ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE BÚSQUEDA, ALMACENAMIENTO Y PROCESAMIENTO DE INFORMACIÓN PARA GENERAR CONTENIDO INTERACTIVO DE TELEVISIÓN DIGITAL

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

DAVID FABIÁN CEVALLOS SALAS
davidcepn@gmail.com

FERNANDO ANDRÉS CEVALLOS SALAS
epnfernando@gmail.com

DIRECTOR: ING. IVÁN MARCELO BERNAL CARRILLO, Ph.D.
ivan.bernal@epn.edu.ec

CODIRECTOR: ING. RAÚL DAVID MEJÍA NAVARRETE, MSc.
david.mejia@epn.edu.ec

Quito, Mayo 2014

DECLARACIÓN

Nosotros, David Fabián Cevallos Salas y Fernando Andrés Cevallos Salas, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

_____ _____

David Fabián Cevallos Salas Fernando Andrés Cevallos Salas

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por David Fabián Cevallos Salas y Fernando Andrés Cevallos Salas, bajo mi supervisión.

Ing. Iván Bernal, Ph.D.	**Ing. David Mejía, MSc.**
DIRECTOR DEL PROYECTO	**CODIRECTOR DEL PROYECTO**

AGRADECIMIENTO

La satisfacción más grande que hemos tenido en nuestras vidas, es la oportunidad de agradecer a todas aquellas personas que nos apoyaron y confiaron en nosotros para lograr este gran sueño, y aunque no existen palabras suficientes para hacerlo, esperamos que estas líneas expresen parte de lo que sentimos.

Queremos agradecer a Dios por bendecirnos con nuestro hogar y por guiar nuestro camino. Por darnos la fortaleza para enfrentar y vencer cada obstáculo que se nos presenta … por hacer posibles nuestros sueños.

A nuestros padres Janneth y Fabián, por brindarnos su amor incondicional todos los días. Por cuidarnos y por guiar nuestros pasos. Gracias por todo el esfuerzo que realizan con tanto amor por nosotros.

A nuestra hermana Erika, por brindarnos su cariño y apoyo. Gracias por estar siempre a nuestro lado, especialmente en los momentos más difíciles, brindándonos todo tu amor.

A nuestra tía Marthita, por apoyarnos en todos nuestros proyectos de vida y por brindarnos su amor y confianza. Gracias por todo lo que haces por nosotros.

A nuestra abuelita Martha, por brindarnos su infinito amor, por sus enseñanzas y por su comprensión. Gracias por velar por nosotros y por estar siempre a nuestro lado.

A los hermanitos Mateo y David Idrovo, por brindarnos siempre su cariño. Por sus palabras de aliento y por su confianza. Gracias por la gran amistad que nos brindan chicos.

Al Dr. Iván Bernal, nuestro más sincero agradecimiento por haber confiado en nosotros para la realización de este proyecto y por su apoyo, tiempo, colaboración y paciencia en el desarrollo del mismo. Gracias por todos sus consejos y por haber atendido tan amablemente a todas nuestras inquietudes.

Al MSc. David Mejía, por habernos dado la oportunidad de llevar a cabo este proyecto y por guiarnos en la realización del mismo. Por sus enseñanzas en las aulas que estamos seguros sabremos utilizar en nuestra vida profesional y por inculcarnos el amor hacia nuestra carrera. Gracias por la gran amistad y confianza que nos ha brindado.

A la Ing. María Soledad Jiménez, al Ing. Henry Echeverría, al Ing. Fernando Flores, al Ing. Adrián Zambrano, al Ing. Pablo Hidalgo, a la Ing. Mónica Vinueza, a la Ing. Soraya Sinche y a todos nuestros profesores de la EPN, gracias por su apoyo y por sus enseñanzas.

A la Sra. Wilmita Guerrero, nuestro más sincero agradecimiento por su inmensa amabilidad.

A tod@s nuestr@s amig@s, por brindarnos su gran amistad y por sus palabra de aliento que jamás olvidaremos … gracias chic@s.

A todas las personas que hicieron posible alcanzar este sueño … mil gracias.

David y Fernando

DEDICATORIA

Con infinito amor a la memoria de
la madre de nuestra madre

CONTENIDO

ÍNDICE DE FIGURAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

ÍNDICE DE TABLAS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

ÍNDICE DE CÓDIGOS

CAPÍTULO 1

CAPÍTULO 2

CAPÍTULO 3

CAPÍTULO 4

RESUMEN

Las aplicaciones interactivas son sin lugar a duda una de las principales ventajas de la televisión digital. El estándar ISDB-Tb (*International System for Digital Broadcast, Terrestrial, Brazilian version*) dispone del *middleware* denominado Ginga, el cual facilita la comunicación del flujo de datos de las aplicaciones con el *hardware* de los *Set Top Boxes* (STB), haciendo a las aplicaciones independientes del receptor.

