Plugin para Composer NCL y Aplicación Interactiva para TV Digital Orientada a Educación Superior

Becerra F.*; Bernal I.*; Mejía D.*

** Departamento de Electrónica, Telecomunicaciones y Redes de la Información (DETRI)*
e-mail: fernandobecerracv@gmail.com, {ivan.bernal; david.mejia}@epn.edu.ec

Resumen: *Este artículo presenta un plugin desarrollado para Composer NCL que permite utilizar el canal de retorno para obtener información de un feed RSS. También presenta una aplicación interactiva desarrollada en NCL, la cual está orientada a informar a los televidentes sobre el programa de "Becas Universidades de Excelencia" que ofrece el Ecuador a sus ciudadanos. La aplicación interactiva obtiene datos desde un feed RSS, funcionalidad generada mediante el plugin desarrollado. Finalmente se presentan las pruebas realizadas tanto con el plugin como con la aplicación.*

Palabras clave: *Ginga, NCL, Canal de retorno, ISDB-Tb, feed RSS, Composer NCL, plugin.*

Abstract: *This article presents a plugin developed for NCL Composer that allows the use of the return channel to obtain information from an RSS feed. It also presents an interactive application developed in NCL, which is oriented to inform viewers about the program "Universities of Excellence Scholarship" offered by Ecuador to its citizen. The interactive application gets data from an RSS feed, generated using the plugin developed. Finally it presents some tests that were performed with both the plugin and the application.*

Keywords: *Ginga, NCL, Return Channel, ISDB-Tb, feed RSS, Composer NCL, plugin.*

1. INTRODUCCIÓN

El Ecuador escogió el estándar ISDB-Tb (*International System for Digital Broadcast, Terrestrial, Brazilian version*) para Televisión Digital. Son muchas las ventajas que ofrece este estándar, pero entre ellas se destaca la interactividad. Las aplicaciones interactivas permiten que el televidente interactúe con la programación, cambiando el paradigma tradicional del televidente a un rol más activo. Por otro lado, las aplicaciones interactivas presentan nuevos desafíos, debido a que, al ser una tecnología nueva, se necesita personal capacitado en esta temática. Por este motivo, la Escuela Politécnica Nacional considera importante el implementar programas que faciliten el desarrollo de aplicaciones interactivas. En este artículo se presenta un *plugin* para el programa Composer NCL (*Nested Context Language*), el cual permite obtener información de un *feed* RSS (*Really Simple Sindication*) de un servidor remoto, usando el canal de retorno. Este *plugin*, denominado consumidorRSS, fue desarrollado para ayudar a las personas que no tienen experiencia o conocimientos del lenguaje NCL o Lua, el mismo permitirá incluir en sus aplicaciones NCL el código necesario para obtener la información de un *feed* RSS.
Por otro lado, en los últimos años en el Ecuador se han realizado cambios significativos en la educación superior como por ejemplo, la categorización de las universidades; las becas para estudios de postgrados, que en número han aumentado significativamente; el sistema de admisión y nivelación (SSNA) que se encarga del ingreso a las universidades; entre otras. Siendo la televisión una indispensable fuente de información y entretenimiento en estos días, se ha desarrollado una aplicación interactiva empleando Composer NCL y el *plugin*, que permitirá difundir a los televidentes información sobre el programa de "Becas de Universidades de Excelencia". La aplicación interactiva presenta información sobre el programa como tal y permite, mediante el uso del canal de retorno, la presentación de datos obtenidos desde un *feeds* RSS.

2. HERRAMIENTAS

2.1 Ginga

Ginga [5] es el *middleware* abierto del estándar ISDB-Tb, que permite el desarrollo de aplicaciones interactivas con el uso de dos subsistemas: Ginga-J, empleado en aplicaciones procedimentales escritas en Java y Ginga-NCL empleado en las aplicaciones declarativas escritas en NCL. Las especificaciones de Ginga se definen en [1].

2.2 Lenguaje NCL

NCL [14] es un lenguaje desarrollado por el laboratorio TeleMídia de la PUC-Rio de Janeiro. Este lenguaje se basa en el lenguaje XML (*Extensible Markup Language*) y en el modelo conceptual NCM (*Nested Context Mode*) [9], ofrece una separación entre el contenido de medios de comunicación y la estructura de una aplicación. El ambiente declarativo en sí es muy limitado ya que no es extensible y no puede utilizar *scripts*, por eso utiliza un lenguaje procedimental llamado Lua. A continuación se presenta una breve descripción de los elementos NCL [14]:

Regiones (*Regions*): Son áreas de presentación de elementos multimedia. Se definen en el encabezado (*head*) del documento NCL, en la sección de las regiones base (*regionBase*). Todo documento NCL debe poseer al menos una región que defina la dimensión y las características de cómo se presentan uno o más nodos multimedia.

Descriptores (*Descriptors*): Son los elementos encargados de definir cómo será presentado un nodo multimedia, asociándolo a una región, están definidos en el encabezado (*head*) del archivo NCL, en la sección llamada base de descriptores (*descriptorBase*).

Contextos (*Contexts*): Se los utilizada para estructurar un documento hipermedia, los mismos que pueden ser anidados con el objetivo de reflejar la estructura del documento y mejorar su organización.

Puertos (*Port*): Forman puntos de interfaz de un contexto ofreciendo acceso externo al contenido del mismo, es decir, para que un enlace apunte a un nodo interno este debe poseer un puerto que lo dirija hacia dicho nodo.

Conectores (*Connectors*): Son elementos que definen uno o más roles para una condición de activación del enlace y para acciones que deben realizarse cuando el enlace este activo.

