100" top="200" left="20"/>

La variable canvas de NCLua correspondiente estará asociado a la región luaRegion de tamaño 300x100 se localiza en la posición (20,200).

Además de los primitivos gráficos que ya se mencionaron, también es posible instanciar nuevos canvas, a través de un constructor canvas:new(...), y mediante esta manera se representan otros objetos gráficos (layers) que después pueden ser compuestos.

La variable canvas de NCLua correspondiente estará asociado a la región luaRegion de tamaño 300x100 se localiza en la posición (20,200).

Además de los primitivos gráficos que ya se mencionaron, también es posible instanciar nuevos canvas, a través de un constructor canvas:new(...), y mediante esta manera se representan otros objetos gráficos (layers) que después pueden ser compuestos.

Un objeto canvas almacena en su estado atributos bajo los cuales las primitivas gráficas funcionan, por ejemplo, si el atributo de color es azul, una llamada al *canvas:drawLine(...)* dibujara una línea azul sobre canvas. Los atributos se accede a través de los métodos de prefijo ***attr*** y sufijo del nombre del atributo (por ejemplo ***attrcolor***), los cuales sirven tanto para la lectura y la escritura (***getter*** y ***setter***).

El primer parámetro de todos los métodos del módulo siempre es ***self***, es decir, una referencia al canvas en cuestión. Por lo tanto, se recomienda utilizar el método llamado Lua utilizando el operador : (color operator) como en:

myCanvas:drawRect('fill', 10, 10, 100, 100)

Las coordenadas pasadas a los métodos son siempre relativas al punto más a la izquierda y a la parte superior de canvas (0,0), como es común entre sistemas gráficos.

Atributos de Canvas:

- ***canvas:attrSize():*** retorna las dimensiones del canvas.
  - Ejemplo:
    - local w,h = canvas:attrSize()

- ***canvas:attrColor(color):*** función que modifica el color del canvas.
  - Ejemplo:
    - canvas:attrColor('white'),
    - canvas:attrColor('red'),
- ***canvas:attrFont(fontFamily,fontSize,fontWeight):*** función que modifica atributos de la fuente.
  - Ejemplo:
    - canvas:attrFont('vera', 24, 'bold'),
    - canvas:attrFont('dejaVuSerif', 20, 'normal')

Funciones:

- ***canvas:new(image_path)*** retorna un nuevo canvas cuyo contenido es la imagen pasada como parámetro.
  - Ejemplo:
    - local img = canvas:new('imagen.png')
  - Las funciones ya vistas pueden ser aplicadas sobre el nuevo canvas.
  - Ejemplo:
    - local w,h = img:attrSize()
- ***canvas:compose(x,y,src):*** función que compone el canvas principal con el canvas especificado en src en la posición x,y.
  - Ejemplo:
    - canvas:compose(0,0,canvas:new('imagen.png'))
- ***canvas:drawLine(x1,y1,x2,y2):*** función que dibuja una línea con sus extremos en (x1,y1) y (x2,y2)
  - Ejemplo:
    - canvas:drawLine(10, 10, 100, 100)
- ***canvas:drawRect(mode,x,y,width,height):*** función que dibuja un rectángulo. El parámetro mode puede tomar los valores 'frame' o 'fill'.
  - Ejemplo:
    - canvas:drawRect('fill', 10, 10, 100, 100)
    - canvas:drawRect('frame', 50, 50, 200, 200)
- ***canvas:drawText(x,y,'texto'):*** función que dibuja un 'texto' en la posición x,y
  - Ejemplo:
    - canvas:drawText(10, 10, 'Hola Mundo!')
    - canvas:drawText(5, 40, 'Ejemplo Lua')
- canvas:flush() función para actualizar la superficie del canvas. •

- Ejemplo:
  - canvas:flush()

Todas las funciones que pueden ser utilizadas por el modulo ***canvas*** pueden ser consultadas en [16]

4.6.1.3. Modulo settings

Las propiedades de un nodo settings solo pueden ser modificadas por medio de los enlaces de NCL. No es posible atribuir valores a campos que representan variables en los nodos settings.

La tabla settings divide sus grupos en varias subtablas correspondiente a cada grupo del nodo ***application/x-ginga-settings***. Por ejemplo en un objeto NCLua la variable del nodo settings ***"system.CPU"*** es referida como ***settings.system.CPU***.

lang = settings.system.language
age = settings.user.age
val = settings.default.selBorderColor settings.user.
age = 18 –> ERRO!

4.6.1.4. Módulo persistent

Aplicaciones NCLua permiten que datos sean guardados en una zona de acceso restringida del middleware y recuperados entre ejecución. Al exhibir Lua define una área reservada, inaccesible para objetos NCL no procedurales. No existe ninguna variable predefinida o reservada a estos objetos procedurales pudiendo atribuir valores a estas variables directamente. El uso de tablas persistentes es semejante a la utilización de tablas settings, excepto por el caso de que el código procedural pueda mudar los valores de los campos.

persistent.service.total = 10
color = persistent.shared.color

4.7. Creación del primer proyecto NCLua en Eclipse.

Para crear el primer proyecto NCLua se ejecuta la plataforma Eclipse, se selecciona en la barra de herramientas la opción File y se procede a escoger *New -> Lua Project* como se muestra en la imagen.

En la siguiente ventana asignamos un nombre al proyecto nuevo Lua (*Lua New Project Wizard*), y damos click en el botón de *Finish* y el proyecto termina su creación.

Una vez creado el nuevo proyecto Lua se debe proceder a crear el documento para la edición del código Lua. Para la creación del documento se da un clic derecho sobre la carpeta que contiene el proyecto general Java, y escogemos la opción de *Other* (o ctrl+N), debido a que la opción para crear el documento Lua no se presenta de forma directa (ver Figura 4.19).

En la imagen que se muestra se selecciona la opción *New Lua File* y se procede a continuar con la generacion del mismo. En la siguiente ventana de creación que se despliega al escoger la opción del nuevo archivo se asigna el nombre para el mismo y tiene una extension .lua (ver Figura 4.20).

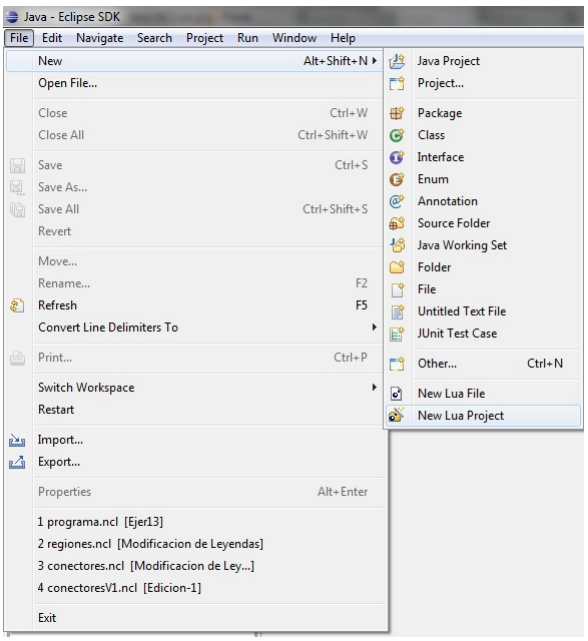


Figura 4.17: Creacion del primer proyecto NCLua
Fuente: Captura de la pantalla del software Eclipse-NCLua.

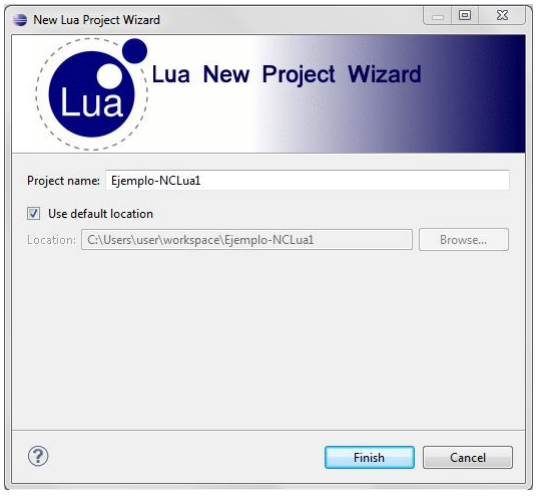


Figura 4.18: Proyecto Java General
Fuente: Captura de la pantalla del software Eclipse-NCLua.

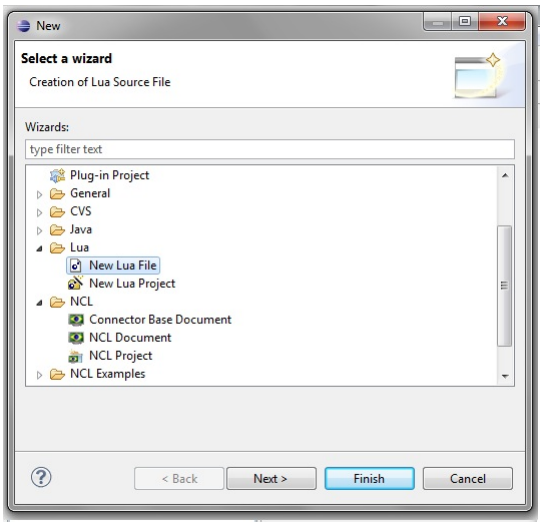


Figura 4.19: Primer ejemplo de creacion de un proyecto Ginga-NCL
Fuente: Captura de la pantalla del software Eclipse-NCLua.

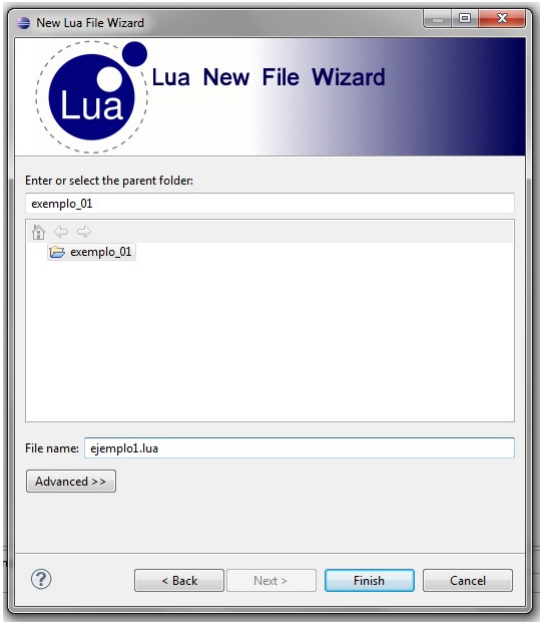


Figura 4.20: Ventana de asignacion del nombre del documento NCL
Fuente: Captura de la pantalla del software Eclipse-NCLua.

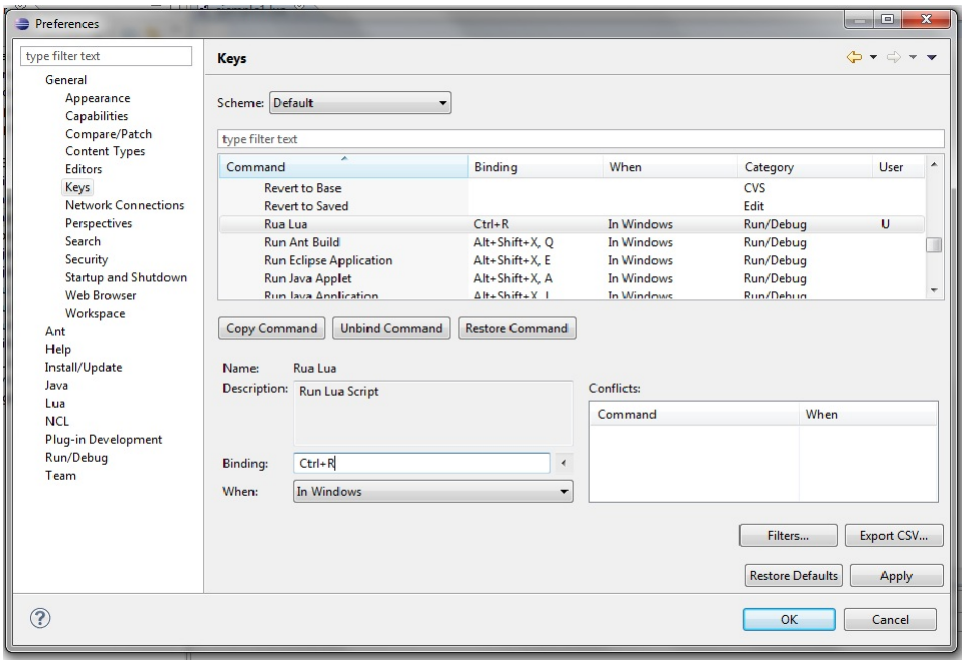


Figura 4.21: Nodo contenido dentro de la carpeta media.
Fuente: Captura de la pantalla del software Eclipse-NCLua.

Para compilar un script Lua damos click derecho sobre el archivo Lua y seleccionamos la opción *Run As-> Lua Aplication.* Para evitar este método de compilación se puede configurar un atajo, en *Window -> Preferences -> General -> Keys*, para asignar a esta tarea una combinación *Ctrl+R* para el script Lua (Rua Lua) como se muestra en la Figura 4.21.

4.8. Ejemplos de programacion en Ginga-NCL.

4.8.1. Ciclo de Vida de Objetos NCLua:

Al iniciar el documento NCL, tres nodos NCLua son iniciados:

- El primero es un script vacio (sin ninguna línea de codigo). No hay arranque y así no hay ningún controlador de eventos.
- El segundo registro es un controlador de eventos que genera un final natural para conseguir un comienzo.
- El tercer registro es un controlador de eventos que crea un temporizador de 3s, que a su vez genera un final natural a punto de expirar.

El documento NCL también crear los botones que indican el estado de cada NCLua, asociado con cada uno:

- Un enlace que pone en marcha un botón de color verde cuando se inicia NCLua.
- Un enlace que inicia un botón rojo y detiene la presentación del botón verde cuando el correspondiente NCLua ha terminado.

El resultado visual que se obtiene del funcionamiento de los tres nodos inicia con la presentación del NCLua del botón verde que se exhibe indefinidamente, el

segundo resultado es que se termina la presentación inmediatamente y empieza la exhibición del botón rojo correspondiente, y el ultimo resultado es que se exhibe el botón verde y después de 3 segundos se vuelve rojo.

Paso 1: Creación del documento NCL para llamar los nodos NCLua..

El documento NCL es responsable de definir e iniciar los tres objetos NCLua, así como la "adhesión lógica" entre cada NCLua y sus botones correspondientes. En este ejemplo el NCL no acciona el final de cualquier objeto NCLua. Esta función es responsabilidad de cada ***script*** NCLua, y es definido dentro de cada uno de los documentos.

El primero NCLua es ligado a otros por medio de un enlace "***onBeginStart***". Como la primera NCLua está ligada al puerto de entrada del documento, los tres objetos inician con la aplicación. Cada NCLua se liga por medio de un enlace "***onBeginStart***" con su respectivo botón verde, por lo que se muestran con el comienzo de cada NCLua. Para ocultar el botón verde y exibir el rojo, cada NCLua también utiliza un enlace "***onEndStopStart***" con sus botones

Paso 2: Creación del primer documento NCLua.

El primer NCLua es un ***script*** vacio (sin ninguna línea de código). En particular, como no tiene un controlador de eventos, nunca señala la finalización del documento NCL. El efecto visual y la exposición es permanente del primer botón verde.

Paso 3: Creación del segundo documento NCLua.