Para crear aplicaciones interactivas con NCL existen al momento pocas herramientas, algunas aún en desarrollo como el IDE NCL Composer. Sin embargo, el uso de esta herramienta demanda conocimientos de programación declarativa e imperativa, por lo que en este Proyecto de Titulación se presenta un sistema de búsqueda, almacenamiento y procesamiento de información para generar contenido interactivo de televisión digital basado en Ginga-NCL.

El sistema de búsqueda, almacenamiento y procesamiento de información para generar contenido interactivo de televisión digital está conformado por:

- Un *plug-in* para el IDE NCL Composer, el cual ha sido nombrado Menu Creator.
- Un Subsistema de Adquisición y Procesamiento Datos (SAPDa).

El *plug-in* para el IDE NCL Composer extiende la funcionalidad de dicho IDE para que sea capaz de crear automáticamente menús, generando de manera transparente para el usuario todo el código NCL correspondiente.

A su vez, el SAPDa está conformado por:

- Un servicio web WCF híbrido que, principalmente, implementa un *bot* de búsqueda para obtener información de sitios web, gestiona las bases de datos para el almacenamiento y procesamiento de información y envía los datos solicitados al STB del televidente a través del canal de retorno.
- Una aplicación de escritorio para consumir el servicio web (cliente), la cual ha sido nombrada Textual Data Creator, que permite al usuario realizar las

acciones de búsqueda, almacenamiento y procesamiento de información.

- Una aplicación de escritorio (aplicación Mixer), la cual ha sido nombrada NCL-Textual Data Mixer, encargada de realizar la generación de datos NCL (elementos media con la información que visualiza el televidente).
- Una base de datos por cada aplicación interactiva creada, la cual es generada automáticamente por el servicio web.

Para crear una aplicación interactiva haciendo uso del sistema, el usuario puede emplear el IDE NCL Composer y el *plug-in* para desarrollar una aplicación interactiva que presente una estructura estática (menús vacíos). Luego, mediante la aplicación cliente (Textual Data Creator) el usuario puede hacer uso del servicio WCF para realizar las acciones de búsqueda, almacenamiento y procesamiento de información, para así poder determinar la información a presentarse en la estructura estática y almacenarla en una bases de datos. Finalmente, empleando la aplicación Mixer (NCL-Textual Data Mixer) el usuario podrá generar los elementos media de la aplicación, obteniendo de esta manera la aplicación interactiva final a ser ejecutada en el STB del televidente.

El *plug-in* para el IDE NCL Composer ha sido desarrollado empleando la biblioteca multiplataforma Qt y su lenguaje nativo C++, lo cual permite que el *plug-in* pueda ser utilizado dentro del IDE NCL Composer al igual que el resto de *plug-ins* oficiales.

El servicio web WCF y las aplicaciones Textual Data Creator y NCL-Textual Data Mixer han sido desarrolladas empleando el IDE Visual Studio 2012, el cual facilita la creación de aplicaciones con interfaz gráfica de usuario para Windows, y el lenguaje de programación C#.

Para el almacenamiento de la información se ha empleado SQL Server 2008 por la facilidad que brinda para la creación de bases de datos relacionales y por la compatibilidad de conexión cliente-servidor que permite con el IDE Visual Studio.

PRESENTACIÓN

El diseño e implementación del sistema de búsqueda, almacenamiento y procesamiento de información para generar contenido interactivo de televisión digital se detalla en cinco capítulos.

En el Capítulo 1 se realiza un resumen del marco teórico de televisión digital. Se analiza el *middleware* Ginga y las opciones de desarrollo para aplicaciones interactivas que este *middleware* ofrece (ambiente declarativo e imperativo). También se realiza una explicación general del canal de retorno y una breve revisión de las herramientas disponibles para la realización de aplicaciones interactivas con Ginga-NCL, entre las que se tiene al IDE NCL Composer.

En este capítulo además, se realiza un análisis del lenguaje NCL centrándose en las entidades que dan lugar al documento NCL de la aplicación con el contenido interactivo a presentarse al televidente. También se realiza un análisis del lenguaje Lua. El capítulo finaliza con una descripción del funcionamiento de los servicios web WCF.

En el Capítulo 2 se presenta el diseño e implementación del *plug-in* de generación automática de menús Menu Creator. En primer lugar, se analiza la estructura del IDE NCL Composer y los mecanismos de comunicación que pueden emplear los *plug-ins* para comunicarse con el composer-core, el cual es el principal componente del IDE. Luego, se presenta en detalle las clases implementadas para la creación del *plug-in*.

El capítulo finaliza con la descripción de las características y funcionalidades del *plug-in*. Para mostrar estas características y funcionalidades, en este capítulo se presenta, a manera de ejemplo, una aplicación interactiva que presenta información sobre la fauna de algunas islas del Archipiélago de Galápagos.

En el Capítulo 3 se presenta el diseño e implementación del Subsistema de Adquisición y Procesamiento de Datos (SAPDa). En primer lugar se presenta el diseño e implementación del servicio web WCF y se analiza sus componentes SOAP y REST. En su componente SOAP se explica cómo se realiza la búsqueda

de información en sitios web empleando el buscador de Google y la conexión con SQL Server para la creación, modificación y eliminación de bases de datos y tablas. En su componente REST, se explica cómo se puede configurar el servicio para retornar información como texto plano a través del canal de retorno.