Enlaces (*Links*): Asocian nodos a través de conectores que definen la semántica de asociación entre ellos.

Enlazador (*Bind*): Este elemento permite asociar una interfaz de un objeto con un conector.

2.3 Lua

Lua [10] es un lenguaje de extensión, suficientemente compacto para usarse en diferentes plataformas, fue diseñado para apoyar a la programación procedimental con las instalaciones de descripción de datos. También ofrece un buen soporte para programación orientada a objetos, programación funcional y programación orientada a datos. Lua se lo emplea como un potente lenguaje ligero y se implementa en una biblioteca escrita en C. Lua no fue desarrollado para el entorno de televisión digital, sin embargo fue en este entorno donde demostró su fuerza y rendimiento. Lua permite dar un nivel superior de interactividad a las aplicaciones escritas en NCL.

2.4 Canal de retorno

El canal de retorno [12] es un medio de transmisión que permite la comunicación entre los STB (*set-top box*) o los televisores con soporte para ISDB-Tb con el proveedor del servicio de interactividad, permitiendo el envío y recepción de datos que el usuario intercambia entre la aplicación interactiva y dicho proveedor, como se observa en la Fig. 1. El canal de retorno puede ser implementado mediante varias tecnologías, cada una de las cuales presentan características que pueden ser apropiadas para determinadas zonas, regiones o países [2].

Usando el canal de retorno se puede realizar aplicaciones como: comercio electrónico, noticias interactivas, servicios de salud, educación, redes sociales, etc. Existen dos formas de interactividad remota: unidireccional y bidireccional. En la interactividad remota unidireccional el STB puede tan solo enviar o recibir datos; mientras que en la interactividad remota bidireccional, el STB puede enviar y recibir datos de forma simultánea.

2.5 Composer NCL

Composer NCL [3] es una herramienta desarrollada por el laboratorio TeleMídia. Con esta herramienta es posible construir programas audiovisuales interactivos con lenguaje NCL. Además, permite programar una aplicación interactiva sin poseer un conocimiento muy amplio sobre el lenguaje de programación NCL.

Esta herramienta está constituida por *plugins* que ayudan al diseño, programación y corrección de errores de una aplicación interactiva, permitiendo extender la funcionalidad de la misma mediante *plugins* desarrollados por terceros. Los *plugins* disponibles en Composer NCL son:

Textual View – Permite interacción con el documento NCL directamente.

Structural View - Permite interacción con la estructura lógica del documento NCL.

Layout View – Permite interacción con las regiones.

Properties View – Permite interacción con las propiedades de los elementos.

Outline View – Permite navegación entre los elementos del documento NCL.

Validator View – Permite validar el documento NCL.

2.6 Qt

Qt [3] es una biblioteca multiplataforma, es ampliamente usada para el desarrollo de aplicaciones con o sin interfaz gráfica de usuario (GUI). Es software libre y de código abierto que se distribuye bajo los términos de la licencia GNU (*Lesser General Public License*) y GNU (*General Public License*). Qt utiliza el lenguaje de programación C++ de forma nativa.

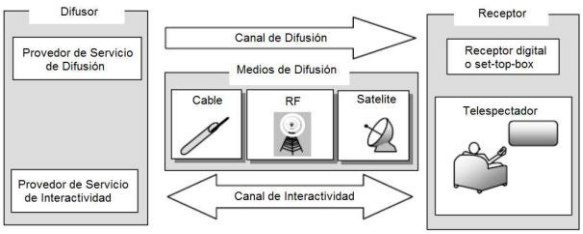

Figura 1. Esquema de funcionamiento de televisión digital

2.7 Set-top Box Virtual Ginga-NCL

Set-top Box Virtual Ginga-NCL [7] es un emulador creado por el laboratorio TeleMídia. Este emulador requiere de una máquina virtual como VMware para su funcionamiento. El emulador está basado en Ubuntu Server 10.10 e incluye la versión completa del *middleware* Ginga.

2.8 Ginga4Windows

Ginga4Windows [8] permite emplear el *middleware* Ginga en un sistema operativo Windows, sin la necesidad de una máquina virtual o de un sistema operativo Linux. Este programa fue desarrollado por el laboratorio TeleMídia, es de código abierto y gratuito.

2.9 EiTV Developer Box

EiTV *Developer Box* [6] es un STB híbrido ISDB-Tb e IPTV (*Internet Protocol Television*) enfocado en desarrolladores de aplicativos Ginga, compatible con Ginga-NCL/Lua y Ginga-J/JavaDTV.

2.10 Escenario de pruebas

Para las pruebas se emplearon dos escenarios: el primero usando emuladores de STB. Para el primer escenario se emplearon los programas Ginga4Windows y *Set-top Box Virtual* Ginga-NCL. El segundo escenario de pruebas se realizó con equipos reales, en un esquema como el presentado en la Fig. 2, empleando una televisión, un STB (EiTV *Developer Box*), y un servidor remoto con *feed* RSS.
Para el desarrollo y la realización de pruebas del *plugin* se utilizó en un inicio Qt *Creator* [13] para la programación y el diseño gráfico del *plugin*, y para las pruebas se utilizó una máquina virtual con Composer NCL.

3. PLUGIN consumidorRSS

Para el *plugin* se desarrolló un archivo principal llamado **main.lua** el cual utiliza varias clases. En la Fig. 3 se presentan los archivos con código Lua que emplea el *plugin*.
La clase **tcp** [16] permite crear una conexión TCP entre Ginga y un servicio alojado en un servidor remoto. La operación de esta clase se basa en el protocolo TCP, el cual se compone de tres etapas: el establecimiento de la conexión, mediante la función **connect(host, port)**; la transferencia de datos mediante las funciones **send()** y **receive()**; y el cierre de la conexión mediante **disconnect()**.