El segundo NCLua registra un controlador de eventos que genera su fin natural para recibir un "***star***" del documento NCL. Visualmente, inmediatamente después de la exhibición del segundo botón verde, es exhibido el botón rojo correspondiente (el botón verde no puede ser visto). Tan pronto el suceso que indica su inicio es recibido, el NCLua pone un evento para señalar su fin natural

Paso 4: Creación del tercer documento NCLua.

El tercero NCLua registra un controlador de eventos que crea un temporizador de tres segundos que, a su vez, genera su fin natural o expiración. Como un efecto visual, tenemos la exhibición del tercer botón verde y, después de tres segundos, cambiando a rojo.

El temporizador de tres segundos (3000 milisegundos) se crea por lo que el evento de inicio es recibido, pasando por la función que debe ser ejecutada cuando el temporizador expire. Esta función llama un evento similar al del segundo NCLua para señalar su fin natural.

Mientras se ejecuta de forma indefinida, el primero NCLua no consume recursos y podría responder a eventos (a pesar de no hacerlo en este caso por no poseer un controlador para este fin). Lo mismo ocurre con el tercero NCLua que espera los tres segundos para terminar.

Codigo del ejemplo NCLua 1.

Codigo del programa principal.

```
<ncl id="nclClicks" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
```

```
    <regionBase>
        <region width="20 %" height="20 %" left="10 %"
        top="40 %" id="rg1"/>
        <region width="20 %" height="20 %" left="40 %"
        top="40 %" id="rg2"/>
        <region width="20 %" height="20 %" left="70 %"
        top="40 %" id="rg3"/>
    </regionBase>

    <descriptorBase>
        <descriptor id="ds1" region="rg1"/>
        <descriptor id="ds2" region="rg2"/>
        <descriptor id="ds3" region="rg3"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="entryPoint" component="lua1"/>

    <!- MEDIAS ->
    <media id="lua1" src="1.lua"/>
    <media id="lua2" src="2.lua"/>
    <media id="lua3" src="3.lua"/>

    <media id="bt1_green" src="buttons/1_green.png" descriptor="ds1"/>
    <media id="bt1_red" src="buttons/1_red.png" descriptor="ds1"/>
    <media id="bt2_green" src="buttons/2_green.png" descriptor="ds2"/>
    <media id="bt2_red" src="buttons/2_red.png" descriptor="ds2"/>
    <media id="bt3_green" src="buttons/3_green.png" descriptor="ds3"/>
    <media id="bt3_red" src="buttons/3_red.png" descriptor="ds3"/>

    <!- BEGIN VIDEO: Start nodos Lua ->
    <link xconnector="onBeginStart">
        <bind role="onBegin" component="lua1"/>
        <bind role="start" component="lua2"/>
        <bind role="start" component="lua3"/>
    </link>

    <!- BEGIN LUA: Starts Run_Boton ->
    <link xconnector="con#onBeginStart">
        <bind role="onBegin" component="lua1"/>
        <bind role="start" component="bt1_green"/>
    </link>

    <link xconnector="con#onBeginStart">
        <bind role="onBegin" component="lua2"/>
        <bind role="start" component="bt2_green"/>
    </link>

    <link xconnector="con#onBeginStart">
        <bind role="onBegin" component="lua3"/>
```

```
        <bind role="start" component="bt3_green"/>
    </link>
    <!– END LUA: Stops Run_Boton, Starts End_Boton –>
    <link xconnector="con#onEndStopStart">
        <bind role="onEnd" component="lua1"/>
        <bind role="stop" component="bt1_green"/>
        <bind role="start" component="bt1_red"/>
    </link>
    <link xconnector="con#onEndStopStart">
        <bind role="onEnd" component="lua2"/>
        <bind role="stop" component="bt2_green"/>
        <bind role="start" component="bt2_red"/>
    </link>
    <link xconnector="con#onEndStopStart">
        <bind role="onEnd" component="lua3"/>
        <bind role="stop" component="bt3_green"/>
        <bind role="start" component="bt3_red"/>
    </link>
  </body>
</ncl>
```

Codigo del archivo 1.lua

vacio.

Codigo del archivo 2.lua

```
function handler (evt)
  event.post {
      class = 'ncl',
      type = 'presentation',
      action = 'stop',
  }
end
event.register(handler)
```

Codigo del archivo 3.lua

```
function handler (evt)
  event.timer(3000, function()
      event.post {
          class = 'ncl',
          type = 'presentation',
          action = 'stop',
    }
  end)
end
event.register(handler)
```

4.8.2. Contador de Clics:

En esta aplicación NCL se muestra un botón "*Clic aquí*" en cuatro momentos diferentes. Si el usuario selecciona con el control remoto durante al menos tres

veces, al final de la presentación es exhibida la imagen "*Ganaste*", de lo contrario se muestra la imagen "*Perdiste*".

No es posible hacer un conteo de los clics puramente en NCL de una forma simple, ya que no hay soporte para expresiones matemáticas en el lenguaje. En este ejemplo se utiliza un documento NCLua para contar y almacenar el número de clics en una propiedad, que será consultada al final para determinar el resultado.

Este ejemplo también introduce el uso de los eventos de atribución, utilizado para controlar las propiedades de los nodos. La mecánica del uso de eventos de atribución es idéntica a los de presentación, una vez que comparten el mismo modelo de máquinas de estado.

El documento NCL mostrado a continuación define un temporizador ("*temp*") con cuatro anclas temporales ("area1" hasta "area4") durante las cuales el botón "*Clic Aquí*" es exhibido. El temporizador también define un anclaje temporal para mostrar el resultado después de que el botón aparece cuatro veces, tambien el documento define un NCLua responsable de contar los clics y exporta a la propiedad "*counter*" para mantener este valor

Paso 1: Creación del nodo media NCLua "clics.lua".

Se define una propiedad inc en NCLua que incrementa un contador donde se selecciona el botón (para un enlace onSelectionSet).

```
<media id="clics" src="clics.lua">
  <property name="inc"/>
  <property name="counter"/>
</media>
```

Paso 2: Creación de los conectores para la visualización del botón y ejecución del programa NCLua.

La puerta de entrada de la aplicación es el temporizador, también activa el NCLua por medio de un enlace "***onBeginStart***". Cada anclaje de exhibición del botón ***"Clic Aquí"*** posee un enlace "***onBeginStart***" y "***onEndStop***" para el botón. Cada vez que el botón es seleccionado por el usuario (que sólo puede acontecer mientras está siendo exhibido), el ancla "***inc***" de NCLua es incrementado en 1 y el botón es ocultado, conforme al enlace **"*onSelectionStopSet*"** fuera del propio botón.

El tratamiento dado al ancla "***inc***" y la propiedad "***counter***" son especificados en el código NCLua . Al final del ancla "***resultado***" del temporizador, dos enlaces prueban si el valor de la propiedad "***counter***" es mayor, igual o menor que tres clics. Dependiendo del resultado, la imagen "***Ganaste***" o "***Perdiste***" es mostrada.

Paso 3: Creación del nodo NCLua.

Como se ha indicado, el incremento es accionado por un *simpleAction* de *set*. Como se indicó en el primer ejemplo del tutorial, las acciones se envían a los tratadores de eventos de NCLua. La acción de *set* que es un sobrenombre para una acción *start* en un evento del tipo *atribucion*, por lo tanto, el evento generado es la siguiente:

```
evt = {
  class = 'ncl',
```

```
    type = 'attribution',
    property = 'inc',
    action = 'start',
    value = '1',
  }
```

Se debe tener en cuenta que los valores de las propiedades de un NCLua (o cualquier otro objeto media) están controlados por el formateador NCL. Por lo tanto, el NCLua debe notificar el formateador que el valor de la propiedad *counter* se modificó.

Codigo del ejemplo NCLua 2.

Codigo del programa principal.

```
<ncl id="nclClics" .....">
  <head>
    <regionBase>
        <region id="rgBoton" width="20 %" height="20 %" left="40 %"
        top="40 %" id="rgButton"/>
        <region id="regfondo" width="100 %" height="100 %"/>
    </regionBase>

    <descriptorBase>
        <descriptor id="dsBoton" region="rgBoton" focusIndex="1"/>
        <descriptor id="desfondo" region="regfondo"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="entryPoint" component="timer"/>
    <port id="inFondo" component="fondo"/>

    <media id="fondo" type="image/jpeg" src="media/fondo.jpg"
    descriptor="desfondo" />

    <!- nodo de tiempo ->
    <media id="timer" type="application/x-ginga-time">
    <!- anchors to exhibit the "Click it" button ->
        <area id="area01" begin="3s" end="6s"/>
        <area id="area02" begin="10s" end="13s"/>
        <area id="area03" begin="17s" end="20s"/>
        <area id="area04" begin="24s" end="27s"/>
        <!- anchor to display the final score ->
        <area id="areaTotal" begin="35s"/>
    </media>

    <!-nodo lua ->
    <media id="clicks" src="clics.lua">
        <!- inc function ->
        <property name="inc"/>
        <!- counter variable ->
```

```
        <property name="counter"/>
    </media>

    <!– boton "Click aqui" –>
    <media id="boton" descriptor="dsButton" src="media/boton.jpg"/>

    <!– button "You won" –>
    <media id="ganaste" descriptor="dsButton" src="media/ganaste.jpg"/>

    <!– button "You lose"–>
    <media id="perdiste" descriptor="dsButton" src="media/perdiste.jpg"/>

    <!– Inicialización del nodo lua –>
    <link xconnector="onBeginStart">
        <bind role="onBegin" component="timer"/>
        <bind role="start" component="clicks"/>
    </link>

    <!– Exhibición del botón "Click aqui" para su selección en los intervalos
–>
    <link xconnector="onBeginStart">
        <bind role="onBegin" component="timer" interface="area01"/>
        <bind role="start" component="boton"/>
    </link>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="timer" interface="area02"/>
        <bind role="start" component="boton"/>
    </link>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="timer" interface="area03"/>
        <bind role="start" component="boton"/>
    </link>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="timer" interface="area04"/>
        <bind role="start" component="buton"/>
    </link>

    <!–Stop del botón "Click aqui" en los intervalos de tiempo –>
    <link xconnector="onEndStop">
        <bind role="onEnd" component="timer" interface="area01"/>
        <bind role="stop" component="boton"/>
    </link>

    <link xconnector="onEndStop">
        <bind role="onEnd" component="timer" interface="area02"/>
        <bind role="stop" component="boton"/>
    </link>

    <link xconnector="onEndStop">
        <bind role="onEnd" component="timer" interface="area03"/>
        <bind role="stop" component="boton"/>
    </link>

    <link xconnector="onEndStop">
```

```
        <bind role="onEnd" component="timer" interface="area04"/>
        <bind role="stop" component="boton"/>
    </link>

    <link xconnector="onEndStopN">
        <bind role="onEnd" component="timer"/>
        <bind role="stop" component="boton"/>
        <bind role="stop" component="ganaste"/>
        <bind role="stop" component="perdiste"/>
        <bind role="stop" component="clicks"/>
    </link>

    <!– Llamado a la función "Clicks" si es presionado el botón –>
        <link xconnector="onSelectionStopSet">
        <bind role="onSelection" component="boton"/>
        <bind role="stop" component="boton"/>
        <bind role="set" component="clicks" interface="inc">
            <bindParam name="var" value="1"/>
        </bind>
    </link>

    <!– exhibicion del resultado –>
        <link xconnector="onCondGteBeginStart">
        <linkParam name="var" value="3"/>
        <bind role="onBegin" component="timer" interface="areaTotal"/>
        <bind role="attNodeTest" component="clicks" interface="counter"/>
        <bind role="start" component="ganaste"/>
    </link>

    <link xconnector="onCondLtBeginStart">
        <linkParam name="var" value="3"/>
        <bind role="onBegin" component="timer" interface="areaTotal"/>
        <bind role="attNodeTest" component="clicks" interface="counter"/>
        <bind role="start" component="perdiste"/>
    </link>
  </body>
</ncl>
```

Codigo del archivo clicks.lua

```
counter = 0
local counterEvt = {
  class = 'ncl',
  type = 'attribution',
  property = 'counter',
}

function handler (evt)
  if evt.class ~= 'ncl' then return end
  if evt.type ~= 'attribution' then return end
  if evt.property ~= 'inc' then return end

  counter = counter + evt.value

  event.post {
      class = 'ncl',
```

```
        type = 'attribution',
        property = 'inc',
        action = 'stop'
    }
    counterEvt.value = counter
    counterEvt.action = 'start'; event.post(counterEvt)
    counterEvt.action = 'stop'; event.post(counterEvt)
end
event.register(handler)
```

4.8.3. Gráficos y Control Remoto:

En este ejemplo se desplaza una imagen por la pantalla hasta alcanzar otra, en este caso se utiliza la imagen de un mono para que sea la que se desplaza hasta colocarse junto a la segunda imagen que en este caso es un plátano. Se debe tener en cuenta que la imagen del mono puede salir de la pantalla y rodear el plátano.

Paso 1: Creación del documento NCL.

En este ejemplo se deja un poco de lado la integración NCL-Lua y muestra una aplicación más autosuficiente en que la comunicación con el documento NCL se produce sólo en el comienzo de la presentación del documento y al final del juego. El juego comienza tan pronto como se carga el documento.

<port id="entryPoint" component="lua"/>

Cuando la imagen del mono alcanza la banana, el NCLua señaliza el inicio del ancla *fin* y el documento exhibe la imagen de *victoria*.

En los ejemplos anteriores, el código de la aplicación se concentró en el documento NCL, en este se observa que prácticamente todo el código se encuentra en su interior del NCLua.

Paso 2: Creación de objetos para representar el mono y la banana en NCLua.

En la primera línea de código hay una referencia para el paquete ***canvas***, con el cual todas las operaciones gráficas son efectuadas. La llamada a ***canvas:new*** carga fragmentos de la imagen *mono.png* que se guarda en la variable *img*. La última línea crea una tabla que almacena las propiedades de mono: su imagen, posición (x,y) y tamaño (dx,dy). El uso de tablas es similar a la de los objetos en lenguajes orientados a objetos y es bastante recurrente en Lua.

```
local img = canvas:new('mono.png')
local dx, dy = img:attrSize()
local mono = { img=img, x=10, y=10, dx=dx, dy=dy }
```

El código para representar a *bananos* es bastante similar al extracto por de la creación de *mono*. La variable ***canvas*** referencia la región NCL destinados a NCLua. Esto significa que esta variable tendrá un valor igual a ***nil*** en un NCLua cuyo documento NCL correspondiente no define una región para el enlace.

Paso 3: Definición de la función del rediseño para realizar los movimientos de la imagen.

En esta función, utilizando el método *canvas:drawRect*, que dibuja un rectángulo que pinta toda la pantalla de color negro (configurado previamente). Las dos

líneas siguientes utilizando el método ***canvas:compose*** para diseñar el plátano y el mono sobre el ***canvas***.

Por último, el ***canvas*** se actualiza con una llamada a ***canvas:flush***.

Paso 4:Definición de la función que indica la superposición de las imágenes.

Esta función recibe dos objetos como mono y plátano y retorna si ambos son superpuestos. Utilizando la misma estructura de la tabla, nuevos personajes puede ser incluido en el juego y utiliza la misma función de colisión.

function collide (A, B)

Hasta este punto sólo se definen los objetos (mono y banana) y funciones (dibujar y chocar), una vez más, el script es sólo un inicializador y las acciones que se toman sólo en respuesta a los eventos.