Luego, se explica el diseño e implementación de la aplicación cliente (Textual Data Creator), cuya interfaz gráfica permite al usuario consumir el servicio SOAP para realizar las acciones de búsqueda, almacenamiento y procesamiento de información.

El capítulo finaliza con la explicación del diseño e implementación de la aplicación Mixer (NCL-Textual Data Mixer) y la descripción del *script* Lua que automáticamente genera para obtener información en el STB a través del canal de retorno.

En el Capítulo 4 se presenta el desarrollo de una aplicación interactiva enfocada en la temática de la Educación Superior del Ecuador, creada haciendo uso del sistema de búsqueda, almacenamiento y procesamiento de información. La aplicación interactiva presenta información general acerca de las once universidades que conformaron la categoría A durante el período 2009-2013, como resultado de la evaluación realizada por el CONEA (Consejo Nacional de Evaluación y Acreditación de la Educación Superior del Ecuador).

A partir de esta aplicación se presenta, como ejemplo de la flexibilidad que el sistema brinda en el desarrollo de aplicaciones interactivas, el desarrollo de una segunda aplicación a partir de la primera que presenta información general acerca de las cinco universidades que en la actualidad conforman la categoría A, como resultado de la evaluación realizada por el CEAACES (Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior) en noviembre de 2013.

El capítulo finaliza presentando las pruebas de funcionamiento y los resultados obtenidos.

En el Capítulo 5 se presentan las conclusiones obtenidas al culminar el proyecto y ciertas recomendaciones que ayudarían a mejorar o aumentar los beneficios que tiene el sistema.

Los anexos no han sido impresos debido a su gran extensión, pero se incluyen en el DVD que acompaña a este documento.

CAPÍTULO 1

FUNDAMENTOS TEÓRICOS

En este capítulo se realiza un estudio de los fundamentos teóricos necesarios para entender el diseño e implementación del sistema de búsqueda, almacenamiento y procesamiento de información, cuyo diagrama se presenta en la Figura 1.1.

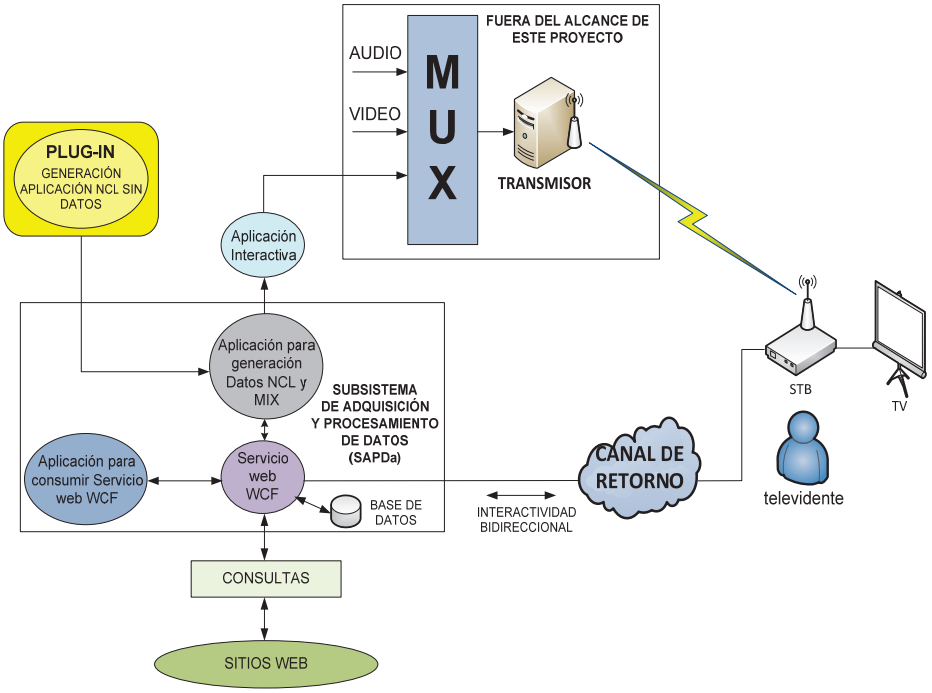


Figura 1.1 Diagrama del sistema

El sistema brinda varias ventajas tales como:

- Posibilita la creación de aplicaciones interactivas en menor tiempo y sin necesidad de conocimientos avanzados del lenguaje NCL.
- Permite crear aplicaciones interactivas con menús de un alto nivel de interactividad. El *plug-in* genera automáticamente los menús con elementos media y varias características y funcionalidades que dan lugar a aplicaciones llamativas para el televidente.

- Permite obtener información de sitios web para que dicha información pueda ser almacenada en bases de datos y, posteriormente, radiada conjuntamente con la aplicación interactiva u obtenida a través del canal de retorno. Para esta última funcionalidad, el sistema genera *scripts* Lua de forma automática.
- Realiza de forma totalmente transparente para el usuario las acciones requeridas para el uso de la información en las bases de datos.