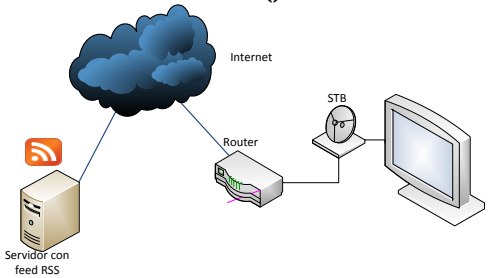

Figura 2. Escenario de pruebas

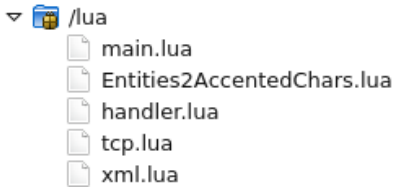

Figura 3. Archivos con código Lua del *plugin* consumidorRSS

En la Fig. 4 se puede observar la función **tcp.execute**, parte del código utilizado en **main.lua**. La función **tcp.execute** realiza las tres etapas de TCP. Mediante la función **tcp.conect(host, 80)** se establece la conexión con el servidor remoto mediante el protocolo TCP, el servidor estará determinado por el par nombre de equipo/puerto, en este caso el nombre del servidor es "yachay.com" y el puerto 80. Con la variable **request** se genera un paquete de datos empleando el protocolo HTTP (*HyperText Transfer Protocol*), el cual contendrá el comando que debe realizarse en el sitio remoto así como el URL (*Uniform Resource Locator*) del recurso que se solicita. En este caso, el paquete de datos define el comando GET con el cual se solicita el recurso especificado en la variable **url** y se indica la versión del protocolo (HTTP/1.0). Mediante **tcp.send(request)** se envía la solicitud HTTP al servidor remoto. El servidor responderá con el recurso solicitado y para obtener dicho recurso se emplea el método **tcp.receive("*a")**, el parámetro "*a" permite obtener todos los datos sin tener que realizar llamadas sucesivas a **tcp.receive()**. La variable **result** contendrá la información del *feed* RSS en formato XML. Por último se cierra la conexión TCP mediante **tcp.desconnect()**.
La clase **tcp** se encarga de realizar la conexión hacia el *feed* RSS y de obtener toda la información en formato XML, luego de lo cual la información deberá procesarse.

3.1 Tratamiento de datos en formato XML con Lua

Una vez que se obtiene la información en formato XML, debe ser procesada para su presentación al usuario. Para simplificar las tareas de obtención y presentación de información se dispone de dos clases: **handler** y **xml** [11]. En la Fig. 5 se observa parte del código del archivo **main.lua** que permite el procesamiento del contenido XML.

```
tcp.execute(
      function ()
         local host = "www.yachay.ec"
         local path = "/feed"
         tcp.connect(host, 80)
         local url = "http://"..host..path
         local request = "GET "..url.." HTTP/1.0\n"
         request = request .. "Host: "..host.."\n\n"
         tcp.send(request)
         local result = tcp.receive("*a")
         tcp.disconnect()
      end
   )
```

Figura 4. Función tcp.execute definida en main.lua

```
dofile("xml.lua")
dofile("handler.lua")

local tablafeed = simpleTreeHandler()
local feedEPN = "feedRSSEPN.xml"
local f, e = io.open(feedEPN, "r")
local xmltext = f:read("*a")
local xmlparser = xmlParser(tablafeed)

xmlparser:parse(xmltext)
```

Figura 5. Parte del código para procesar el contenido XML definido en main.lua

La clase **handler** ofrece varias funciones para crear estructuras en formato XML. La función más importante es **simpleTreeHandler**, la cual básicamente se comporta como un controlador simplificado que intenta generar una estructura basada en una tabla. La estructura de árbol XML se asigna automáticamente a una estructura de tabla recursiva con nombres de nodos como llaves y elementos secundarios, ya sean como tablas de valores o como cadenas de texto. La clase **xml** proporciona un analizador de secuencias XML, sus datos pasan a la instancia del analizador a través del método **parse**.
La clase **Entities2AccentedChars** permite cambiar la codificación de los resultados, los cuales pueden estar en formato ASCII y deberán ser transformados al formato ISO-8859-1 para su presentación correcta, sobre todo los caracteres especiales del idioma español.

3.2 Interfaz gráfica del plugin consumidorRSS

Composer NCL fue desarrollado en Qt, por lo tanto el *plugin* debe ser escrito en este lenguaje. Para el diseño de la interfaz gráfica se debe tener en cuenta la información necesaria para los elementos Lua. Presionando el botón "Generar", se generarán todos los códigos (NCL y Lua) y se realizarán sus validaciones respectivas. La interfaz gráfica debe ser fácil de usar, ya que las personas que la utilizarán no necesariamente serán programadores.
La interfaz permite ingresar la siguiente información: "Página del *feed* RSS", "Fuente del Texto", "Color del Texto", "Tamaño del Texto" y "Nombre del Archivo". Estos datos se utilizan para la conFig.ción de los archivos Lua. En "Página del feed RSS" se debe ingresar el URL del *feed* RSS. En "Fuente del Texto" se define la fuente que se empleará para la visualización de los resultados, las posibles fuentes son: *Times*, *Courier* y *Helvetica*. "Color del Texto" permite definir el color del texto para su presentación, pudiendo escoger de entre las siguientes opciones: *white*, *aqua*, *lime*, *yellow*, *red*, *fuchsia*, *purple*, *maroon*, *blue*, *navy*, *teal*, *green*, *olive*, *silver*, *gray* y *black*. El conjunto de opciones de "Color del Texto" y de "Fuente del Texto" están de acuerdo a [1]. Además en "Tamaño del texto" se define el tamaño de los datos a ser presentados, el rango de valores permitidos está entre 1 a 40. Finalmente, en "Nombre del Archivo" se debe ingresar el nombre del archivo Lua principal, por ejemplo: main.lua.