Paso 5:Declaración de la función de tratamiento de eventos

En la función de control de eventos es en donde toda la acción del juego sucede. En este ejemplo solo interesa tratar los eventos de pulsación de teclas, por tanto, la función debe filtrarlos:

```
if (evt.class == 'key') and (evt.type == 'press') then
  ...
end
```

El tratador de teclas comprueba el valor de la tecla y altera la posición del mono de acuerdo:

```
if evt.key == 'CURSOR_UP' then
  mono.y = mono.y - 10
elseif evt.key == 'CURSOR_DOWN' then
  mono.y = mono.y + 10
elseif evt.key == 'CURSOR_LEFT' then
  mono.x = mono.x - 10
elseif evt.key == 'CURSOR_RIGHT' then
  mono.x = mono.x + 10
end
```

Un evento de tecla lleva la información *type* que indica si el botón esta siendo pulsado o liberado, y *key* que indica el valor de la tecla en cuestión.

Después de que el movimiento del mono, la función de colicion con un plátano:

```
if collide(mono, banana) then
  ...
end
```

Si hay una colisión, el NCLua señala el formateador para su anclaje *fim* y comienza una marca un *flag* (IGNORE = true) para que los eventos futuros sean ignorados.

Codigo del ejemplo NCLua 3.

Codigo del programa NCL.

```
<ncl id="Graficos" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
      <region id="rgGanaste" width="20%" height="20%" left="10%"
```

```
        top="40 %" />
        <region id="rgLua" width="50 %" height="50 %" right="10 %"
        top="25 %" />
    </regionBase>

    <descriptorBase>
        <descriptor id="dsLua" region="rgLua" focusIndex="luaIdx"/>
        <descriptor id="dsGanaste" region="rgGanaste"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="entryPoint" component="lua"/>
    <media type="application/x-ginga-settings" id="programSettings">
        <property name="service.currentKeyMaster" value="luaIdx"/>
    </media>

    <media id="lua" src="mono.lua" descriptor="dsLua">
        <area id="fim"/>
    </media>

    <media id="ganaste" src="Ganaste.jpg" descriptor="dsGanaste"/>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="lua" interface="fim"/>
        <bind role="start" component="ganaste"/>
    </link>
  </body>
</ncl>
```

Codigo del archivo mono.lua

```
– mono: guarda la imagen, posición inicial y dimensiones
local img = canvas:new('mono.png')
local dx, dy = img:attrSize()
local mono = { img=img, x=10, y=10, dx=dx, dy=dy }

– Banana: guarda la imagen, posición inicial y dimensiones
local img = canvas:new('banana.png')
local dx, dy = img:attrSize()
local banana = { img=img, x=150, y=150, dx=dx, dy=dy }

– Función del rediseño:
– Llamada a cada botón pulsado
– Primero la parte del fondo, a continuación, el plátano y finalmente la
imagen del mono.
function redraw ()
  canvas:attrColor('black')
  canvas:drawRect('fill', 0,0,
  canvas:attrSize())
  canvas:compose(banana.x, banana.y, banana.img)
  canvas:compose(mono.x, mono.y, mono.img)
```

```
  canvas:flush()
end

– Función del rediseño:
– Llamada a cada botón pulsado
– Revisa si el mono esta sobre la banana.
function collide (A, B)
  local ax1, ay1 = A.x, A.y
  local ax2, ay2 = ax1+A.dx, ay1+A.dy
  local bx1, by1 = B.x, B.y
  local bx2, by2 = bx1+B.dx, by1+B.dy

  if ax1 > bx2 then
      return false
  elseif bx1 > ax2 then
      return false
  elseif ay1 > by2 then
      return false
  elseif by1 > ay2 then
      return false
  end
  return true
end

local IGNORE = false


– Función de tratamiento de eventos:
function handler (evt)
  if IGNORE then
      return
  end

  – Solo interesan los eventos de los botones
  if evt.class == 'key' and evt.type == 'press'
  then
      – Sólo las flechas mueve la imagen del mono
      if evt.key == 'CURSOR_UP' then
          mono.y = mono.y - 10
      elseif evt.key == 'CURSOR_DOWN' then
          mono.y = mono.y + 10
      elseif evt.key == 'CURSOR_LEFT' then
          mono.x = mono.x - 10
      elseif evt.key == 'CURSOR_RIGHT' then
          mono.x = mono.x + 10
      end

      – Comprueba si el mono está en la parte superior del banano
      if collide(mono, banana) then
          – caso esteja, sinaliza que a ancora de FIM esta ocorrendo
          event.post {
            class = 'ncl',
            type = 'presentation',
            label = 'fim',
```

```
            action = 'start',
          }
          – e ignora los eventos posteriores
          IGNORE = true
        end
    end

    – Redibuja toda la pantalla
    redraw()
end
event.register(handler)
```

4.8.4. Animaciones:

Para realizar las animaciones se utiliza la clase ***user***, estas se realizan por medio de la utilización del API de Lua que ofrece para televisión digital. Se debe considerar que simplemente no se puede crear un bucle en el controlador de eventos para animar el corredor, ya que en cuanto este ejecutándose la animación, ningún otro evento seria tratado. En NCLua no hay soporte para multi-threading , por lo que no se puede simplemente disparar el corredor en un nuevo proceso.

La solución es atravez de dos recursos sin explotar: la función ***uptime*** y la clase de evento ***user*** . La idea es crear eventos de la clase ***user***, mientras que el corredor no llega a la línea de meta. Por lo tanto, el controlador calcula cuánto tiempo ha pasado a cada llegada del evento y se basan en que calcula cuánto el corredor debe moverse

Paso 1:Creación del tratador de eventos.
El primer ***if*** filtra sólo el evento de inicio (*start*) y eventos de clase ***user*** que se utilizarán para realimentar la animación. Al llamar a la función ***uptime*** esta retorna el tiempo transcurrido desde el inicio de la NCLua.

En el caso de una clase de evento ***user***, el campo *evt.time* tiene el tiempo del último ciclo de la animación, sobre este campo se calcula la diferencia para el tiempo actual y la posición del corredor se actualiza. La posición del corredor se pone a prueba y si no ha llegado todavía a la línea de meta, un evento de la nueva clase ***user*** se registra con el momento actual.

La animación utiliza dos cuadros del corredor para dar la sensación de que este mueve sus piernas. El cuadro actual se calcula con el fin de cambiar entre 0 y 1 a cada 5 píxeles de movimiento.

Paso 2:Definición del controlador que redibuja la función.
El fondo se dibuja en negro y las líneas de salida y llegada en blanco. Por último la imagen del corredor se dibuja sobre canvas. La función compose puede recibir partes de la imagen como parámetros que deben ser compuestas sobre canvas. En el ejemplo una de las dos mitades de la imagen se dibuja cada cinco pixeles (ver Figura 4.22).

Figura 4.22: Imagen del corredor.

Se puede notar que la imagen del corredor es una franja con los cuadros de animación.

Codigo del ejemplo NCLua 4.

Codigo del programa NCL.

```
<ncl id="Graficos" .....>
  <head>
    <regionBase>
        <region id="rgLua" width="90 %" height="20 %" left="5 %"
        top="40 %" />
        <region id="rgWin" width="20 %" height="20 %" left="10 %"
        top="40 %" />
    </regionBase>

    <descriptorBase>
        <descriptor id="dsLua" region="rgLua" focusBorderColor="white"/>
        <descriptor id="dsWin" region="rgWin"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="entryPoint" component="lua"/>
    <media id="lua" src="anim.lua" descriptor="dsLua"/>
  </body>
</ncl>
```

Codigo del archivo anim.lua

```
– Corredor: guarda la imagen, posición inicial y dimensiones
local img = canvas:new('runner.png')
local dx, dy = img:attrSize()
local runner = { img=img, frame=0, x=50-dx/2, y=10, dx=dx, dy=dy }

– Función del rediseño:
– Llamada a ciclo de animación
local dx, dy = canvas:attrSize()
local INI, END = 50, dx-58
function redraw ()
  – fondo
  –canvas:attrColor('black')
  –canvas:drawRect('fill', 0,0, dx,dy)

  – linea de largada y llegada
  canvas:attrColor('white')
```

```
    canvas:drawRect('fill', INI,0, 8,dy)
    canvas:drawRect('fill', END,0, 8,dy)

    – corredor
    local dx2 = runner.dx/2
    runner.img:attrCrop(runner.frame*dx2,0, dx2,runner.dy)
    canvas:compose(runner.x, runner.y, runner.img)
    canvas:flush()
  end

  – Función de tratamiento de eventos
  function handler (evt)
    – La animación comienza en 'start' y es realimentada por eventos da classe
'user'
    if (evt.class == 'ncl' and evt.type == 'presentation' and evt.action ==
'start') or
        (evt.class == 'user') then
        local now = event.uptime()

        – Mueve el corredor si el tiempo ya ha pasado
        if evt.time then
            local dt = now - evt.time
            runner.x = runner.x + dt/50
        end

        – Si no ha alcanzado su meta, sigue dando ciclos a la animación
        if runner.x < END then
            event.post('in', { class='user', time=now })
        end

        – Modifica el marco del corredor cada 5 píxeles
        runner.frame = math.floor(runner.x/5) % 2
        redraw()
    end
  end
  event.register(handler)
```

4.8.5. Paso de valores:

En este ejemplo se explora el paso de valores de propiedades entre nodos NCLua. Un campo de entrada y dos de salida son exhibidos en la pantalla, en cuanto el usuario inicie el programa (presionando la tecla *ENTER*) escoge la posición desde la que desea que inicie el texto, el primer campo de salida es automáticamente actualizado con el valor de la entrada. En cuanto el usuario presione la tecla *ENTER*, el valor de entrada se copia en el segundo campo de salida.

La escritura del texto en este ejemplo debe ser realizada por medio de las teclas del computador que simulan los botones del control remoto, para dar la entrada de los caracteres, es decir el teclado numérico del "0-9".

Paso 1:Definición de los nodos NCLua de entrada y salida.

Los nodos de entrada y salida son implementados en los archivos *entrada.lua* y *salida.lua*, respectivamente. Ambos tratan la propiedad ***text*** que contiene el texto de visualización. En el caso del NCLua de entrada, el campo ***text*** es controlado por el propio programa, y es alterado cada vez que el usuario presiona una nueva tecla. En el caso del NCLua de salida el campo debe ser comparado por el documento NCL atravez de los enlaces de ***set***.

En este ejemplo se tratan los NCLuas como si fueran cajas negras, no importa el contenido de los archivos *.lua* y el autor del documento NCL solo conoce la interfaz que cada uno de los NCLua ofrece con sus propiedades ***text***, de esta manera, los archivos *.lua* puede ser reutilizado en otras aplicaciones.

El campo de entrada es representado por el siguiente código:

```
<media id="input" src="entrada.lua" descriptor="dsInput">
  <area id="select"/>
  <property name="text"/>
</media>
```

También tiene el ancla select que se inicia cuando el usuario presiona *ENTRAR*. Los campos de salida son representadas por el código siguiente:

```
<media id="output1" src="salida.lua" descriptor="dsOutput1">
  <property name="text"/>
</media>

<media id="output2" src="salida.lua" descriptor="dsOutput2">
  <property name="text"/>
</media>
```

Paso 2:Definición de los enlaces que copian el contenido del campo de entrada en los campos de salida.

La parte más importante del documento es la que contiene los enlaces responsables de copiar el contenido del campo de entrada en los de salida. El primer campo de salida es actualizado siempre que la propiedad ***text*** del campo de entrada es modificado (*"onEndAttributionSet"*).

El enlace con la función ***rol= "get"*** permite consultar el valor de una propiedad de cualquier nodo, guardandolo en la variable ***$get***. En este ejemplo la propiedad consultada es la de ***text*** del nodo de *entrada*. La variable ***$get***, a su vez, se utiliza para enlazar con ***role= "set"***, de modo que el valor de entrada es copiado en la "*salida 1*".

El segundo campo de salida se actualizará cuando el usuario presiona la tecla *ENTER* en el nodo de entrada (*"onBeginSet"*). EL mecanismo de ***role="get"*** también es usado para copiar el valor de entrada.

Codigo del ejemplo NCLua 5.

Codigo del programa NCL.

```
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
        <region width="60%" height="15%" left="10%"
        top="10%" id="rgInput"/>
```

```
        <region width="60 %" height="15 %" left="10 %"
        top="40 %" id="rgOutput1"/>
        <region width="60 %" height="15 %" left="10 %"
        top="70 %" id="rgOutput2"/>
    </regionBase>

    <descriptorBase>
        <descriptor id="dsInput" region="rgInput" focusIndex="inputIdx"/>
        <descriptor id="dsOutput1" region="rgOutput1"/>
        <descriptor id="dsOutput2" region="rgOutput2"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
</head>

<body>
    <port id="entryPoint" component="input"/>

    <media type="application/x-ginga-settings" id="programSettings">
        <property name="service.currentKeyMaster" value="inputIdx"/>
    </media>
    <media id="input" src="input.lua" descriptor="dsInput">
        <area id="select"/>
        <property name="text"/>
    </media>
    <media id="output1" src="output.lua" descriptor="dsOutput1">
        <property name="text"/>
    </media>
    <media id="output2" src="output.lua" descriptor="dsOutput2">
        <property name="text"/>
    </media>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="input"/>
        <bind role="start" component="output1"/>
        <bind role="start" component="output2"/>
    </link>
    <link xconnector="onEndAttributionSet">
        <bind role="onEndAttribution" component="input" interface="text"/>
        <bind role="set" component="output1" interface="text">
            <bindParam name="var" value="$get"/>
        </bind>
        <bind role="get" component="input" interface="text"/>
    </link>
    <link xconnector="onBeginSet">
        <bind role="onBegin" component="input" interface="select"/>
        <bind role="set" component="output2" interface="text">
            <bindParam name="var" value="$get"/>
        </bind>
        <bind role="get" component="input" interface="text"/>
    </link>
```

```
  </body>
</ncl>
```

Codigo del archivo input.lua

```
text = nil
local TEXT = ''
local CHAR = ''

local KEY, IDX = nil, -1
local MAP = {
    ['1'] = { '1', '.', ',' }
  , ['2'] = { 'a', 'b', 'c', '2' }
  , ['3'] = { 'd', 'e', 'f', '3' }
  , ['4'] = { 'g', 'h', 'i', '4' }
  , ['5'] = { 'j', 'k', 'l', '5' }
  , ['6'] = { 'm', 'n', 'o', '6' }
  , ['7'] = { 'p', 'q', 'r', 's', '7' }
  , ['8'] = { 't', 'u', 'v', '8' }
  , ['9'] = { 'w', 'x', 'y', 'z', '9' }
  , ['0'] = { '0' }
}

local UPPER = false
local case = function (c)
  return (UPPER and string.upper(c)) or c
end

local dx, dy = canvas:attrSize()
canvas:attrFont('vera', 3*dy/4)
function redraw ()
  canvas:attrColor('black')
  canvas:drawRect('fill', 0,0, dx,dy)

  canvas:attrColor('white')
  canvas:drawText(0,0, TEXT..case(CHAR)..'|')

  canvas:flush()
end

local evt = {
  class = 'ncl',
  type = 'attribution',
  name = 'text',
}

local function setText (new, outside)
  TEXT = new or TEXT..case(CHAR)
  text = TEXT
  CHAR, UPPER = '', false
  KEY, IDX = nil, -1

  -- notifica al documento NCL
  if not outside then
      evt.value = TEXT
      evt.action = 'start'; event.post(evt)
```

```
      evt.action = 'stop'; event.post(evt)
  end
end

local TIMER = nil
local function timeout ()
  return event.timer(1000,
      function()
        if KEY then
          setText()
        end
      end)
end

local function nclHandler (evt)
  if evt.class ~= 'ncl' then return end

  if evt.type == 'attribution' then
      if evt.name == 'text' then
        setText(evt.value, true)
      end
  end

  redraw()
  return true
end
event.register(nclHandler)

local sel = {
  class = 'ncl',
  type = 'presentation',
  label = 'select',
}

local function keyHandler (evt)
  if evt.class ~= 'key' then return end
  if evt.type ~= 'press' then return true end
  local key = evt.key

  -- SELECT
  if (key == 'ENTER') then
      setText()
      sel.action = 'start'; event.post(sel)
      sel.action = 'stop'; event.post(sel)

  -- BACKSPACE
  elseif (key == 'CURSOR_LEFT') then
      setText( (KEY and TEXT) or string.sub(TEXT, 1, -2) )

  -- UPPER
  elseif (key == 'CURSOR_UP') then
      UPPER = not UPPER

  -- SPACE
  elseif (key == 'CURSOR_RIGHT') then
      setText( (not KEY) and (TEXT..' ') )
```

```
    – NUMBER
    elseif _G.tonumber(key) then
        if KEY and (KEY ~= key) then
          setText()
        end
        IDX = (IDX + 1) % #MAP[key]
        CHAR = MAP[key][IDX+1]
        KEY = key
    end

    if TIMER then TIMER() end
    TIMER = timeout()
    redraw()
    return true
  end
  event.register(keyHandler)
```