1.1 TELEVISIÓN DIGITAL [9], [14], [18]

Se entiende por televisión digital o DTV (por sus siglas en inglés *Digital TV*) a un conjunto de tecnologías de transmisión y recepción de audio, video y datos por medio de señales digitales, brindando múltiples ventajas y abriendo la posibilidad de prestar nuevos servicios en el hogar [18]. Además de lograr un sonido e imagen de mejor calidad, la televisión digital brinda la oportunidad de crear aplicaciones interactivas.

Una aplicación interactiva es un programa informático que interactúa con el televidente a través de elementos media como texto, gráficos, audio y video. En una aplicación interactiva es el televidente el que decide la información que desea visualizar haciendo uso del control remoto. La aplicación debe ser capaz de ser ejecutada en el receptor del televidente sobre cualquier plataforma de *hardware* y sistema operativo [9].

Para la recepción se requiere de un receptor con el sintonizador digital incorporado, o si se emplea un receptor analógico, de un decodificador (también llamado *Set Top Box*, STB) que permita la conversión de la señal digital a la analógica convencional [18].

El sistema brasileño de televisión digital, adoptado por el Ecuador en marzo de 2010, posee una arquitectura en capas jerárquicas para la transmisión y recepción de aplicaciones interactivas y contenido televisivo, en la que cada capa ofrece servicios a la capa inmediatamente superior y hace uso de los servicios brindados por la capa inmediatamente inferior, como se presenta en la Figura 1.2 [14].

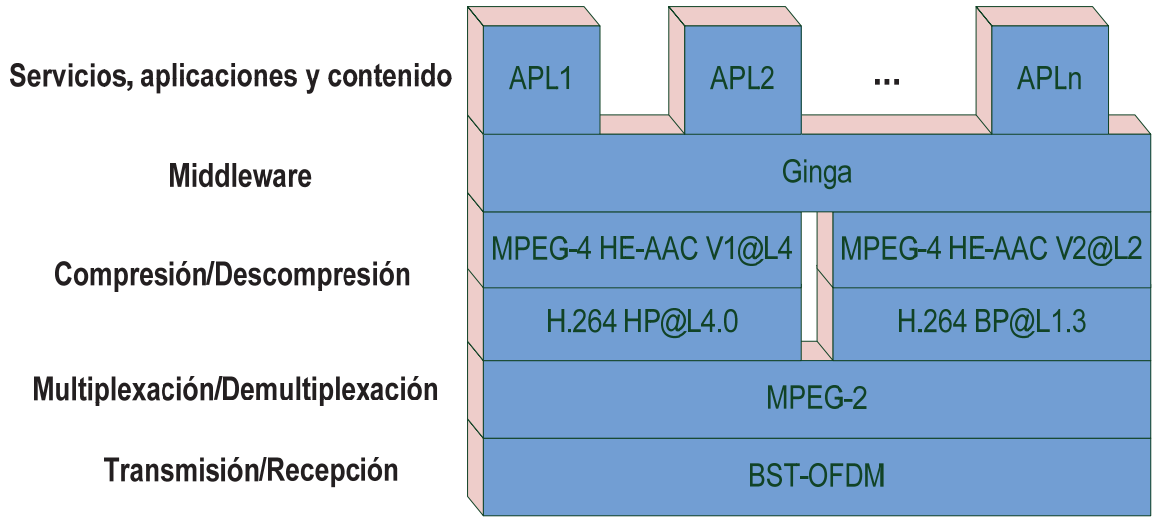


Figura 1.2 Arquitectura del sistema brasileño de televisión digital [14]

La arquitectura está conformada por las siguientes capas:

- **Servicios, aplicaciones y contenido:** En el transmisor es la capa responsable de la generación de las señales de audio, video y datos, así como de la implementación de servicios interactivos [9]. En el receptor permite que las aplicaciones y el contenido televisivo puedan presentarse al televidente.
- **Middleware:** Funciona como una capa de abstracción del *software* en el receptor. Su propósito es permitir a las aplicaciones comunicarse con las capas inferiores independientemente del *hardware* del receptor y sistema operativo empleado. El sistema brasileño tiene su propio *middleware* denominado Ginga [14].
- **Compresión/Descompresión:** En esta capa se encuentran las técnicas empleadas para la compresión de audio y video. El sistema brasileño hace uso del estándar MPEG-4 HE AAC para la compresión de audio y del estándar H.264[1] para la compresión de video [14]. Sin la compresión (especialmente de video) la señal de televisión digitalizada no se podría transmitir ni siquiera ocupando todo el ancho de banda de un canal analógico [18]. En el lado del receptor, esta capa decodifica los flujos de audio y video comprimidos.

[1] También denominado estándar MPEG-4 parte 10 o *Advanced Video Coding* (AVC).