Una vez integrada la aplicación con Composer NCL se determinaron nuevos requerimientos. Estas nuevas funcionalidades nacen debido a que el *plugin* debe crear automáticamente los diferentes elementos que interactúan con la aplicación interactiva Ginga-NCL, como: media, área, descriptores y conectores base, todos ellos con sus respectivas propiedades. En la Fig. 6 se puede observar la versión final de la interfaz de usuario del *plugin*.
En el grupo "Dimensiones de la Región" se agregó un **QComboBox** que permite escoger dos opciones: Porcentaje y Píxeles. Si se escoge Porcentaje en los **QLineEdit** de las propiedades *top*, *left*, *heigth* y *width* se pueden ingresar valores decimales. Si por el contrario, se escoge Píxeles solo se pueden ingresar valores enteros. En el grupo "Conectores" se utilizó también un **QComboBox** en el que se agregaron los conectores que pertenecen a "defaultConnBase" que está integrado en Composer NCL. Además se programó que el primer elemento que se presente en este **QComboBox** sea "InicioAutomatico", el cual, al ser seleccionado agrega un *port* en el *body* del documento NCL que está asociado al elemento media que se generará cuando entre en funcionamiento el *plugin*, lo que implica que se inicie de forma automática sin la necesidad de otro elemento media.

3.3 Integración del plugin con Composer NCL

Un *plugin* es un segmento de programa que puede ser distribuido de forma separada o integrado en la aplicación principal con el objetivo de ampliar sus funcionalidades. Composer NCL [4] se basa en dos interfaces principales escritas en C++ para el uso de *plugins*, las interfaces son: **IPluginFactory** y **IPlugin**.

3.3.1 IPluginFactory

IPluginFactory es responsable de crear nuevas instancias de **IPlugin**, y entrega información global sobre el *plugin* creado, como su fabricante, la versión, etc.

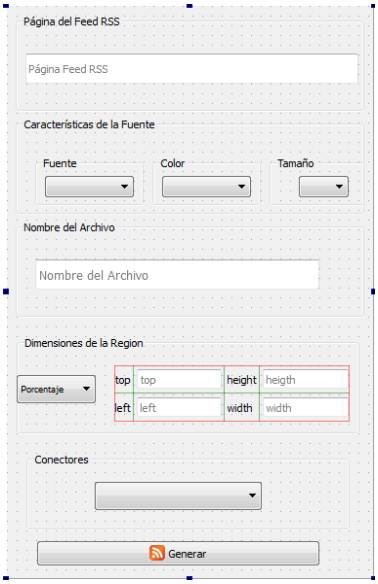

Figura 6. Interfaz de usuario del *plugin*

Cada vez que se inicia un nuevo documento, Composer NCL trata de crear una nueva instancia del *plugin* mediante la llamada al método de **IPluginFactory** denominado **createPluginInstance()**.

3.3.2 IPlugin

IPlugin es el responsable de comunicarse directamente con el núcleo de Composer NCL y de interactuar con los demás *plugins*. Cuando se crea un documento de Composer NCL, se requiere una nueva instancia del *plugin*. **IPlugin** está vinculado con la instancia del documento y será llamado cada vez que haya cambios en cualquier *plugin* de Composer NCL. De esta forma, **IPlugin** también es capaz de exigir cambios en el documento.

3.3.3 Signals y Slots

Qt define una forma dinámica de comunicar eventos con los cambios de estado que estos pueden provocar y las reacciones de los mismos denominada *Signals* y *Slots* [15]. Lo original de este mecanismo es que los objetos que se comunican dinámicamente, no requieren conocerse mutuamente. Para ello **QObject**, la clase base de gran parte de los objetos que conforman Qt incorpora un método llamado **connect()**, que se encarga de establecer dicha comunicación entre dos objetos. Todo objeto derivado de **QObject** puede poseer dos tipos de funciones propias especiales: *signals* y *slots*. *Signals* son funciones que permiten emitir una señal cuando hay algún cambio de estado en el objeto al que pertenecen. *Slots* son funciones que finalizan la conexión y que ejecutan una serie de acciones una vez que reciben el mensaje de la señal. Una señal puede conectarse a más de un *slot* para llevar a cabo diferentes tipos de actividades. Igualmente diferentes señales pueden conectarse con un mismo *slot* que posee una actividad que puede ser desplegada de varias formas.

3.3.4 Núcleo de Composer NCL

Para la integración del *plugin* con Composer NCL y con los demás *plugins*, se debe utilizar el módulo "Composer-core" [3]. Este módulo contiene clases y funciones que ayudan con la integración del *plugin*. Para poder integrar el *plugin* se debe utilizar *signals* y *slots*, *signals* se utiliza para la comunicación con los otros *plugins* y *slots* se utiliza para la comunicación con el núcleo de Composer NCL.
Para agregar elementos al documento NCL se debe utilizar el método **Composer::extension::IPlugin::addEntity()**, el cual acepta los siguientes parámetros: **QString type**, con el que se especifica el tipo de elemento NCL que se desea agregar en el documento NCL; **QString parentEntityId**, para especificar el identificador único del elemento padre de la entidad que se desea añadir; **QMap<QString, QString> &atts**, con el que se especifican los parámetros de la entidad. Para obtener la información de un *plugin* se utiliza el método **Composer::core::model::Project::getPluginData()**, el cual define un único parámetro **QString pluginId**, en el que se especifica el nombre del *plugin* del que se desea obtener los datos. Por ejemplo, si se desea obtener la información del *plugin* denominado *Textual View*, se deberá pasar en el argumento la siguiente cadena de texto: "br.puc-rio.telemidia.NCLTextualView". Estos métodos deben ser llamados cuando se requiera agregar un elemento o proveer información del *plugin* mediante el uso del mecanismo de *signal* y *slots*.