Codigo del archivo output.lua

```
  local TEXT = ' '

  local dx, dy = canvas:attrSize() canvas:attrFont('vera', 3*dy/4)
  function redraw ()
    canvas:attrColor('black')
    canvas:drawRect('fill', 0,0, dx,dy)

    canvas:attrColor('white')
    canvas:drawText(0,0, TEXT)
    canvas:flush()
  end

  local function handler (evt)
    if evt.class ~= 'ncl' then return end

    if evt.type == 'attribution' then
        if evt.name == 'text' then
          if evt.action == 'start' then
            TEXT = evt.value
            evt.action = 'stop'
            event.post(evt)
          end
        end
    end

    redraw()
  end
  event.register(handler)
```

4.8.6. Consulta a Google:

En el ejemplo se presenta en pantalla un campo de entrada y otro de salida, después de que el usuario escriba el tema que desea consultar en el campo de entrada este es buscado en Google. El resultado de la búsqueda (la dirección URL devuelta por "voy a tener suerte") se ha cargado en el campo resultado.

Paso 1:Definición del nodo de entrada y el de salida.

El nodo de entrada tiene la propiedad text, que mantendrá el contenido actual, y el ancla ***select***, se inicia cuando el usuario pulsa *ENTER* :

```
<media id="input" src="input.lua" descriptor="dsInput">
  <area id="select"/>
  <property name="text"/>
</media>
```

El nodo de salida sólo tiene la propiedad ***text***, cuyo valor se muestra la pantalla.

```
<Media Id="output" src="output.lua" descriptor="dsOutput">
  <propiedad name="text"/>
</media>
```

Paso 2:Definición del nodo google.

El nodo google posee dos propiedades ***search*** y ***result***

```
<media id="google" src="google.lua">
  <property name="search"/>
  <property name="result"/>
</media>
```

Cuando el valor de ***search*** es alterado, el escript realiza la consultado con el ultimo valor y el resultado es guardado en la propiedad ***result***.

El enlace *"onBeginSet"* inicia la consulta por lo tanto el ancla ***select*** del nodo de entrada se inicia, es decir, cuando el usuario presiona *ENTER*. El valor de entrada es copiado para la propiedad ***search*** del nodo google. Cuando la búsqueda termina, el nodo google altera su propiedad ***result***, causando que el enlace ***"onEndAttributionSet"*** actualice el campo de salida.

Paso 3:Establecimiento de la comunicación mediante TCP.

La comunicación se realiza a través de TCP mediante el uso de classe de eventos *'tcp'*. Su uso no es trivial, ya que las llamadas son asincrónicas y el programador debe controlar cada paso de la comunicación (conexión, enviar, recibir y desconexión) y llamadas desconectadas.

Para facilitar la programación, se define junto con este ejemplo, el archivo *tcp.lua* . Este módulo admite el control de llamadas asincrónicas, permitiendo que la comunicación sea programada secuencialmente.

La función principal, *tcp.execute*, es recibir una función que puede contener los siguientes comandos:

- tcp.connect: recibe la dirección y el puerto de destino
- tcp.send: recibe una cadena con el valor que se transmite
- tcp.receive: devuelve una cadena con el valor que se recibe
- tcp.disconnect: sierra la conexión

Codigo del ejemplo NCLua 5.

Codigo del programa NCL.

```
<ncl id="main" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
  <head>
    <regionBase>
        <region width="80 %" height="15 %" left="10 %"
        top="20 %" id="rgInput"/>
        <region width="80 %" height="10 %" left="10 %"
        top="50 %" id="rgOutput"/>
    </regionBase>

    <descriptorBase>
        <descriptor id="dsInput" region="rgInput" focusIndex="inputIdx"/>
        <descriptor id="dsOutput1" region="rgOutput"/>
    </descriptorBase>

    <connectorBase>
        <importBase documentURI="conectores.ncl" alias="con"/>
    </connectorBase>
  </head>

  <body>
    <port id="entryPoint" component="input"/>

    <media type="application/x-ginga-settings" id="programSettings">
        <property name="service.currentKeyMaster" value="inputIdx"/>
    </media>
    <media id="input" src="input.lua" descriptor="dsInput">
        <area id="select"/>
        <property name="text"/>
    </media>
    <media id="google" src="google.lua">
        <property name="search"/>
        <property name="result"/>
    </media>
    <media id="output" src="output.lua" descriptor="dsOutput2">
        <property name="text"/>
    </media>

    <link xconnector="onBeginStart">
        <bind role="onBegin" component="input"/>
        <bind role="start" component="google"/>
        <bind role="start" component="output"/>
    </link>
    <link xconnector="onBeginSet">
        <bind role="onBegin" component="input" interface="select"/>
        <bind role="set" component="google" interface="search">
            <bindParam name="var" value="$get"/>
        </bind>
        <bind role="get" component="input" interface="text"/>
    </link>
  </body>
</ncl>
```

Codigo del archivo input.lua

El código es el mismo que en el ejemplo 5.

Codigo del archivo output.lua

El código es el mismo que en el ejemplo 5.

Codigo del archivo google.lua

```
require 'tcp'

function handler (evt)
  if evt.class ~= 'ncl'        then return end
  if evt.type ~= 'attribution'        then return end
  if evt.action ~= 'start'        then return end
  if evt.name ~= 'search'        then return end

  event.post {
     class = 'ncl',
     type = 'attribution',
     name = 'search',
     action = 'stop',
  }

  tcp.execute(
     function ()
       tcp.connect('www.google.com.br', 80)
       tcp.send('GET /search?hl=pt-BR&btnI&q='..evt.value..'\n')
       local result = tcp.receive()
       if result then
          result = string.match(result, 'Location: http://(.-)\r?\n') or
          'nao encontrado'
       else
          result = 'error: ' .. evt.error
       end
       local evt = {
          class = 'ncl',
          type = 'attribution',
          name = 'result',
          value = result,
       }
       evt.action = 'start'; event.post(evt)
       evt.action = 'stop' ; event.post(evt)
       tcp.disconnect()
     end
  )
end
event.register(handler)
```

Codigo del archivo parser.lua

```
local str = [[<div class="al adr"><span class=adr id=sxaddr dir=ltr>
<span class=street-address>Av Afrânio Melo
Franco</span>, <span class=street-address>209</span> -
<span class=locality>Rio de Janeiro</span> -
<span class=region>RJ</span>, <span class=postal-code>
22430-060</span>, <span class=country-name>Brazil</span></span>
```

```
&#8206; - <span dir=ltr class=nw><span class=tel id=sxphone>
(0xx)21 3183-8215</span></span>&#8206;</div><div class=sa>
</div><span style=color:#77c><a class=a href=/maps?li=d&amp;
hl=en&amp;f=d&amp;iwst]]
```

```
local street, city, state, cep, country, fone =
      str:match ('<div class="al adr">.-'
         .. '<span class=street %-address>(.-)</span>.-'
         .. '<span class=locality>(.-)</span>.-'
         .. '<span class=region>(.-)</span>.-'
         .. '<span class=postal %-code>(.-)</span>.-'
         .. '<span class=country %-name>(.-)</span>.-'
         .. '<span class=tel id=sxphone>(.-)</span>')
print(street, city, state, cep, country, fone)
```

Codigo del archivo parser.lua

```
local _G, coroutine, event, assert, pairs, type
  = _G, coroutine, event, assert, pairs, type
local s_sub = string.sub

module 'tcp'

local CONNECTIONS = {}

local current = function ()
  return assert(CONNECTIONS[assert(coroutine.running())])
end

local resume = function (co, ...)
  assert(coroutine.status(co) == 'suspended')
  assert(coroutine.resume(co, ...))
  if coroutine.status(co) == 'dead' then
     CONNECTIONS[co] = nil
  end
end
function handler (evt)
  if evt.class ~= 'tcp' then return end

  if evt.type == 'connect' then
     for co, t in pairs(CONNECTIONS) do
        if (t.waiting == 'connect') and
           (t.host == evt.host) and (t.port == evt.port) then
           t.connection = evt.connection
           t.waiting = nil
           resume(co)
           break
        end
     end
     return
  end

  if evt.type == 'disconnect' then
     for co, t in pairs(CONNECTIONS) do
        if t.waiting and
           (t.connection == evt.connection) then
```

```
                t.waiting = nil
                resume(co, nil, 'disconnected')
            end
        end
        return
    end

    if evt.type == 'data' then
        for co, t in pairs(CONNECTIONS) do
            if (t.waiting == 'data') and
                (t.connection == evt.connection) then
                resume(co, evt.value)
            end
        end
        return
    end
end
event.register(handler)

function execute (f, ...)
    resume(coroutine.create(f), ...)
end

function connect (host, port)
    local t = {
        host = host,
        port = port,
        waiting = 'connect'
    }
    CONNECTIONS[coroutine.running()] = t

    event.post {
        class = 'tcp',
        type = 'connect',
        host = host,
        port = port,
    }
    return coroutine.yield()
end

function disconnect ()
    local t = current()
    event.post {
        class = 'tcp',
        type = 'disconnect',
        connection = assert(t.connection),
    }
end

function send (value)
    local t = current()
    event.post {
        class = 'tcp',
        type = 'data',
```

```
        connection = assert(t.connection),
        value = value,
    }
end

function receive (pattern)
    pattern = pattern or '' -- TODO: '*l'/number
    local t = current()
    t.waiting = 'data'
    t.pattern = pattern
    if s_sub(pattern, 1, 2) ~= '*a' then
        return coroutine.yield()
    end
    local all = ''
    while true do
        local ret = coroutine.yield()
        if ret then
            all = all .. ret
        else
            return all
        end
    end
end
```

4.9. Conclusiones y Recomendaciones.

4.9.1. Conclusiones:

La TVD me brinda un cambio drástico respecto a lo que estamos acostumbrados al ver un programa de TV en la actualidad, ya que como usuarios cambiamos de tener un papel pasivo a tener un papel activo, pudiendo interactuar con la programación, brindándonos la oportunidad de poder interactuar con otros usuarios, o poder seleccionar información del programa que se está presentando en ese instante.

La selección de la herramienta para comenzar a desarrollar el contenido es lo más importante, debido a que para personas que no poseen conocimientos en programación Java se puede optar por la herramienta composer que brinda un ambiente más amigable ya que es del tipo de programación gráfica, pero esta posee muchas limitaciones en cuanto al soporte e interacción con lenguaje imperativo Lua que es el que complementa al lenguaje declarativo NCL, limitando la generación de contenido interactivo y la calidad de presentación del programa al usuario.

La utilización de los plugins de Eclipse NCL y NCLua, que están desarrollados sobre código libre y es sobre los cuales se basa el presente manual, me brindan una gran ventaja respecto a muchos otros programas de desarrollo de contenido interactivo, ya que al ejecutarse sobre la misma plataforma Eclipse me brindan una mejor estructura del programa, además al tratarse de que Lua es un script este me permite reutilizar código, generando programas que ocupan una menor cantidad de memoria y que el contenido interactivo solo pueda ser limitado por la imaginación del programador. Aunque el desarrollo del contenido sobre los

plugins es más complejo que cualquier otro programa de edición grafica este es el más completo respecto a las normas ABNT NBR y es el programa sobre el que se está desarrollando la mayor cantidad de contenido interactivo en Latinoamérica, por lo que me brinda un mayor soporte, y una evolución mucho más rápida que los otros programas.

Al desarrollar aplicaciones interactivas con canal de retorno se debe tener en cuenta que estas no están soportadas en un 100 % en la actualidad y que por el momento todavía esta opción se encuentra en desarrollo y está evolucionando constantemente, por este motivo para aplicaciones de este tipo solamente se simula el funcionamiento de la transmisión por medio de programas de código libre que aunque no respetan los tamaños de la taza de bits que puede trasmitir el canal, este tipo de práctica es muy útil para poder tener una idea de cómo funciona el mismo y estar familiarizados con el tema.

4.9.2. Recomendaciones:

Debido que el estándar Brasileño ISDB-Tb está basado en el desarrollo de contenido interactivo sobre software libre para la simulación de las aplicaciones al igual que para el desarrollo, siempre se debe trabajar sobre las últimas versiones que se publican en las páginas web oficiales de ginga, principalmente las de Brasil, ya que las versiones de Chile, Argentina, y Perú están basadas sobre el mismo y solamente poseen modificaciones menores, para cada país. Esto se comprobó ya que la última versión de VSTB de ginga ya posee una ventaja notoria sobre su versión anterior, que tenia una gran cantidad de problemas como la resolución, respuesta de ejecución y tamaño de pantalla, además la versión más reciente me ofrece la facilidad de seleccionar el tipo de resolución de pantalla facilitando el poder simular la presentación de los programas en varios tamaños de pantalla. El manual inicio desarrollándose sobre la versión anterior, pero antes de terminarlo se publicó la última versión del VSTB que es ubuntu-server10.10-ginga-i386, por lo que se optó por migrar todo el manual a la misma y es sobre el cual esta desarrollado.

Al trabajar con programas para desarrollar contenido interactivo para que pueda ser mostrado en pantalla, siempre se debe seleccionar un programa que me brinde soporte a todos los recursos de desarrollo que se desean implementar además de trabajar sobre las versiones más actualizadas ya sea de los programas o de los plugins que se ejecutan sobre otros programas y se utilizan para crear dichas aplicaciones, esto es ya que como se trata de software libre estos están en constante desenvolvimiento, haciendo que cada versión corrija los errores de las anteriores y cumplan en mayor parte con las normativas ABNT NBR, ya que como se pudo comprobar versiones más antiguas no tienen un soporte para todos los conectores que se detallan en dicha normativa o para la organización de un documento NCL.

Antes de comenzar a desarrollar programas interactivo se debe tener presente el tipo de contenido que se quiere mostrar así como la armonía de colores que se quieren utilizar, que a pesar que la norma brasileña no me restringe a la utilización de ciertas combinaciones en diferencia a las otras normas de TVD, para el espectador le tornaría molesto algunas combinaciones, y en el caso de personas que sufren de daltonismo les sería muy difícil distinguir entre dos colores de tonos similares. Una sugerencia de combinaciones así como del tipo de letra y

otros factores importantes a considerar para desarrollar un contenido interactivo más agradable y amigable para el usuario esta detallada en el Apéndice C del presente manual.