- **Multiplexación/Demultiplexación:** También denominada capa de transporte. En el transmisor es la capa encargada de tomar el flujo de audio y video compresos y combinarlos con el flujo de datos en un proceso de multiplexación. La información resultante multiplexada recibe el nombre de *transport stream* o flujo de transporte [14]. En el lado del receptor, esta capa realiza la función inversa, esto es, toma el *transport stream* y demultiplexa la señal para separar el flujo de audio, video y datos con el fin de que sean procesados por la capa superior. El sistema brasileño hace uso del estándar MPEG-2 para esta función [9].
- **Transmisión/Recepción:** También denominada capa física. Es la responsable de llevar la información desde el transmisor hasta el receptor del televidente. Para esto, el sistema brasileño adoptó la misma tecnología de modulación que el sistema japonés basada en BST-OFDM[2] [9] para la transmisión de la señal en un sistema de televisión digital terrestre. Sin embargo, como existen diversos sistemas de televisión digital, el contenido televisivo podría ser encapsulado en IP o algún otro protocolo para su transmisión en IPTV, WebTV, P2PTV u otros sistemas en lugar de emplear un esquema de modulación.

Un sistema de televisión digital terrestre se expone en la Figura 1.3. Se lo puede entender como un sistema cliente-servidor [9]. El servidor corresponde a una radiodifusora (estación de emisión) que provee todo el programa y el cliente corresponde al usuario final o televidente. El contenido de televisión comprende el audio y video proveniente de una cámara o de un servidor de video. Además de las capas mencionadas anteriormente, el sistema puede emplear un canal de retorno[3] para interactuar con el televidente.

Para que las aplicaciones interactivas puedan ser ejecutadas, el receptor (ya sea un equipo con el sintonizador digital incorporado o un STB) debe contener el

[2] BST-OFDM (*Band Segmentation Transmission- Orthogonal Frequency Division Multiplexing*) es una técnica de modulación en la que se divide el canal de 6 MHz en 13 segmentos de 428,57 KHz para asignarse cada uno a un servicio diferente. OFDM permite que la información de cada segmento sea transmitida a una frecuencia de portadora diferente [9].
[3] Ver sección 1.3.

middleware Ginga que permite la comunicación entre las aplicaciones interactivas y el *hardware* subyacente del receptor [9].

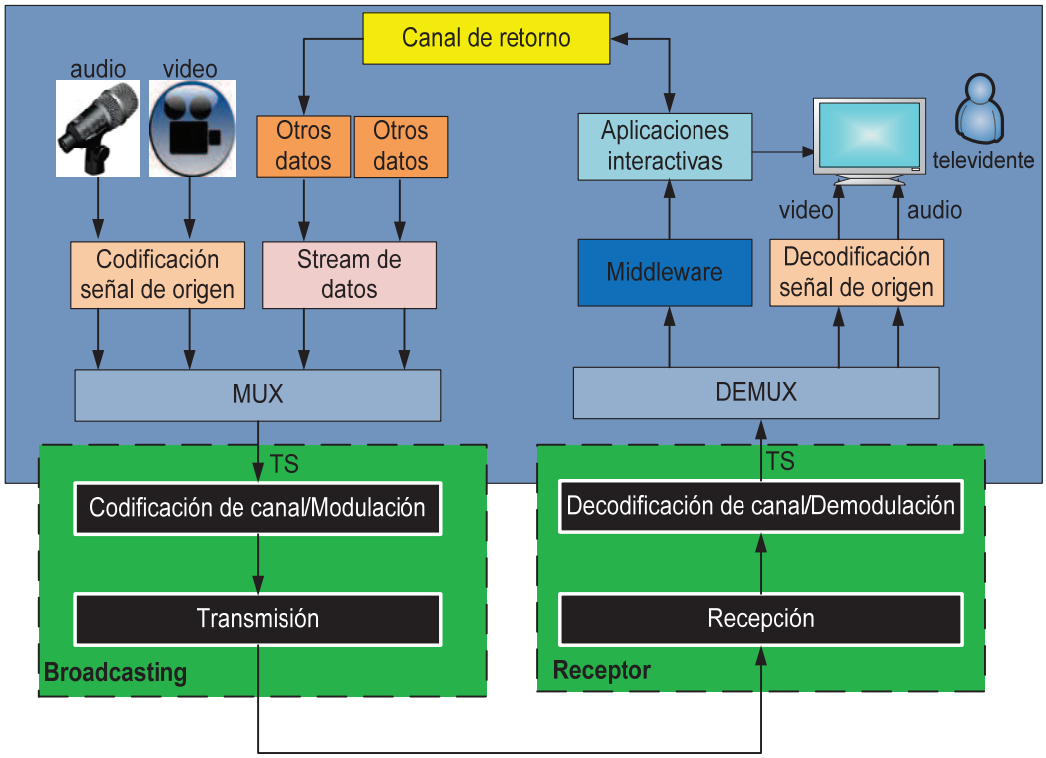


Figura 1.3 Sistema de televisión digital terrestre [9]

1.2 MIDDLEWARE GINGA Y AMBIENTES DE PROGRAMACIÓN PARA LA CREACIÓN DE APLICACIONES INTERACTIVAS [5], [6], [8], [9], [12], [14], [19], [20], [22]

Una aplicación interactiva podría ser ejecutada directamente sobre el sistema operativo del receptor; sin embargo, los sistemas operativos de propósito general no dan un soporte adecuado a las aplicaciones de televisión digital [9]. Por lo tanto, para hacer a las aplicaciones independientes del *hardware* y del *software* de un receptor específico, así como para brindar un mejor soporte a las aplicaciones, es necesario de un *middleware* que cumpla con las funciones de abstracción de la heterogeneidad en las capas inferiores del receptor. El *middleware* del sistema brasileño denominado Ginga cumple con esta función.