3.3.4 Archivo NCL generado por el plugin

Cuando se ejecute el *plugin*, se generará un archivo NCL el cual contendrá un elemento media asociado al *body* de la estructura NCL de dicho archivo, además de esto el *plugin* generará un elemento área, con las dimensiones establecidas en la interfaz de usuario. Esta área se utilizará con el *descriptor* del elemento media para especificar en qué parte de la pantalla se presentará la información obtenido del *plugin*. Por último se escoge cuándo se presentará la información del *feed* RSS en la aplicación, para lo cual se deben agregar los conectores en el *body* del archivo NCL. Es posible que varios de los conectores requieran de otros elementos media para interactuar entre ellos, por lo que no es posible automatizar todos los conectores, por lo que el usuario deberá especificar ciertas propiedades de forma manual. El *plugin* verifico la exista del archivo con los conectores base, en caso de que no exista, el *plugin* creará el archivo con dichos conectores. La lista de conectores base está filtrada para que solo se presenten los conectores que tengan en sus elementos la palabra *start*, ya que con este elemento se puede indicar cuándo se deberá presentar (iniciar) la información del *feed* RSS.

3.3.5 Resultado del plugin

El *plugin* fue integrado exitosamente en Composer NCL como se observa en la Fig. 7. En la Fig. 8 se presenta una aplicación NCL que muestra la información obtenida del *feed* RSS de la Escuela Politecnica Nacional.

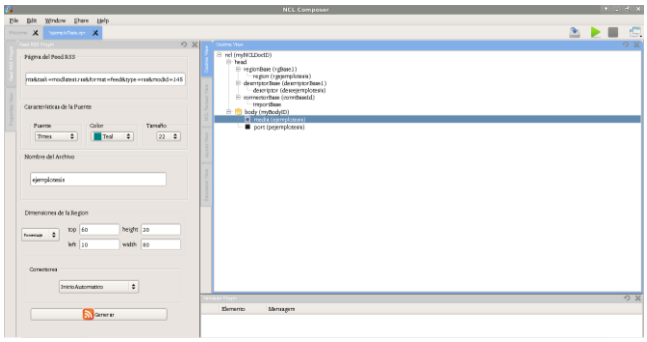

Figura 7. Integración del *plugin* consumidorRSS con Composer NCL

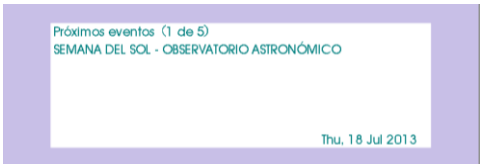

Figura 8. Aplicación diseñada con el *plugin*

4. APLICACIÓN INTERACTIVA "BECAS UNIVERSIDADES DE EXCELENCIA"

4.1 Análisis de requerimientos

Para la realización de la aplicación se tomó en cuenta una serie de preguntas que usualmente las personas realizan sobre las becas, estas son: ¿Qué estudiar?, ¿Qué rubros cubre la beca?, ¿Las becas son rembolsables?, Requisitos, ¿Cómo postular? y ¿Qué documentos se requieren para postular (documentos habilitantes)? La aplicación busca responder a estas inquietudes de forma breve y concisa, debido que hay que considerar que el televidente usualmente mira la televisión mientras descansa, por lo que no debe sobrecargársele de información. También se tomó en cuenta que el programa no debe bloquear la programación habitual, por lo que se redimensionará el video guardando la relación de aspecto de 16:9. Por otro lado, es importante la distribución de colores pues brinda un nivel estético al programa que debe llamar la atención del televidente.

4.2 Desarrollo

En la Fig. 9 se presenta la distribución en la pantalla, generada por el *plugin Layout View* con los elementos media que intervienen en la aplicación, estos elementos son imágenes con la información que se desea presentar.
El archivo NCL que contiene el código de la aplicación dispone de un *body* con un *port* que está asociado con un elemento media llamado **video,** el cual permitirá presentar la señal de la programación recibida por el STB, para esto se necesita establecer en el elemento mencionado la propiedad **src** con la cadena "sbtvd-ts://0". Para presentar el icono de interactividad en la aplicación se agregó un conector **onBeginStart_delay**. Este conector permite asociar la presentación de un elemento en particular con un retardo (Start_delay), con la presentación de otro elemento (onBegin), para lo cual utilizará dos elementos *bind*: **onBegin** y **start.** Mediante **onBegin** que estará asociado con un elemento media, se iniciará el conector al ser presentado dicho elemento, produciendo las acciones definidas por el segundo elemento *bind*. La acción definida mediante **start**, que también estará asociada con un elemento media, permite indicar que elemento iniciará (o se presentará en pantalla).