Aunque nuestro país todavía está muy atrasado respecto al tema de TVD en Latinoamérica respecto a los otros países que también adoptaron esta norma, es de mucha ayuda el poder acceder a las paginas oficiales de Ginga de cada país, ya que se puede interactuar con otros desarrolladores que en ocasiones con sus experiencias pueden ayudar en ciertos problemas que se presentan al introducirse en el tema, además se puede compartir y ayudar a otros a solucionarlos también, cumpliendo con el objetivo principal de Ginga que pretende atravez del código libre la evolución constante y la mejora del estándar, realizada no solo por los creadores del estándar si no también por personas particulares interesadas en el tema.

Nombre:	***onKeySelectionSetStartStopDelay***
Condición:	Pulsación de la tecla ***aTecla*** (función ***onSelection***).
Acción:	▪ Establece el valor de la propiedad relacionada con la función ***set*** como el si fuera el valor ***oValor*** pasado como parámetro en esta ligación, después de un retardo ***oRetardo***. ▪ Exhibe el nodo ligado a la función ***start***. ▪ Detiene el nodo conectado a la función ***stop***.
Código NCL:	<causalConnector id="onKeySelectionSetStartStopDelay"> <connectorParam name="oValor"/> <connectorParam name="aTecla"/> <connectorParam name="oRetardo"/> <simpleCondition role="onSelection" key="$aTecla" /> <compoundAction operator="par"> <simpleAction role="set" value="$oValor" delay="$oRetardo"/> <simpleAction role="start" delay="$oRetardo"/> <simpleAction role="stop" delay="$oRetardo"/> </compoundAction> </causalConnector>
Lectura:	Cuando el nodo ligado a la función ***onBegin*** comienza, se asigna un valor ***oValor*** a la propiedad conectada a la función ***set*** (después de un retardo **oRetardo**), inicia el nodo ligado a la función ***start*** (después de un retardo ***oRetardo***), y detiene el nodo conectado a la función ***stop*** (después de un retardo ***oRetardo***).
Observación:	Los enlaces que utilice este conector deben identificar, como parámetros adicionales para la conexión (***bindParam***), los siguientes valores: ▪ El código del botón que debe ser capturado, a través del parámetro aTecla. ▪ El valor que debe ser atribuido a la propiedad, a través del parámetro oValor. ▪ La duración del retardo para la atribución de valores a la propiedad, a través del parámetro oRetardo.

Cuadro 4.14: Definición del conector ***onKeySelectionSetStartStopDelay.***

Apéndice A

Entornos de desarrollo.

A.1. Instalacion de Ginga-NCL Virtual SetTopBox.

Para simular la Televisión donde será proyectada la aplicación interactiva se debe utilizar una máquina virtual llamada VMware que debe tener instalada una imagen llamada Ginga –NCL Virtual STB.

A.1.1. VMware

El VSTB consiste en en Fedora (Linux) modificado como ya se dijo en el capítulo 3. Para poder correrla en cualquier plataforma se debe virtualizar. Para esto se utilizó el software VMWare. Para instalar la máquina virtual VMware existen 2 opciones gratuitas: VMware Player y VMware server. Se deben descargar de la página web *http://www.vmware.com/products/player* se pueden utilizar cualquiera de las opciones, pero para hacer una instalación y ejecución más sencilla se optó por le versión VMware Player. La instalación de la máquina virtual es relativamente fácil, ya que la misma se la realiza en modo guiado como se puede ver en la Figura A.1. Es importante la memoria RAM que se selecciona ya que esta debe ser la misma memoria que tiene el STB real donde lo vamos a implementar.

Figura A.1: Pantalla de instalación de VMware

Una vez instalado el programa VMware como se muestra en la Figura A.2 se debe escoger la opción "Open Virtual Machine" para poder subir la imagen Ginga–NCL VSTB, ésta debe ser descargada previamente de la página web *http://www.gingancl.org.br/sites/gingancl.org.br/files/ferramentas/ubuntu-server10.10-ginga-v.0.12.4-i386.zip* de forma gratuita y en su versión más actual, con la maquina ya descargada se la descomprime y guarda en una ubicación del disco

rígido, esta dirección es a la que se tiene que referenciar en el momento de abrir la máquina virtual para que pueda ser ejecutada por el programa.

Se debe tener en cuenta que el adaptador de red para la conexión de la máquina virtual debe estar seleccionada como NAT, ya que en las otras opciones la tarjeta virtual presenta conflictos con la tarjeta de red física del host anfitrión, causando que la Ip del VSTB desaparezca en ocasiones haciendo imposible direccionar el mismo en el momento que se desea ejecutar una aplicación interactiva.

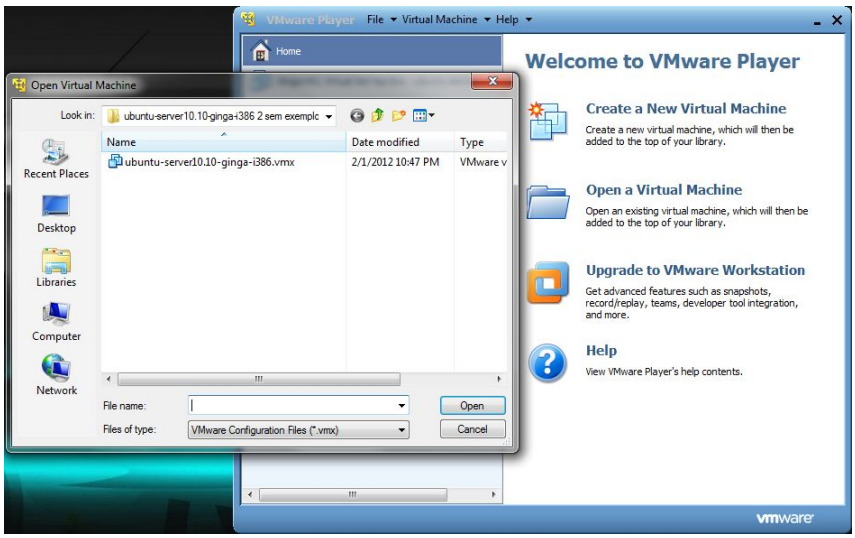


Figura A.2: Cargando la maquina virtual

Al abrir la máquina virtual, el proceso de arranque se inicia. Luego se debe elegir la opción con la resolución de la pantalla más adecuada para el tipo de contenido interactivo que se quiere desarrollar y el entorno de hardware en el que se va a utilizar (ver Figura A.3). Por lo tanto, el núcleo y todo el diseño de SeptTopBox serán cargados. La selección predeterminada al arrancar el VSTB es la última elegida por el usuario.

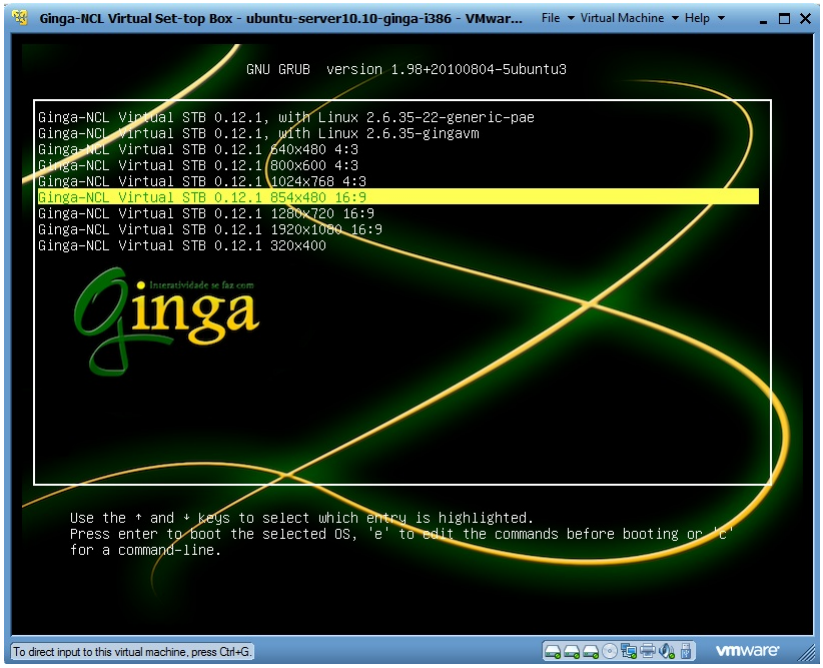


Figura A.3: Pantalla de inicio para la selección del tamaño de la pantalla del VSTB

Se deben espere unos segundos, hasta que la interfaz gráfica este inicializado completamente como muestra la Figura A.4. En el VSTB de la imagen escogió la pantalla de un tamaño de resolución de 16:9.

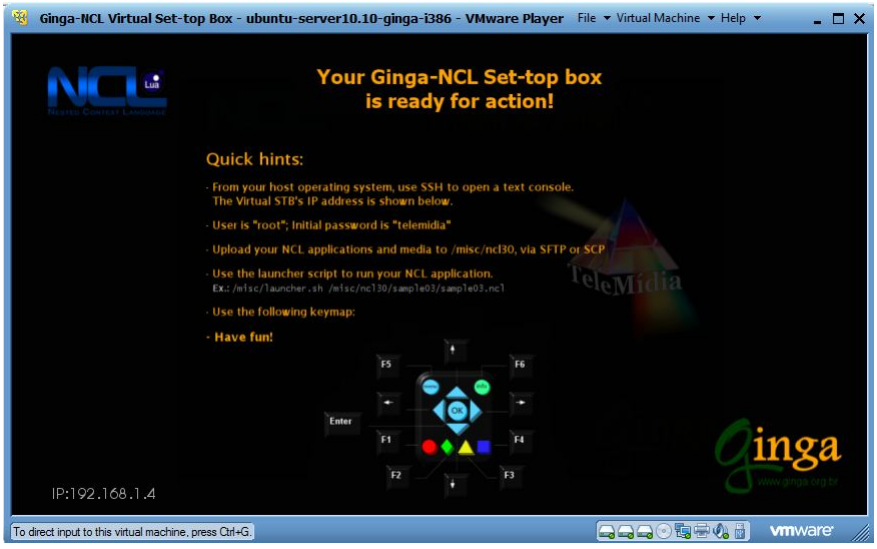


Figura A.4: Interfaz grafica inicializada

Como se observa en la imagen, el SetTopBox está listo para ser usado. La pantalla anterior ofrece importante información que es necesaria para direccionar la VSTB, sobre todo las que se muestran en la Tabla A.1:

Las aplicaciones desarrolladas en NCL, se deberán guardar en el directorio /misc/ncl30. Para esto se utilizará una conexión remota, en este manual la versión de Eclipse que se utilizó ofrece una conexión sin la necesidad de utilizar un software adicional, solamente realizando las configuraciones necesarias que se van a indicar en el siguiente punto para la configuración de la "Remote Ginga-NCL Play". El sistema también muestra el equivalente del Control Remoto al teclado

Nombre de Usuario	root
Contraseña	telemidia
Dirección IP por defecto asociado al Ginga-NCL Virtual SetTopBox	192.168.1.4

Cuadro A.1: Información del VSTB

del computador con los botones reconocidos para la interacción como se muestra en la Figura A.5.

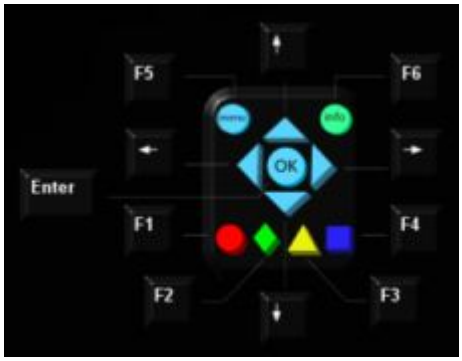


Figura A.5: Equivalencia del control remoto.

A.2. Instalacion de Eclipse.

Para la instalación del plugin NCL es aconsejable usar Eclipse versión 3.5.2 o superior, para la generación de este manual se utilizó la versión 3.7.1, que es la última que existe actualmente y que puede ser descargada de forma gratuita de la página oficial del software: *http://www.eclipse.org/downloads*.

Antes de instalar Eclipse es necesario instalar el JDK 1.6 o posterior (funciona también para JDK 1.5), de manera previa a la instalación del mismo. En los sistemas operativos Windows se recomienda chequear el contenido de la variable de entorno JAVA_HOME, dado que debe tener el path completo del directorio donde está instalado el JDK. En el caso de existir más de un JDK instalado en la computadora, es importante tener en cuenta tal aspecto, para saber a cuál JDK se está direccionando. Para instalar un JDK simplemente se debe ejecutar el instalador guiado.

Eclipse está desarrollado en Java y no es necesario efectuar la instalación, simplemente se debe descomprimir y ejecutar el archivo par para iniciar su uso.

A.2.1. Instalación del plugin NCL Eclipse

NCL Eclipse es un plugin que permite y estimula el desarrollo de aplicaciones en NCL. A partir de la versión 1,4 (la versión actual 1,5) se puso a disposición un sistema de instalación automática para Eclipse. Para realizar la instalación de NCL Eclipse se inicia el programa, luego se debe hacer clic en *Help -> Install New Software* como puede verse en la Figura A.6.

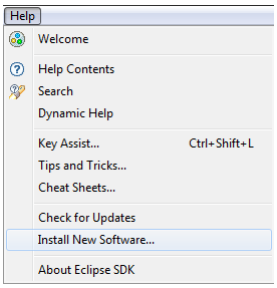


Figura A.6: Instalación del Plugin eclipse, Paso 1.

La pantalla de instalación se debe hacer clic en Agregar (Add), y se mostrara la Figura A.7.

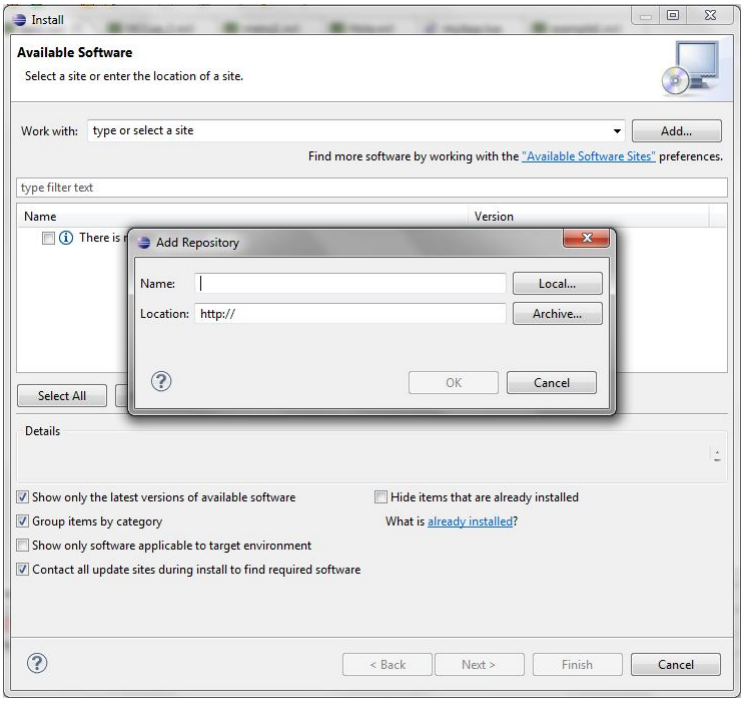


Figura A.7: Instalación del Plugin eclipse, Paso 2

En el cuadro de diálogo se coloca la información del nombre y la ubicación del sitio donde Eclipse buscara el plugin para la instalación. La información que se debe colocar es la siguiente:

- *Name:* NCL Eclipse
- *Location:* http://www.laws.deinf.ufma.br/ncleclipse/update/

Haga clic en *Aceptar* y Eclipse mostrara si hay actualizaciones en la dirección indicada. Para ver el Eclipse-NCL desmarcar la opción “Agrupar elementos por categoría". Seleccione Eclipse-NCL, haga clic en *Siguiente* y por último en *Finalizar*, se espera unos segundos a que termine la instalación.