Ginga es vital en el sistema brasileño de televisión digital ya que permite relacionar a dos importantes actores de la industria como lo son los generadores de contenido y los fabricantes de receptores [9].

1.2.1 REQUISITOS

Para que Ginga funcione como capa de abstracción del *software* y sea capaz de brindar soporte a distintos tipos de aplicaciones interactivas, debe cumplir con ciertos requisitos al igual que cualquier otro *middleware* de televisión digital.

Un primer requisito es que el *middleware* debe ser capaz de soportar el sincronismo de elementos media de forma general, y de la interacción con el usuario de forma particular. Esta característica permite crear aplicaciones capaces de presentar elementos media en una determinada posición del terminal y en un determinado instante de tiempo, detenerlos y pausarlos cuando se desee, así como responder a los eventos generados por el televidente [9].

El contenido media a mostrarse a un televidente no siempre será el mismo que se muestra a otro televidente. Este contenido podría requerir ser adaptado de acuerdo con la idiosincracia y cultura de la población a la cual va dirigido. Esto lleva a un segundo requisito que debe soportar un *middleware*: ofrecer la adaptación de contenido y la forma en la que el contenido es exhibido.

El terminal en el que se muestra el contenido interactivo tiene un papel importante. El televidente podría desear hacer uso de una aplicación interactiva no solo en un televisor, sino en otro tipo de terminales como un telefóno celular, un computador, un asistente personal digital (PDA), etc. Esto lleva a un tercer requisito que el *middleware* debe cumplir: dar soporte para múltiples dispositivos de visualización de contenido.

Para permitir una interactividad total con el televidente, muchas veces es necesario traer el contenido media en tiempo de ejecución a través del canal de retorno[4] desde un servidor central. Por lo tanto, un cuarto requisito que debe cumplir un *middleware* es brindar soporte para la exhibición de contenido en vivo (en tiempo de ejecución).

Un quinto requisito consiste en brindar soporte para la definición de relaciones de sincronismo espacial y temporal independientemente del contenido de los elementos media.

[4] Ver sección 1.3.

El *middleware* Ginga cumple con estos cinco requisitos permitiendo brindar soporte a distintos tipos de aplicaciones interactivas [9].

1.2.2 AMBIENTES DE PROGRAMACIÓN PARA LA CREACIÓN DE APLICACIONES INTERACTIVAS

1.2.2.1 Ambiente Declarativo

El ambiente de programación declarativo es empleado por lenguajes de programación de alto nivel de abstracción[5], los cuales son intuitivos y fáciles de usar para crear aplicaciones con una funcionalidad específica [14]. En los lenguajes declarativos, el programador tan solo indica el tipo de tareas que se deben realizar sin preocuparse de los detalles de cómo el ejecutor (intérprete, compilador o máquina real o virtual de ejecución) lleva a cabo dichas tareas, por lo tanto, no se requieren muchas líneas de código y la probabilidad de cometer errores por parte del programador es menor que al emplear un lenguaje imperativo [9].

Los lenguajes declarativos más comunes en el área de televisión digital son NCL[6], SMIL[7] y XHTML[8]. Ginga emplea el lenguaje declarativo NCL [9].

1.2.2.2 Ambiente Imperativo

Un ambiente de programación imperativo[9] involucra que el programador indique cada paso que debe ser ejecutado (algoritmo), por lo que se puede decir que el programador tiene mayor control sobre el código, el cual está obligado a cumplir con lo que el programador le indique [14]. Sin embargo, la aplicación está expuesta a posibles errores que el programador podría cometer. El uso de lenguajes imperativos usualmente demanda cierto nivel de conocimientos en el lenguaje de programación que se emplee, pues introduce una mayor complejidad de programación en la creación de la aplicación [9].

---

[5] Los lenguajes de abstracción permiten aislar un elemento del resto de elementos que lo acompañan.
[6] *Nested Context Language*. Ver sección 1.4.
[7] *Synchronized Multimedia Integration Language.* Es un lenguaje declarativo que posibilita la creación de presentaciones multimedia permitiendo integrar audio, video, imágenes, texto, etc.
[8] *eXtensible HyperText Markup Language.* Es básicamente HTML expresado como XML.
[9] También denominado ambiente procedural.