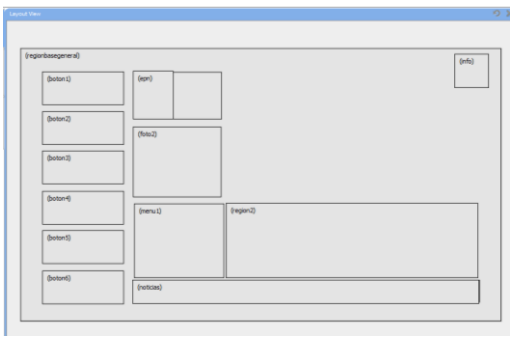

Figura 9. Distribución de los elementos en pantalla mediante el *plugin Layout View*

Además el elemento **start** tiene un parámetro *bindParam* que permite especificar el tiempo de retardo, es decir permite definir que el elemento media asociado al **start** se presente luego de un tiempo especificado posterior a la presentación del elemento media asociado a **onBegin**. En la Fig. 10 se observa el código de **onBeginStart_delay** implementado.
Como se mencionó antes, la aplicación debe redimensionar el video. El redimensionamiento se lo consigue definiendo la propiedad **bounds** (límites) del elemento media. Para indicar cuándo se debe dimensionar el video, se utilizó el conector **onKeySelectionSet**. Este conector permite indicar que en el momento en que se presenta un elemento media y se presiona un botón específico del control remoto (onKeySelection), el conector se activa y presenta otro elemento media, el cual puede cambiar su posición y su tamaño (set). Este conector contiene dos elementos *bind*: **onSelection** y **set**. Mediante **onSelection** se indican las condiciones que deben cumplirse para iniciar este conector, siendo estas: el elemento media que debe estar seleccionado (especificado mediante el elemento **component**) y el botón que debe presionarse (especificada mediante el parámetro *bindParam*). Mediante **set** se indica que se cambiará la posición o el tamaño del elemento media definido por **component**, para esto se emplea la interfaz **bounds** del elemento media, y mediante el parámetro *bindParam***,** se especifica su nueva ubicación y sus dimensiones usando una cadena con cuatro valores: "izquierda, parte superior, ancho, altura" especificados como porcentajes o como píxeles. Al presionar el botón **OK**, la aplicación procederá a cambiar la posición y el tamaño del elemento **video**. En la Fig. 11 se observa el código del conector **onKeySelecionSet** implementado.
Para mejorar la organización de los elementos que van a interactuar en la aplicación se creó un *context* llamado **ctxBecas**. Para conectar el elemento **mInteractividad** con **ctxBecas** se utilizó el conector **onKeySelectionStopStart**. Este conector permite que en el momento en que se presenta un elemento media y se presiona un botón del control remoto (onKeySelection), el conector se activa y oculta un elemento media (stop) y después presenta otro elemento media (start).

```
<link id="link1" xconnector="conn#onBeginStart_delay">
    <bind component="video" role="onBegin">
    </bind>
    <bind component="mInteractividad" role="start">
        <bindParam name="delay" value="4s">
        </bindParam>
    </bind>
</link>
```

Figura 10. Código del conector onBeginStart_delay

```
<link id="link2" xconnector="conn#onKeySelectionSet">
    <bind component="mInteractividad" role="onSelection">
        <bindParam name="keyCode" value="ENTER">
        </bindParam>
    </bind>
    <bind component="video" interface="bounds" role="set">
        <bindParam name="var" value="42.25%,1.75%,53%,53%">
        </bindParam>
    </bind>
</link>
```

Figura 11. Código del conector onKeySelecionSet

En la Fig. 12 se observa el código del conector **onKeySelecionStopStart** implementado, se puede apreciar que el ícono de interactividad se ocultará y se iniciará el *context* cuando el usuario presione la tecla **OK**.

Al iniciar el *context* **ctxBecas** aparecen los elementos media que permiten navegar por la aplicación y que contienen contenido informativo sobre la misma, así como también el *script* Lua, el cual iniciará su funcionamiento para obtener la información del *feed RSS*. La página principal que contiene estos elementos se presenta en la Fig. 13.

Para desplazarse dentro de estos elementos se utilizan *descriptors*. Cada *descriptor* dispone de los siguientes atributos: **focusIndex**, para establecer el enfoque que va a tener el elemento al presionar los cursores de desplazamiento del control remoto; **moveDown** y **moveUp**, para realizar el desplazamiento hacia abajo o arriba respectivamente. Con estos elementos se realizó un menú vertical el cual contiene seis elementos media, que tienen en su atributo **focusIndex** los valores de "1" a "6", respectivamente, y en los atributos **moveDown** y **moveUp** se establece a que elemento debe saltar el foco al presionar los cursores **DOWN** y **UP** del control remoto.

Para que al seleccionar un elemento del menú se despliegue la información relacionada a este, se utilizó el conector **onSelectionStopNStartN**, este conector permite ingresar más de un elemento *bind* tanto para ocultar (Stop**)** o para activar (Start**)**. En la Fig. 14 se observa el código del conector **onSelectionStopNStartN** implementado, en la cual se puede apreciar que al seleccionar un elemento (**mboton6**) del menú y presionar la tecla **OK**, se ocultará la información (**mLibreta** y **mLibreta2**) y se iniciarán otros elementos media (**mLibreta6** y **mMenu2**) con su información respectiva. Se utilizó este conector ya que la información relacionada a cada menú es amplia y por este motivo se la separó en varias imágenes.

```
<link id="link3" xconnector="conn#onKeySelectionStopStart">
    <bind component="mInteractividad" role="onSelection">
        <bindParam name="keyCode" value="ENTER">
        </bindParam>
    </bind>
    <bind component="mInteractividad" role="stop">
    </bind>
    <bind component="ctxBecas" role="start">
    </bind>
</link>
```