Después de la instalación se le pedirá que reinicie Eclipse, se debe aceptar y esperar a que inicie de forma automática. Al iniciar de nuevo el plugin de Eclipse ya está instalado.

Para que Eclipse pueda simular el documento NCL se debe apuntar a la dirección de la máquina virtual de Ginga. En la barra de herramientas se escogerá

la opción *Windows*, luego la opción *Preferences*.

En la ventana *Preferences* se cambiara el *Hostname* a la dirección IP de la máquina virtual Ginga (ver Figura A.8), esto se debe realizar ya que VSTB funciona como un receptor con acceso remoto, y al configurar esta opción no es necesario la utilización de un software de acceso remoto de protocolo SSH. La dirección IP que se coloca es la que se muestra una vez que está corriendo el VSTB en la máquina virtual de VMware.

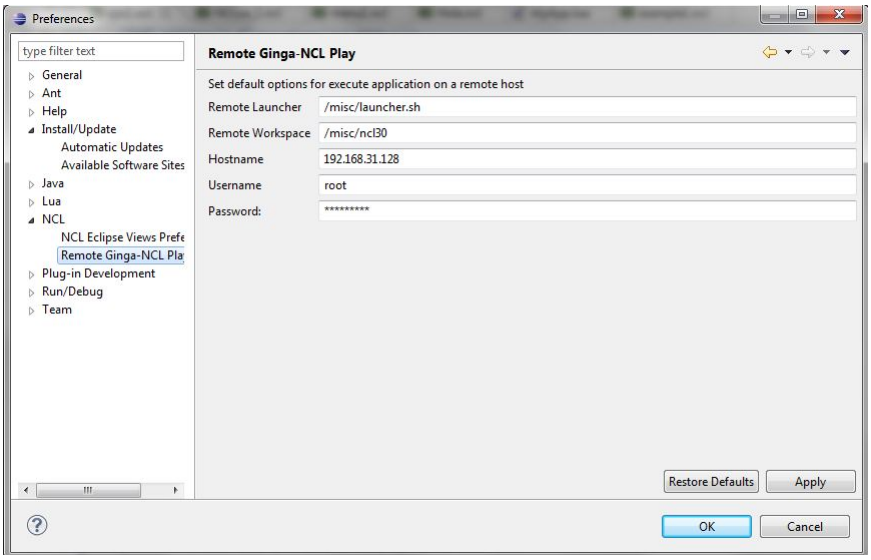


Figura A.8: Configuración de accesos remotos.

A.2.2. Instalación del plugin LuaEclipse.

La instalación sigue el estándar de Eclipse, y por lo tanto, es muy similar a lo que se realizó para instalar Eclipe-NCL. Sólo hay que sustituir el sitio de información:

- *Name:*Lua Eclipse
- *Location:* http://luaeclipse.luaforge.net/preview/update-ite/win32.win32.x86

Para que se pueda generar archivos de LuaEclipse se debe tener previamente instalado en el computador el programa Lua, ya que Eclipse utiliza el mismo para interpretar este código y debe ser direccionado desde la plataforma Eclipse. Este programa puede ser descargado de forma gratuita de la página oficial: *http://www.lua.org.*

Después de la instalación de LuaEclipse, se debe establecer el intérprete del lenguaje Lua para las aplicaciones. Para ello, se debe seleccionar en la barra de herramientas de Eclipse la opción *Windows*, luego se selecciona *Preferences*, y se escoge la categoría de *Lua -> Installed Interpreters*, como se muestra en la Figura A.9. En esta ventana se adiciona (Add) un nombre para el intérprete y por debajo se selecciona la ubicación del archivo ejecutable llamado lua.exe que se encuentra en la carpeta donde está instalado Lua. (normalmente C: \Program Files (x86) \Lua\ 5.1\lua.exe).

El uso de estas herramientas respecto a las otras que también se utilizan para generar contenido interactivo radica en la gran cantidad de soporte que estas

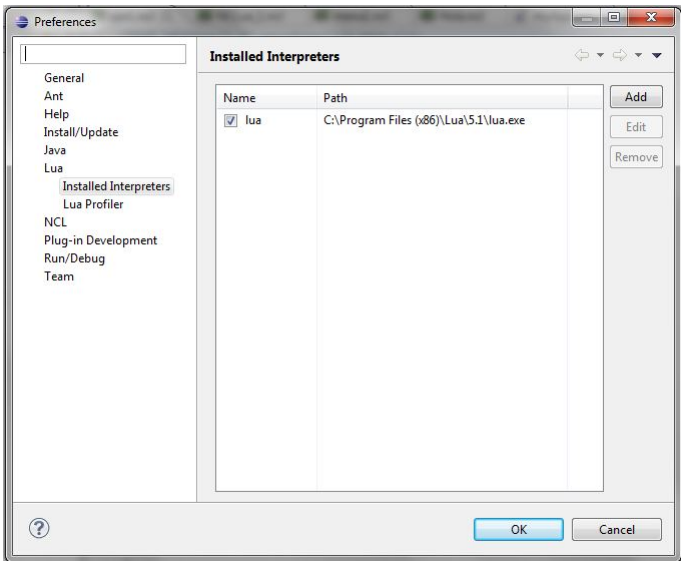


Figura A.9: Configuración del interpretador de Lua Eclipse.

brindan en el desarrollo, siendo las más potentes para la generación de dicho contenido, y las que más se han desarrollado desde su creación.

Apéndice B

Limitaciones técnicas y particularidades.

B.1. El tamaño del pixel.

La televisión muestra varias diferencias en relación a un PC, lo que no puede ser ignorado, ésta tiene una pantalla con resolución más baja, zona periférica sujeta a distorsiones, ofrece dispositivos bastante limitada con respecto a un PC, no ofrece rollover horizontal, presenta más lentitud en las respuestas. Todas estas características limitan considerablemente los recursos de la usabilidad en un proyecto para las aplicaciones de la televisión digital interactiva.

Ya que los monitores poseen pixeles cuadrados mientras que las pantallas de TV poseen pixeles rectangulares con 1.067 veces más de ancho que de alto (ver Figura B.1), se produce una distorsión al presentar el contenido en la pantalla de un televisor.

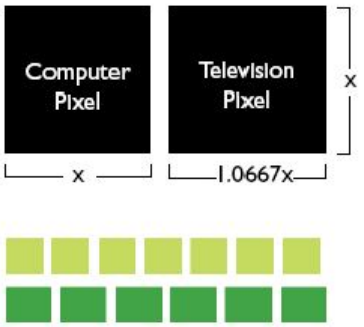


Figura B.1: Diferencias de pixeles entre la CP y la TV.

Para evitar esta distorsión, las imágenes creadas por computadora para TV (4:3), deben ser diseñadas con 768x576 px y luego reducidas horizontalmente a 720px.

B.2. Problemas en la pantalla de TV.

El problema principal es que las pantallas de TV son desarrolladas para mostrar imágenes en movimiento, por lo que estas presentas 3 tipos de problemas cuando se las utiliza para mostrar fotografías y gráficos estáticos.

1. Flicker (Parpadeo).
2. Bloom (Florecimiento).
3. Moiré.

Cuando se realiza contenido interactivo en un computador es fundamental testear frecuentemente lo que se está produciendo en los distintos tipos de pantallas de TV.

B.2.1. Flicker (Parpadeo).

Debido a que las pantallas de TV utilizan la exploración entrelazada impar para mostrar una imagen, el barrido produce líneas que se alternan a un ritmo de 50 veces por segundo, por esta razón cualquier pixel o línea de los mismos, que ocupe una sola línea de exploración, parpadeará, ver FiguraB.2.

Figura B.2: Problema de Flicker en la TV

Para evitar este parpadeo se recomienda que no se utilicen líneas que sean de un tamaño menor a 3 px, además al desarrollar el contenido interactivo no se deben usar tipografías con serif o terminaciones muy finas.

B.2.2. Bloom (Florecimiento).

Hace referencia a la distorsión en las líneas verticales que se producen en la pantalla, ver Figura B.3.

Figura B.3: Problema de Bloom en la TV

Para evitar el Bloom se recomienda evitar fuertes contrates de tono o luminosidad en líneas rectas verticales especialmente en superficies pequeñas.

B.2.3. Moiré.

Efecto de “tartán brillante" que ocurre cuando los patrones regulares, tales como las rejillas o puntos, se giran fuera de la vertical verdadera, es decir se aprecia como una imagen ondulante o una imagen superpuesta sobre otra, ver Figura B.4.

Figura B.4: Problema de Moiré en la TV.

Para evitar este efecto se recomienda no manejar patrones intrincados, utilizar regiones grandes, claramente definidas, de colores fríos y desaturados, este efecto no se aprecia cuando se utilizan líneas curvas, esta es una de las razones por las que un televisor está desarrollado para imágenes en movimiento ya que este disminuye la distorsión.

B.3. Normas para la realización de contenido interactivo.

La BBC de Londres, como uno de los precursores de la televisión digital en Europa y en el mundo, después de muchas pruebas y estudios, recomienda algunos formatos de fuentes para textos en esta modalidad y colores para el uso de elementos gráficos, imágenes y botones.

EL SKY de Brasil ha venido adoptando normas en sí mismas que no siguen las de la BBC, y permite al diseñador utilizar un patrón de usabilidad que no respeta las limitaciones para la generación de contenido que se quiere mostrar al usuario. Los tamaños de las fuentes y los colores no siguen el estándar de la BBC, aunque es importante considerar los estudios que ya existen para poder obtener una mejor armonía de colores en el desenvolvimiento de contenido interactivo y en cuanto al texto este pueda ser de fácil comprensión y lectura, pero sobre todo que la interactividad con el espectador sea muy sencilla, clara y agradable.

B.3.1. Los colores.

Las pantallas de televisión posee una gama más limitada de colores que los de los monitores como se muestran en la Tabla B.1. Para lograr la paridad en términos de intensidad de color, las imágenes deben ser atenuadas y desaturadas cuando se pasa del monitor a la pantalla del televisor.

Rojo: no pasar los 230	Blanco: no pasar los 16 y 230
Verde: no pasar los 230	Negro: no pasar los 16 y 230
Azul: no pasar los 230	

Cuadro B.1: Gama de colores de la TV.

En el desarrollo de apelaciones interactivas se recomienda evitar el uso de colores cálidos como rojos y naranjas, además de blancos puros y negros puros.

B.3.1.1. Connotaciones Locales del Color.

La asociación de colores no es la misma entre culturas. En los Estados Unidos los buzones de correo son azules; en Inglaterra son rojos; en Grecia son amarillos. En los Estados Unidos el rojo es asociado con peligro o detenerse, verde con algo

positivo o avanzar. Esta asociación de rojo y verde no existe en todo el mundo. La Tabla B.2 presenta algunas asociaciones culturales con el color. Los colores usados en la pantalla deben reflejar también las expectativas del color de quienes los miran.

	Rojo	Amarillo	Verde	Azul	Blanco
China	Felicidad	Abundancia, Poder	Dinastía, Cielo	Cielos, Nubes	Muerte, Pureza
Egipto	Muerte	Felicidad, Prosperidad	Fertilidad, Fuerza	Virtud, Fe	Gozo
Francia	Aristocracia	Temporal	Criminalidad	Verdad, Libertad, Paz	Neutralidad
India	Vida, Creatividad	Éxito	Prosperidad, Fertilidad		Muerte, Pureza
Japón	Ira, Peligro	Gracia, Nobleza	Futuro, Juventud, Energía	Villanía	Muerte
Estados Unidos	Peligro, Detenerse	Cobardía, Precaución	Seguridad, Avanzar	Masculinidad	Pureza

Cuadro B.2: Asociaciones culturales con el color

B.3.1.2. Combinaciones efectivas de fondos y primeros planos.

Un estudio realizado por Lalomia y Happ (1987) usando 16 colores de primer plano y 8 de fondo estableció la Tabla de efectividad basada en el tiempo de respuesta de los usuarios al identificar caracteres y sus gustos subjetivos. En la tabla B.3 se muestra las mejores y peores combinaciones evaluadas. Las que representan "Bueno" significa más de 80 % de efectividad. Las etiquetadas con Malo" bajo el 20 % de efectividad, las que sobre éste nivel se consideran regulares y las que están alrededor del 60 % no están etiquetadas.

Primer Plano.	Fondo							
	Negro	Azul	Verde	Cyan	Rojo	Magenta	Marrón	Blanco
Negro	X	Regular	Regular	Buena	Regular	Buena	Regular	Buena
Azul	Regular	X	Regular	Regular	Mala	Regular	Regular	Buena
Azul Intenso	Regular	Regular	Mala	Mala	Regular	Regular	Mala	Mala
Cyan	Buena	Regular	Mala	X	Regular	Regular	Mala	Regular
Cyan Intenso	Buena	Buena	Regular	Buena	Buena	Buena	Regular	Regular
Verde	Buena	Buena	X	Mala	Buena	Regular	Mala	Mala
Verde Intenso	Regular	Buena	Regular	Regular	Regular	Regular	Regular	Regular
Amarillo	Buena	Buena	Regular	Buena	Regular	Buena	Regular	Regular
Rojo	Regular	Regular	Mala	Regular	X	Mala	Mala	Regular
Rojo Intenso	Regular	Regular	Mala	Regular	Regular	Regular	Regular	Regular
Magenta	Regular	Regular	Mala	Regular	Mala	X	Mala	Regular
Magenta Intenso		Regular	Buena	Regular	Regular	Mala	Regular	Regular
Marrón	Regular	Regular	Mala	Regular	Regular	Mala	X	Regular
Gris	Regular	Mala	Regular	Regular	Mala	Regular	Mala	Regular
Blanco	Regular	Buena	Regular	Mala	Regular	Regular	Regular	X
Blanco Intenso	Buena	Regular	Buena	Buena	Regular	Regular	Regular	Regular

Cuadro B.3: Mejores y peores combinaciones de colres, Lalomia y Happ (1987)

Los resultados llegan a algunas conclusiones interesantes:

- La mayoría de buenas combinaciones tienen un color brillante o intenso como primer plano.
- La mayoría de malas combinaciones tienen bajo contraste.
- El color más benigno es el negro.
- El color menos usable es el marrón
- La mayor flexibilidad para elegir un color de primer plano es usando azul o negro de fondo.
- Marrón y verde son las peores elecciones de fondo.

Existe un estudio más completo realizado por Bailey (1989) considerando los resultados de Lalomia y Harp más algunos otros, el cual se muestra resumido en la Tabla B.4. Como conclusión: combinaciones de colores desaturados alcanza los mejores resultados.