Los lenguajes imperativos, en la creación de aplicaciones interactivas, son usados para tareas específicas como procesamiento matemático, manipulación de texto, comunicación a través del canal de retorno[10] y tareas que requieran de la implementación de algoritmos o estructuras de datos.

Entre los lenguajes de programación más comunes en el área de televisión digital están C, Java, Lua y ECMAScript. Ginga emplea Java y Lua como lenguajes imperativos [9].

1.2.2.3 Aplicaciones Híbridas

Una aplicación híbrida es aquella cuyas entidades poseen contenido de tipo declarativo e imperativo [14]. Para poder crear una aplicación híbrida se debe extender la funcionalidad de una aplicación declarativa para ejecutar contenido imperativo, o bien extender la funcionalidad de una aplicación imperativa para crear y ejecutar contenido declarativo.

Las aplicaciones declarativas normalmente hacen uso de *scripts*, cuyo contenido es de tipo imperativo, para así dar lugar a aplicaciones híbridas. De la misma forma, una aplicación imperativa puede hacer referencia a una aplicación declarativa que posee contenido media, por ejemplo, contenido gráfico, o puede construir e iniciar una presentación de aplicaciones con contenido declarativo.

1.2.3 ARQUITECTURA DEL MIDDLEWARE GINGA

La arquitectura del *middleware* Ginga está dividida en tres módulos como se muestra en la Figura 1.4. El primer módulo lo constituye Ginga-NCL para la ejecución de aplicaciones declarativas y el segundo Ginga-J para la ejecución de aplicaciones imperativas [8]. Ambos módulos se encuentran entrelazados para permitir la ejecución de aplicaciones híbridas [14]. El tercer módulo comprende Ginga-CC, el cual brinda servicios para dar soporte a los ambientes declarativo e imperativo y hace uso de un grupo de API (*Application Programming Interfaces*) que ofrecen un conjunto de funciones y procedimientos para permitir la compatibilidad con el sistema operativo [14].

[10] Ver sección 1.3.

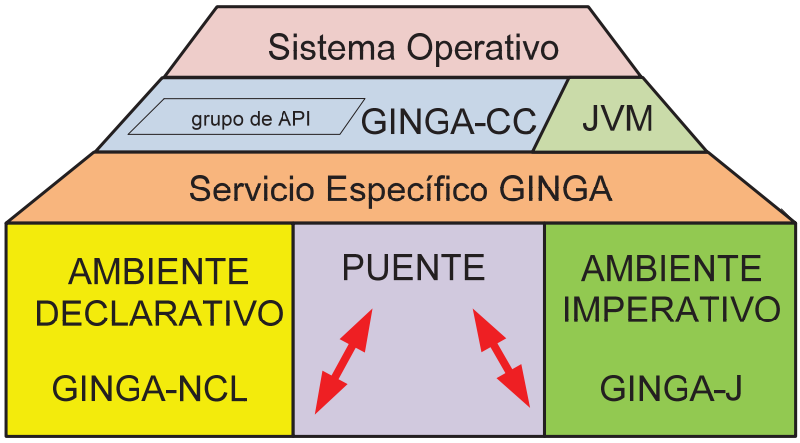


Figura 1.4 Arquitectura del *middleware* Ginga [8]

1.2.3.1 Ginga-NCL

Ginga-NCL provee una infraestructura de presentación para aplicaciones interactivas declarativas desarrolladas con el lenguaje NCL.

La Figura 1.5 muestra los componentes de Ginga-NCL. Básicamente se encuentra conformado por un formateador que se encarga de recibir y controlar las aplicaciones interactivas desarrolladas en NCL garantizando que la sincronización espacial y temporal establecida por el diseñador sea respetada. El analizador de XML y el conversor realizan la conversión de la aplicación NCL a la estructura interna de datos de Ginga-NCL, con el fin de que el formateador pueda controlar la aplicación. Ambos componentes son solicitados por el formateador para realizar las acciones que el diseñador le indique. El componente programador cumple con la función de organizar el orden de la presentación del documento NCL y es el encargado de dar la orden al administrador de reproducción para iniciar la reproducción adecuada de un determinado objeto media.

El administrador de diseño es el motor de presentación a través del cual se brinda soporte a múltiples dispositivos. El administrador de bases privadas cumple con la función de recibir los comandos de edición de los documentos NCL y dar mantenimiento a los documentos NCL que se presentan. Finalmente, el administrador de contextos NCL soporta el contenido incluyendo su adaptación de presentación.