Figura 12. Código del conector onKeySelecionStopStart

Figura 13. Página principal de la aplicación NCL

```
<link id="link4" xconnector="conn#onSelectionStartNStopN">
    <bind component="mboton6" role="onSelection">
        <bindParam name="keyCode" value="ENTER">
        </bindParam>
    </bind>
    <bind component="mLibreta" role="stop">
    </bind>
    <bind component="mlibreta2" role="stop">
    </bind>
    <bind component="mlibreta6" role="start">
    </bind>
    <bind component="mMenu2" role="start">
    </bind>
</link>
```

Figura 14. Código del conector onSelectionStopNStartN

Para desplazarse entre las imágenes que contienen información se utilizó el conector **onKeySelectionStopStart** el cual permite ocultar una imagen para presentar otra, lo particular de este conector es que se utilizaron los cursores **LEFT** y **RIGHT** para activar la condición de funcionamiento del mismo. En la Fig. 15 se observa el código del conector **onKeySelectionStopStart** implementado, y se puede apreciar que el elemento media (**mLibreta4**) que tiene información se ocultará y se presentará la información de otro (**mLibreta41**) cuando el usuario presione la tecla **RIGHT** y el elemento media (**mLibreta4**) está seleccionado.

La distribución de los elementos media y su interacción presentada por el *plugin Structural View* se pueden observar en la Fig. 16, en la cual se da la percepción del funcionamiento de la aplicación y del desarrollo, ya que se encuentran todos sus elementos.

4.1 Funcionamiento de la aplicación

Para el ícono de interactividad se eligió una imagen que lleva solo el nombre de la aplicación "Becas" y un círculo con la palabra "OK" que insinúa que se debe presionar el botón OK en el control remoto para interactuar con la aplicación. En la Fig. 17 se observa el ícono en la parte superior derecha de la imagen.

Después de presionar el botón OK se desplegará la página principal con las diferentes opciones de navegación, como se presenta en la Fig. 15. Esta aplicación utiliza el *feed* RSS de la Escuela Politécnica Nacional y mostrará la información de dicho *feed*. También existe información de ayuda para el uso del control remoto con las indicaciones necesarias para usar la aplicación.

4.2 Pruebas de la aplicación en equipos reales

En la Fig. 18 y la Fig. 19 se presentan las imágenes obtenidas al realizar las pruebas con equipos reales.

```
<link id="link12" xconnector="conn#onKeySelectionStopStart">
    <bind component="mlibreta4" role="onSelection">
        <bindParam name="keyCode" value="CURSOR_RIGHT">
        </bindParam>
    </bind>
    <bind component="mlibreta4" role="stop">
    </bind>
    <bind component="mlibreta41" role="start">
    </bind>
</link>
```

Figura 15. Código del conector onKeySelectionStopStart

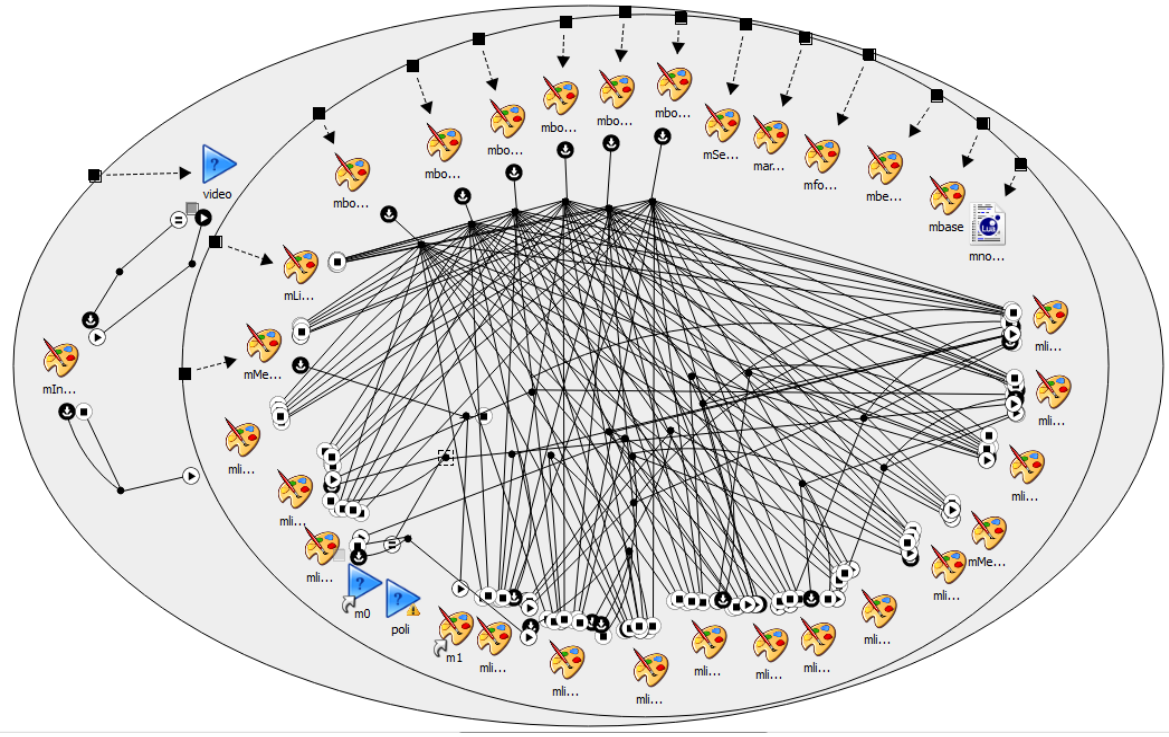

Figura 16. Distribución de elementos e interacción según el *plugin Structural View*

4.2 Encuestas MOS

Las encuestas MOS (*Mean Opinion Score*) se han empleado para conocer el punto de vista del usuario respecto a la calidad de un servicio. Se realizaron encuestas MOS a un grupo de 50 personas sobre la aplicación interactiva "Becas Universidades de Excelencia" para conocer la opinión del televidente. Cada uno de los ítems de la encuesta se calificó con un valor numérico de 1 (malo) a 5 (excelente). En las encuestas se tomaron en consideración los siguientes parámetros: diseño gráfico, navegabilidad, interactividad y contenido. Con los resultados de la encuesta MOS se realizaron los cálculos de la media, desviación estándar y mediana. La media de toda la aplicación es 4.229, la desviación estándar es 0.677, lo que demuestra la semejanza de opiniones entre los encuestados y la mediana tiene un valor de 4. Estas encuestas permiten concluir que la aplicación tiene una adecuada aceptación.