Fondos	Primeros Planos Aceptables	
Negro	Cyan Obscuro, Amarillo Obscuro, Blanco Obscuro	Verde Claro, Cyan Claro, Magenta Claro, Amarillo Claro, Blanco Claro
Azul	Verde Obscuro, Amarillo Obscuro, Blanco Obscuro	Verde Claro, Cyan Claro, Amarillo Claro, Blanco Claro
Verde	Negro, Azul Obscuro	Amarillo Claro, Blanco Claro
Cyan	Negro, Azul Obscuro	Amarillo Claro, Blanco Claro
Rojo		Verde Claro, Cyan Claro, Amarillo Claro, Blanco Claro
Magenta	Negro, Azul Obscuro	Cyan Claro, Amarillo Claro, Blanco Claro
Amarillo	Negro, Azul Obscuro, Rojo Obscuro	
Blanco	Negro, Azul Obscuro	

Cuadro B.4: Estudio de combinaciones de colores realizado por Bailey (1989)

B.3.2. Recomendaciones para imágenes.

Las imágenes deben poseer una resolución mínima de 72dpi para que las imágenes mostradas sean de gran calidad, se debe limitar el uso de archivos PNGs solo para cuando se necesite transparencia en la imagen, ya que estos son mucho más pesados que los de formato JPGs..

Para que las imágenes se muestren de una mejor manera es recomendable que éstas estén en un formato JPGs y comprimirlas al 40 % para bajar su peso, además estos archivos no deben poseer la propiedad *Exif*.

B.3.3. Tamaños y formatos de pantallas.

Las aplicaciones interactivas que se están generando en la actualidad son para diferentes normas de televisión, ya que la mayoría de los televidentes tienen televisores tradicionales que poseen un espectro de formato 4:3, mientras que los televisores que están siendo desarrollados para TVD poseen otro tipo de formato 16:9 como se indica en la Figura B.5, por esta razón el contenido interactivo que se genere debe seguir estas consideración para una mejor exhibición del mismo.

- PAL N..... 720x576 px. (4:3)
- PAL Ancho ..... 1024x576 px. (16.9)
- HDTV 720p ..... 1280x720 px. (16:9)
- HDTV 1080i ..... 1920x1080 px. (16.9)

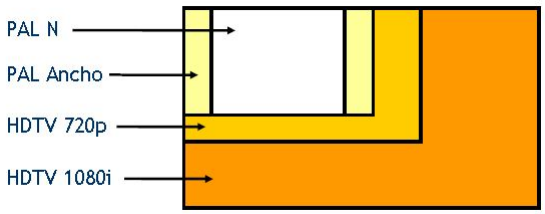


Figura B.5: Normas de TV

Cuando se presenta contenido con diferencias de formado pueden darse distorsiones como las que se pueden ver en las figuras. Se produce una distorsión (ver Figura ver Figura B.6(a)) "Petiso y gordo" cuando el contenido 4:3 es mostrado en 16:9, o una distorsión (ver Figura B.6 (b)) "Alto y flaco" cuando el contenido 16:9 es mostrado en 4:3.

(a) (b)

Figura B.6: Distorciones por diferencias de Formatos: (a)Petiso y gordo, (b)Alto y flaco.

Otro tipo de distorsión es la que se presenta por medio de bandas negras ya que los formatos del contenido son diferentes del de pantalla, éstas bandas pueden presentarse a los costados (ver Figura B.7 (a)), o en la parte superior e inferior de la imagen (ver Figura B.7 (b))

(a) (b)

Figura B.7: Distorciones con bandas: (a) Laterales, (b) Superior e Inferior.

Una recomendación para evitar estas distorsión por la presentación en diferentes formatos es que se corten píxeles para aplicaciones SDTV que van a ser presentados en una pantalla HDTV tal como se muestra en la Figura B.8.

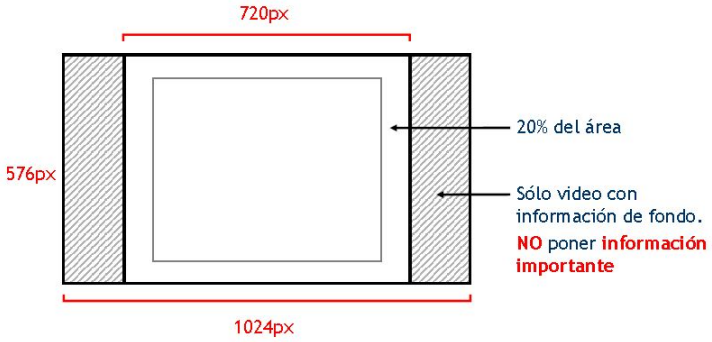


Figura B.8: Tamaño de pantallas de TV con diferentes formatos.

Existen otras técnicas como el Centro Court Out y Buzón como se muestra en la Figura B.9 . En la pantalla de 16:9 el ajuste se realiza en la relación 14:9.

Figura B.9: Relacion de una pantalla de 14:9 a una de 16:9

B.3.4. El texto.

El texto debe poseer fácil legibilidad, es decir debe estar en el punto óptimo donde el ojo reconoce los caracteres y puede leer un texto e interpretarlo de manera cómoda, rápida y sin esfuerzos.

B.3.4.1. Familias tipográficas.

Existen millones de familias tipográficas o fuentes, que se pueden catalogar en 3 grandes grupos:

- *Con Serif:* Time New Roman, Garamond.

- *Sin Serif:* Arial, Tahoma, Helvética, Vera, Trebuchet, Verdana, Tiresias.
- *Decorativas:* Caligráficas, góticas, de fantasías, etc.

Para la redacción de aplicaciones que presenten textos largos en pantalla preferir las Sin Serif, ya que este grupo posee tipos de letras sencillas, sin detalles lo que facilita la lectura del contenido al televidente.

B.3.4.2. Variables tipográficas.

Constituyen alfabetos alternativos dentro de la misma familia, que permiten obtener diferentes soluciones de color y ritmo de lectura, manteniendo un criterio de diseño que las emparenta entre sí. Existen 4 tipos de variables:

- Cuerpo o tamaño (ver FiguraB.10).

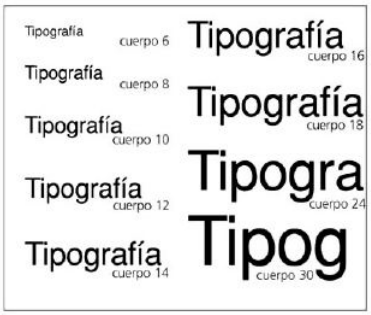


Figura B.10: Cuerpo o tamaño.

- Tono (ver Figura B.11).

Figura B.11: Tono

- Inclinación (ver Figura B.12).

Figura B.12: Inclinación.

- Proporción (ver Figura B.13)

Figura B.13: Proporción

B.3.4.3. Sentido de lectura.

Al realizar una aplicación interactiva se debe tener en cuenta que el sentido de lectura nuestro es de Izquierda a Derecha y de Arriba hacia Abajo, por esta razón el sentido a escoger debe ser el que facilite al televidente su comprensión, además de permitirle una lectura fluida. Existen 5 formas de alinear un texto:

- A izquierda
- A derecha
- Centrado
- Justificado
- Asimétrico

B.3.4.4. Ancho de columnas.

Equilibrar el ancho de las columnas justificadas con el tamaño de la fuente, ya que el espaciado entre letras, palabras y líneas también afecta al tipo y al color. Las palabras parecen de un tono más luminoso si las letras están más separadas.

B.3.4.5. Espacios en blanco.

Son tan importantes como los espacios escritos, ya que este espaciado favorecer la legibilidad cuando el contraste de color es escaso o cuando es necesario imprimir en color un gran fragmento textual. Se debe tener en cuenta que cuando se habla sobre legibilidad no se debe fijar en los factores que determinan el correcto espaciado entre letras (set) o palabras. Estos factores son el tipo utilizado, el cuerpo con el que se trabaja y el grosor de la letra. Un "set" uniforme proporciona una textura o color homogéneo del texto, lo cual genera una mayor legibilidad.

B.3.4.6. Recomendaciones.

El espectador NO LEE!!
Lo primordial al querer generar contenido interactivo es saber que los espectadores no están acostumbrados a leer textos estáticos y largos en la pantalla, ya que esto es bastante molesto e incómodo. El usuario escanea la pantalla con la vista, buscando puntos que llamen su atención, por esto se recomienda:

- Armar jerarquías de lectura con título, subtítulos, destacados para guiar el ojo del usuario hacia lo más importante
- Como máximo 50 palabras por pantalla / página
- Separar en párrafos
- Evitar fuentes menores a 14px de tamaño

Forma de los caracteres.
Evitar tipografías con detalles muy finos o de aristas muy delgadas, ya que pueden causar algunas distorsiones, por esto como ya se indicó la mejor familia para mostrar texto en una pantalla de TV son la sin serif.

NUNCA!! escribir textos largos todo en mayúsculas.
Las minúsculas generan un ritmo de lectura mucho más cómodo y legible para textos largos. Permiten diferenciar claramente cuando empieza y termina un párrafo por el uso de la mayúscula al comenzar la oración. Consumen mucho menos espacio que las mayúsculas y estéticamente son más armoniosas.

NO combinar más de 2 fuentes distintas en un mismo sitio.
Usar siempre la misma tipografía le da al sitio un sentido de unidad, para lograr ritmo, es decir darle a la presentación animación y que llame la atención del espectador se debe jugar con las variables tipográficas, las mejores son:

- Tamaño
- Tono (negrita, normal y light)
- El color

Generar buen contraste entre texto y fondo para lograr mayor legibilidad.
En la pantalla de TV la mayor legibilidad se logra con fondo negro y letras blancas (ver Figura B.14), pero se pueden usar combinaciones basándose en las Tablas B.3 y B.4, para una mejor armonización en el diseño de la aplicación interactiva.

Figura B.14: Mejor legibilidad

NO se deben usar colores complementarios para texto y fondo, ya que generan una vibración molesta que cansa mucho la vista (ver Figura B.15).

Figura B.15: Vibración molesta

El texto no debe ser colocado sobre texturas muy densas. Si se quiere poner un texto sobre una imagen, se debe ubicarlo en zonas planas y de color uniforme, ya que esto facilita su lectura.

Interlineado.

El texto presentado en una pantalla necesita más interlineado para facilitar su comprensión y lectura. Otro factor a considerar como ya se indicó en puntos anteriores es el espaciado entre letras, palabras, líneas y márgenes de párrafos, además de considerar el tipo de letra y el color del texto y del fondo. Las palabras parecen de un tono más luminoso si las letras están más separadas. Del mismo modo, si se incrementa el espaciado que hay entre palabras y líneas, el tipo parece adquirir un valor más brillante.

Apéndice C

Interactividad.

C.1. Audiencias.

Todos los programas de TV tienen solamente un numero de televidentes determinado, porque es imposible que un solo tipo de programación le sea agradable a todos los espectadores. Al realizar un programa que posea contenido interactivo se debe considerar que este puede ser visto tanto por personas jóvenes como de la tercera edad, que por lo general no poseen conocimientos del ámbito de interactividad, por esta razón las aplicaciones deben ser fáciles de usar para el usuario en general.

El espectador al observar un programa que posee una señal de TV analógica no realiza ninguna acción con el mismo ya que históricamente cumple un rol pasivo frente a al televisor. Por lo tanto la programación con contenido interactivo debe poseer una interacción y navegación simple clara y rápida de entender "Menos es Más".

C.2. Tipos de controles remotos.

Los controles remotos difieren enormemente entre ellos, ya que dependen del fabricante del televisor. El diseño y los nombres de los botones pueden variar o, incluso, pueden no estar, algunos modelos de controles remotos se muestran en la Figura C.1.

Figura C.1: Modelos de contorles remotos de TVD.

C.3. Principios de una nueva navegación.

1. Indicarle al espectador donde está parado,cómo llegó hasta ahí y dónde puede ir.
2. Proveer feedback cada vez que el usuario realiza una acción.
3. Enseñarle al espectador en pocos segundos como usar el servicio.
4. Usar metáforas y modelos mentales culturales más conocidos para diseñar la navegación.
5. Ser consistente y predecible en toda la aplicación.
6. Ayudar al usuario a tomar decisiones.
7. Proveer mensajes claros y concisos
8. Educar al usuario para que cada vez puedas usar aplicaciones más complejas.
9. Proveer salidas claras en cualquier punto de la navegación

En la Figura C.2 se muestra un ejemplo de utilización de los botones de interactividad de un control remoto.

Figura C.2: Ejemplo de Navegación de Futbol.

C.3.1. Instrucciones de Navegación.

El espectador, cuando busca instrucciones, escanea la pantalla y evita los texto largos ya que parecen de contenido aburrido. Para escribir instrucciones que llamen la atención del televidente se recomienda empezar la frase con la acción, además poner en mayúsculas la tecla que se debe presionar, un ejemplo que representa claramente esto es la tecla de salida.

- Para salir presione EXIT.
- Salida = EXIT.

C.3.2. Botones de colores.

La mayoría de los controles remotos tienen 4 botones de colores (ver Figura C.3) que se utilizan para realizar la interactividad con el espectador:

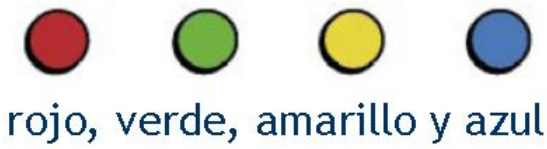


Figura C.3: Botones de colores para realizar la interactividad en TVD.

Estos botones pueden ser usados con diferentes propósitos dentro de las aplicaciones, como por ejemplo:

- opciones de navegación
- selección de opciones
- preguntas por si/no
- etc.

C.3.2.1. Recomendaciones.

Siempre se debe mantener el orden de los colores como aparecen en el control remoto, como se muestran en la Figura C.4.

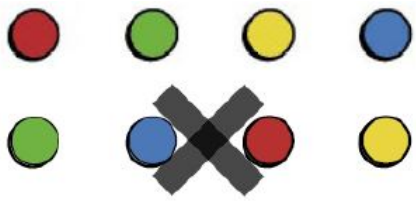


Figura C.4: Disposicion de los botones en el control remoto.

No siempre es considerado este estándar por los fabricantes ya que se pueden encontrarse en el mercado controles remotos con otro tipo de orden en la colocación de los botones (ver Figura C.5).

Figura C.5: Controles remotos que no respetan el estándar de orden de los botones.

Se debe tratar de mantener la disposición de los botones de colores en pantalla aunque haya menos de 4. También se debe mantener la consistencia de la funcionalidad para cada color en particular. Nunca asignar la misma función a 2 colores distintos, además nunca se recomienda usar los colores para funciones como subir / bajar / derecha / izquierda; ya que para esto están las flechas.

C.3.3. Los números.

Los números en un control remoto son una buena opción para la navegación de una aplicación, ya que el programador del contenido interactivo le puede asignar a estos funciones que el crea necesario pero siguiendo las recomendaciones que se citan a continuación.

C.3.3.1. Recomendaciones.

1. Son una buena opción para menúes con menos de 10 ítems
2. Si se necesitan cifras de 2 o 3 dígitos
   - proveer feedback
   - preveer que el usuario puede equivocarse y querer modificar lo que indicó con el control.
3. Mantener la consistencia en toda la aplicación.

C.3.4. Las flechas y el OK.

Los botones de las flechas en combinación con el "OK", son también una muy buena opción para la navegación, estas pueden ser utilizadas principalmente para desplazamiento y para selección (ver Figura C.6).

Figura C.6: Botones las flechas y OK

C.4. Los tiempos.