Figura 17. Inicio de la aplicación

Figura 18. Inicio de la aplicación en equipos reales

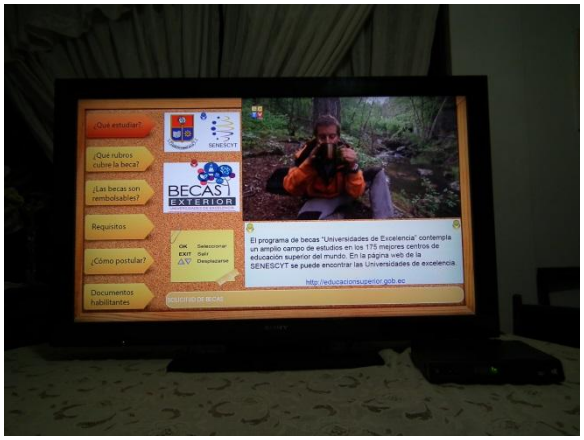

Figura 19. Página principal de la aplicación en equipos reales

5. CONCLUSIONES

Las aplicaciones Ginga NCL causan un impacto positivo en los televidentes, ya que pueden darle un valor agregado a la programación normal de la televisión, permitiendo que estos interactúen con la programación.

El *plugin* que se desarrolló ayuda a las personas que no tienen conocimientos sobre los lenguajes de programación NCL y Lua en la creación de aplicaciones para televisión digital que obtengan y presenten la información desde un *feed* RSS.

Como la televisión es un medio de comunicación masivo con gran aceptación en la sociedad ecuatoriana, la aplicación interactiva realizada podrá ser utilizada por un gran número de personas, ya que la televisión tiene mayor acogida que otros medios de comunicación, permitiendo difundir a gran escala el tema "Becas Universidades de Excelencia".

La generación de aplicaciones Ginga presenta nuevos retos como por ejemplo, es necesario realizar trabajos multidisciplinarios, pues no solamente se requiere de programadores, sino también diseñadores y comunicadores sociales que ayuden a mejorar la calidad visual y el contenido de la aplicación para poder captar la atención del televidente.

Las limitantes a la hora de desarrollar aplicaciones para televisión digital son el tamaño del texto y el contenido en si ya que si se tiene demasiada información el televidente no le va a interesar y simplemente no la utilizará.

Un inconveniente de las aplicaciones para televisión digital es su tamaño que estas tengan ya que surgirán problemas si se transmiten aplicaciones demasiado grande debido a la memoria reducida que tiene la mayoría de STB, los fabricantes de STB recomiendan que las aplicaciones no deben ser mayor a de 6 Mega Bytes para un correcto procesamiento de las mismas.

Para reducir el tamaño de las aplicaciones se aconseja comprimir las imágenes que se desea presentar utilizando mejores tasa de compresión, pero se debe tomar en cuenta que cuando se comprimen las imágenes pierden su resolución y calidad.

REFERENCIAS

[1] ABNT NBR 15606-2, Documento electrónico http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15606-2_2007Ing_2008.pdf

[2] ABNT NBR 15607-1, Documento electrónico http://www.abnt.org.br/imagens/Normalizacao_TV_Digital/ABNTNBR15607-1_2008Ed1.pdf

[3] Blanchette, Jasmin. "C++ GUI Programming with Qt 4". 2006.

[4] Composer NCL. http://Composer.telemidia.puc-rio.br.

[5] Comunidad Ginga Ecuador. http://comunidadgingaec.blogspot.com.

[6] EiTV Developer Box. http://www.eitv.com.br/devbox_es.php.

[7] GINGA NCL. http://www.softwarepublico.gov.br/dotlrn/clubs/ginga/gingancl/xowiki/gingancl_vm.

[8] Ginga4Windows. http://Composer.telemidia.puc-rio.br/en/news/ginga4windows_0.13.1_released.

[9] Gomes Soares, Luiz. "Part 1 – NCM Core". Documento electrónico http://www.telemidia.puc-rio.br/sites/telemidia.puc-rio.br/files/Part1%20-%20NCM%203.0.pdf

[10] Lerusalimschy, Roberto. "Programming in Lua". 2006.

[11] Moreira Gomes, Johnny. "Utilizando o Lua Xml Parser para leitura simples de arquivos xml em aplicações NCL/Lua"

[12] Paucar Curasma, Ronald. "Análisis y Modelamiento de las Técnicas de Canal de Retorno e Interactividad para el Estándar de Televisión Digital Terrestre ISDB-T". 2010.

[13] Qt Creator. http://qt-project.org/wiki/category:tools::qtcreator.

[14] Soares, Luiz Fernando Gomes. "Progrmando em NCL 3.0". 2012.

[15] Telemidia, "Exemplo 6 - Consulta ao Google". http://www.telemidia.puc-rio.br/~francisco/nclua/tutorial/exemplo_06.html.

[16] Thelin, Johan. "Foundations of Qt Development". 2007.