Durante todo un programa de TV la interacción puede variar según el bloque que el espectador está viendo. Debido a esto se recomienda calcular cuidadosamente los tiempos para la interacción, además de tener en cuenta el tiempo de descarga de Ginga, esto se puede ver en el ejemplo:

Programa de cocina

- Primer bloque: información sobre los cocineros de ese día (ver Figura C.7)

Figura C.7: Primer bloque

- Segundo bloque: recetas que están haciendo en ese momento (ver Figura C.8)

Figura C.8: Segundo bloque

Apéndice D

Base de conectores.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!– Generated by NCL Eclipse –>
<ncl id="connectorBase" xmlns="http://www.ncl.org.br/NCL3.0/EDTVProfile">
 <head>
  <connectorBase>

                                <!– OnBegin –>

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

    <causalConnector id="onBeginStop">
        <simpleCondition role="onBegin"/>
        <simpleAction role="stop"/>
    </causalConnector>

    <causalConnector id="onBeginPause">
        <simpleCondition role="onBegin"/>
        <simpleAction role="pause"/>
    </causalConnector>

    <causalConnector id="onBeginResume">
        <simpleCondition role="onBegin"/>
        <simpleAction role="resume"/>
    </causalConnector>

    <causalConnector id="onBeginSet">
        <connectorParam name="var"/>
        <simpleCondition role="onBegin"/>
        <simpleAction role="set" value="$var"/>
    </causalConnector>

                                <!– OnEnd –>

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

```
<causalConnector id="onEndStop">
    <simpleCondition role="onEnd"/>
    <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndPause">
    <simpleCondition role="onEnd"/>
    <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndResume">
    <simpleCondition role="onEnd"/>
    <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndSet">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>

                    <!- OnMouseSelection ->

<causalConnector id="onSelectionStart">
    <simpleCondition role="onSelection"/>
    <simpleAction role="start" />
</causalConnector>

<causalConnector id="onSelectionStop">
    <simpleCondition role="onSelection"/>
    <simpleAction role="stop" />
</causalConnector>

<causalConnector id="onSelectionPause">
    <simpleCondition role="onSelection"/>
    <simpleAction role="pause" />
</causalConnector>

<causalConnector id="onSelectionResume">
    <simpleCondition role="onSelection"/>
    <simpleAction role="resume" />
</causalConnector>

<causalConnector id="onSelectionSetVar">
    <connectorParam name="var" />
    <simpleCondition role="onSelection"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>

                    <!- OnKeySelection ->

<causalConnector id="onKeySelectionStart">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="start"/>
```

```
</causalConnector>

<causalConnector id="onKeySelectionStop">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onKeySelectionPause">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onKeySelectionResume">
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onKeySelectionSetVar">
    <connectorParam name="keyCode"/>
    <connectorParam name="var"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>

                    <!– OnBeginAttribution –>

<causalConnector id="onBeginAttributionStart">
    <simpleCondition role="onBeginAttribution"/>
    <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onBeginAttributionStop">
    <simpleCondition role="onBeginAttribution"/>
    <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onBeginAttributionPause">
    <simpleCondition role="onBeginAttribution"/>
    <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onBeginAttributionResume">
    <simpleCondition role="onBeginAttribution"/>
    <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onBeginAttributionSet">
    <connectorParam name="var"/>
    <simpleCondition role="onBeginAttribution"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>
```

```
                    <!– OnEndAttribution –>

<causalConnector id="onEndAttributionStart">
    <simpleCondition role="onEndAttribution"/>
    <simpleAction role="start"/>
</causalConnector>

<causalConnector id="onEndAttributionStop">
    <simpleCondition role="onEndAttribution"/>
    <simpleAction role="stop"/>
</causalConnector>

<causalConnector id="onEndAttributionPause">
    <simpleCondition role="onEndAttribution"/>
    <simpleAction role="pause"/>
</causalConnector>

<causalConnector id="onEndAttributionResume">
    <simpleCondition role="onEndAttribution"/>
    <simpleAction role="resume"/>
</causalConnector>

<causalConnector id="onEndAttributionSet">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <simpleAction role="set" value="$var"/>
</causalConnector>

                    <!– OnBegin multiple actions –>

<causalConnector id="onBeginStartStop">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="stop"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartPause">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartResume">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStartSet">
```

```
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopStart">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="start"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopPause">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopResume">
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginStopSet">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStart">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="start"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onBeginSetStop">
    <connectorParam name="var"/>
    <simpleCondition role="onBegin"/>
    <compoundAction operator="seq">
```

```
            <simpleAction role="set" value="$var"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginSetPause">
        <connectorParam name="var"/>
        <simpleCondition role="onBegin"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginSetResume">
        <connectorParam name="var"/>
        <simpleCondition role="onBegin"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

                    <!- OnEnd multiple actions ->

    <causalConnector id="onEndStartStop">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStartPause">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStartResume">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStartSet">
        <connectorParam name="var"/>
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
```

```
            <simpleAction role="start"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStopStart">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStopPause">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStopResume">
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndStopSet">
        <connectorParam name="var"/>
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndSetStart">
        <connectorParam name="var"/>
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndSetStop">
        <connectorParam name="var"/>
        <simpleCondition role="onEnd"/>
        <compoundAction operator="seq">
            <simpleAction role="stet" value="$var"/>
            <simpleAction role="stop"/>
        </compoundAction>
```

```
</causalConnector>

<causalConnector id="onEndSetPause">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndSetResume">
    <connectorParam name="var"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

            <!– OnMouseSelection multiple actions –>

<causalConnector id="onSelectionStartStop">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="stop"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartPause">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartResume">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStartSet">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="start"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
```

```
</causalConnector>

<causalConnector id="onSelectionStopStart">
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="start"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopPause">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopResume">
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionStopSet">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="set" value="$var"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStart">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="start"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetStop">
    <connectorParam name="var"/>
    <simpleCondition role="onSelection"/>
    <compoundAction operator="seq">
        <simpleAction role="stet" value="$var"/>
        <simpleAction role="stop"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onSelectionSetPause">
```

```
        <connectorParam name="var"/>
        <simpleCondition role="onSelection"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onSelectionSetResume">
        <connectorParam name="var"/>
        <simpleCondition role="onSelection"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

                <!– OnKeySelection multiple actions –>

    <causalConnector id="onKeySelectionStartStop">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStartPause">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStartResume">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStartSet">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="set" value="$var"/>
```

```
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStopStart">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStopPause">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStopResume">
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionStopSet">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionSetStart">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionSetStop">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
```

```
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionSetPause">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onKeySelectionSetResume">
        <connectorParam name="var"/>
        <connectorParam name="keyCode"/>
        <simpleCondition role="onSelection" key="$keyCode"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

                <!– OnBeginAttribution multiple actions –>

    <causalConnector id="onBeginAttributionStartStop">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStartPause">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStartResume">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStartSet">
```

```
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStopStart">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStopPause">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStopResume">
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionStopSet">
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionSetStart">
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionSetStop">
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
```

```
            <simpleAction role="set" value="$var"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionSetPause">
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onBeginAttributionSetResume">
        <connectorParam name="var"/>
        <simpleCondition role="onBeginAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

                <!- OnEndAttribution multiple actions ->

    <causalConnector id="onEndAttributionStartStop">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="stop"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStartPause">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStartResume">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="start"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStartSet">
        <connectorParam name="var"/>
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
```

```
            <simpleAction role="start"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStopStart">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStopPause">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="pause"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStopResume">
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="resume"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionStopSet">
        <connectorParam name="var"/>
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stop"/>
            <simpleAction role="set" value="$var"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionSetStart">
        <connectorParam name="var"/>
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="set" value="$var"/>
            <simpleAction role="start"/>
        </compoundAction>
    </causalConnector>

    <causalConnector id="onEndAttributionSetStop">
        <connectorParam name="var"/>
        <simpleCondition role="onEndAttribution"/>
        <compoundAction operator="seq">
            <simpleAction role="stet" value="$var"/>
            <simpleAction role="stop"/>
        </compoundAction>
```

```
</causalConnector>

<causalConnector id="onEndAttributionSetPause">
    <connectorParam name="var"/>
    <simpleCondition role="onEndAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="pause"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndAttributionSetResume">
    <connectorParam name="var"/>
    <simpleCondition role="onEndAttribution"/>
    <compoundAction operator="seq">
        <simpleAction role="set" value="$var"/>
        <simpleAction role="resume"/>
    </compoundAction>
</causalConnector>

                    <!–Miscellaneous–>

<causalConnector id="onKeySelectionStopResizePauseStart">
    <connectorParam name="width"/>
    <connectorParam name="height"/>
    <connectorParam name="left"/>
    <connectorParam name="top"/>
    <connectorParam name="keyCode"/>
    <simpleCondition role="onSelection" key="$keyCode"/>
    <compoundAction operator="seq">
        <simpleAction role="stop"/>
        <simpleAction role="setWidth" value="$width"/>
        <simpleAction role="setHeight" value="$height"/>
        <simpleAction role="setLeft" value="$left"/>
        <simpleAction role="setTop" value="$top"/>
        <simpleAction role="pause"/>
        <simpleAction role="start"/>
    </compoundAction>
</causalConnector>

<causalConnector id="onEndResizeResume">
    <connectorParam name="left"/>
    <connectorParam name="top"/>
    <connectorParam name="width"/>
    <connectorParam name="height"/>
    <simpleCondition role="onEnd"/>
    <compoundAction operator="seq">
        <simpleAction role="setLeft" value="$left"/>
        <simpleAction role="setTop" value="$top"/>
        <simpleAction role="setWidth" value="$width"/>
        <simpleAction role="setHeight" value="$height"/>
        <simpleAction role="resume"/>
```

```
            </compoundAction>
        </causalConnector>

        <causalConnector id="onKeySelectionStopSetPauseStart">
            <connectorParam name="bounds"/>
            <connectorParam name="keyCode"/>
            <simpleCondition role="onSelection" key="$keyCode"/>
            <compoundAction operator="seq">
                <simpleAction role="stop"/>
                <simpleAction role="set" value="$bounds"/>
                <simpleAction role="pause"/>
                <simpleAction role="start"/>
            </compoundAction>
        </causalConnector>
    </connectorBase>
  </head>
</ncl>
```

Bibliografía

[1] LIFIA. Laboratorio de investigación y formación en informática avanzada. 2011. URL: http://tvd.lifia.info.unlp.edu.ar.

[2] J. E. T. Altamirano. Diseño y desarrollo de una aplicación de contenidos interactivos para tv digital basada en el middleware ginga del sistema brasileño. 2010. URL: http://www3.espe.edu.ec:8700/bitstream/21000/2647/1/T-ESPE-029809.pdf.

[3] Fabiana Toledo V. DE Azevedo. Modelagem da programação não linear para televisão digital interativa. 2009.

[4] Instituto Superior Técnico. Televisão de alta definição. 2008. URL: http://www.img.lx.it.pt/-fp/cav/ano2007{_}2008/MERC/Trabalho{_}10/HDTV{_}54392{_}54429/HDTV{_}54392{_}54429/hdtv{_}html/hdtv.html.

[5] J. M. Villanueva and C. V. Diaz. Informe Preliminar: Estado del Arte de Receptores Set-Top-Box–Aplicaciones. 2010. URL: http://aat.inictel-uni.edu.pe/files/SET{_}TOP{_}BOX(Informe{_}de{_}Avance1).pdf.

[6] Pagina Oficial de MHP. Multimedia home platform. 2011. URL: http://www.mhp.org.

[7] R. Laiola and R. M. de Resende. Interactividad y sincronización en tv digital. 2010. URL: http://rodrigo.laiola.com.br/academic/laiola{_}monografia{_}interatividade.pdf.

[8] A. C. Jordi, B. David, B. Giuliano, D. G. Luca, and Z. Riccardo. Interactive Digital Terrestrial Television: The Interoperability Challenge in Brazil. *International Journal of Digital Multimedia Broadcasting*, 2009, 2009. URL: http://www.hindawi.com/journals/ijdmb/2009/579569/.

[9] INICTEL-UNI. Investigación del Estudio del Middleware GINGA y Guía de usuario del Middleware GINGA. 1, 2010. URL: http://www.ginga.org.pe/ginga/doc{_}template/pdf/Informe{_}Ginga{_}2010{_}AAT.pdf.

[10] Thiago Monteiro Proba. Un framework para desarrollo de aplicaciones declarativas en el sbtvd. 2010. URL: http://www.cin.ufpe.br/$\sim$tg/2009-2/tmp.pdf.

[11] ABNT NBR 15606-4. Televisão digital terrestre-Codificação de dados e especificações de transmissão para radiodifusão digital-Parte 4: Ginga-J-Ambiente para a execução de aplicações procedurais. *Asociación Brasilera de Normas Técnicas*, 4, 2010. URL: http://www.dtv.org.br.

[12] L. Quingaluisa and J. Torres. Estudio e investigación del middleware ginga j del estándar brasileño de televisión digital. caso práctico: Desarrollo de una aplicación interactiva aplicando metodología openup/basic como parte del proyecto espe-ginga. 2011. URL: `http://repositorio.espe.edu.ec/bitstream/21000/4748/1/T-ESPE-032865.pdf`.

[13] Comunidad Ginga Bolivia. Software libre. 2011. URL: `http://ginga.softwarelibre.org.bo`.

[14] C. S. S. Neto, L. F. G. Soares, R. F. Rodrigues, and S. D. J. Barbosa. Construindo Programas Audiovisuais Interativos Utilizando a NCL 3.0 Ea Ferramenta Composer, 2010. URL: `http://www.ncl.org.br`.

[15] Rafael Carvalho, Joel Ferreira, Jean Ribeiro Damasceno, Julia Veranda da Silva, and Debora Muchaluat. Introducción al Lenguaje NCL e Lua: Desenvolvimiento de Aplicaciones Interactivas para TV Digital. 2009. URL: `http://www.midiacom.uff.br/gtvd/files/apostila.pdf`.

[16] ABNT NBR 15606-2. Televisión digital terrestre-codificación de datos y especificaciones de transmisión para radiodifusión digital-parte 2: Ginga-ncl para receptores fijos y móviles-lenguaje de aplicación xml para codificación de aplicacione. *Asociación Brasilera de Normas Técnicas*, 2, 2007. URL: `http://www.dtv.org.br`.

[17] Ginga Brasil. Tv interactiva. 2011. URL: `http://www.ginga.org.br`.

[18] Laboratorio de Sistemas Multímedia. Telemidia. 2011. URL: `http://www.telemidia.puc-rio.br`.

[19] Ginga Ecuador. Comunidad ginga ecuador. 2011. URL: `http://comunidadgingaec.blogspot.com`.

[20] L. F. G. Soares, R. F. Rodrigues, and M. F. Moreno. Ginga-NCL: The declarative environment of the Brazilian digital TV system. *Journal of the Brazilian Computer Society*, 12(4), 2007. URL: `http://www.ncl.org.br`.

[21] Guilherme Natal Dalsotto. O advento da televisão digital interativa, o telespectador recebe e envia informações. 2010. URL: `http://ged.feevale.br/bibvirtual/Monografia/MonografiaGuilhermeDalsotto.pdf`.

[22] Pagina Oficial de Lua. Manual de referencia de lua 5.1. 20072008. URL: `http://www.lua.org/manual/5.1/es/manual.html`.

[23] Ginga Argentina. Comunidad ginga argentina. 2011. URL: `http://comunidad.ginga.org.ar`.

[24] Pagina Oficial de NCL. Nested context language. 2011. URL: `http://www.ncl.org.br`.

[25] Pablo. J. Souza. Júnior. Luacomp: Ferramenta de autoria de aplicações para tv digital. 2009.

[26] Ginga Brasil. Portal do software público brasileiro. 2011. URL: `http://www.softwarepublico.gov.br`.