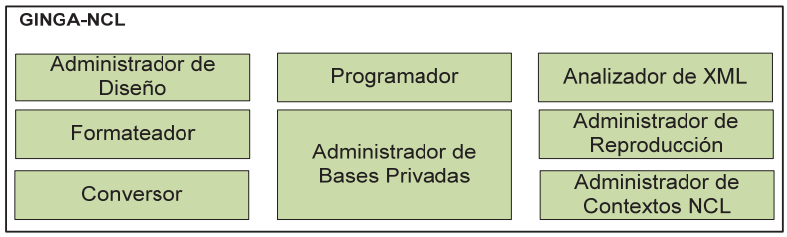


Figura 1.5 Componentes de Ginga-NCL [8]

El ambiente Ginga-NCL permite el uso de Lua, un lenguaje de *scripting*, para la creación de aplicaciones híbridas [14]. Los objetos media[11] de tipo Lua consisten en *scripts* que pueden ser insertados dentro de documentos NCL al igual que cualquier otro tipo de media como imágenes, video o audio[12]. De esta manera, el programador tiene mayor control sobre la funcionalidad de la aplicación para realizar tareas mucho más complejas como la ejecución de algoritmos y la comunicación con un servidor a través del canal de retorno[13].

1.2.3.2 Ginga-J

Ginga-J es el módulo imperativo de Ginga que soporta la ejecución de aplicaciones procedurales creadas en base al lenguaje de programación Java (usualmente llamadas Xlet) [14]. El mecanismo de ejecución del código procedural se basa en la máquina virtual de Java (en inglés *Java Virtual Machine*, JVM), la cual interpreta y ejecuta las instrucciones generadas por el compilador de Java.

Una aplicación imperativa creada con Ginga-J puede hacer referencia a una aplicación declarativa o construir e iniciar una presentación de aplicaciones interactivas con contenido declarativo, creando así una aplicación híbrida [14].

1.2.3.3 Ginga-CC

Ginga-CommonCore (Ginga-CC) es el encargado de concentrar los servicios necesarios tanto para la máquina de presentación (ambiente declarativo) así como para la máquina de ejecución (ambiente imperativo) [14].

---

[11] También llamado nodo de contenido en el modelo *Nested Context Model* (NCM). Ver sección 1.4.1.3.1.
[12] Ver sección 1.5.
[13] Ver sección 1.3.

La Figura 1.6 muestra los componentes de Ginga-CC. Está conformado principalmente por decodificadores de contenido y procedimientos para la obtención de los flujos de audio y video a través del flujo de transporte MPEG-2. Estos decodificadores cumplen con el rol de presentar tipos comunes de contenidos como JPEG, PNG, MPEG y otros formatos.

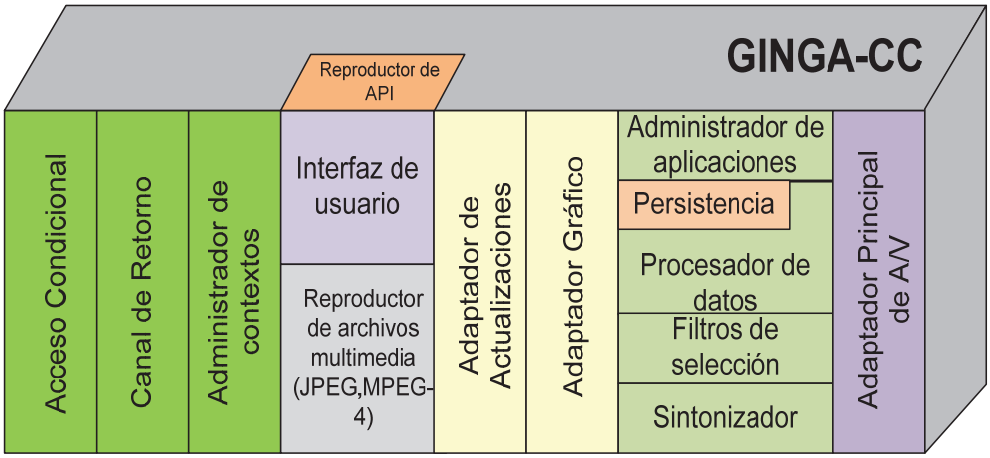


Figura 1.6 Componentes de Ginga-CC [8]

1.2.4 HERRAMIENTAS PARA EL DESARROLLO DE APLICACIONES INTERACTIVAS CON GINGA-NCL

Para poder desarrollar aplicaciones interactivas basadas en Ginga-NCL existen básicamente dos herramientas de desarrollo: NCL Composer y NCL Eclipse.

1.2.4.1 NCL Composer

NCL Composer es una herramienta multiplataforma y flexible de composición multimedia para crear aplicaciones en NCL para televisión digital interactiva (iDTV) [22]. Fue desarrollado en base a la biblioteca multiplataforma Qt[14] y C++ por el laboratorio TeleMídia de la PUC-Rio (Pontificia Universidad Católica de Río de Janeiro) [14].

NCL Composer está basado en una estructura de árbol jerárquico de entidades NCL[15]. Una entidad es una clase que modela de manera lógica una determinada

[14] Qt es una biblioteca cuyo lenguaje nativo es C++, empleada para desarrollar aplicaciones multiplataforma con o sin interfaz gráfica de usuario [6].
[15] Ver sección 1.4.

etiqueta del documento NCL, pudiendo contener entidades hijas y al mismo tiempo ser hija de otra entidad (entidad padre). La Figura 1.7 muestra la lógica de trabajo del modelo interno de NCL Composer. A partir del árbol jerárquico de entidades se genera el documento NCL.

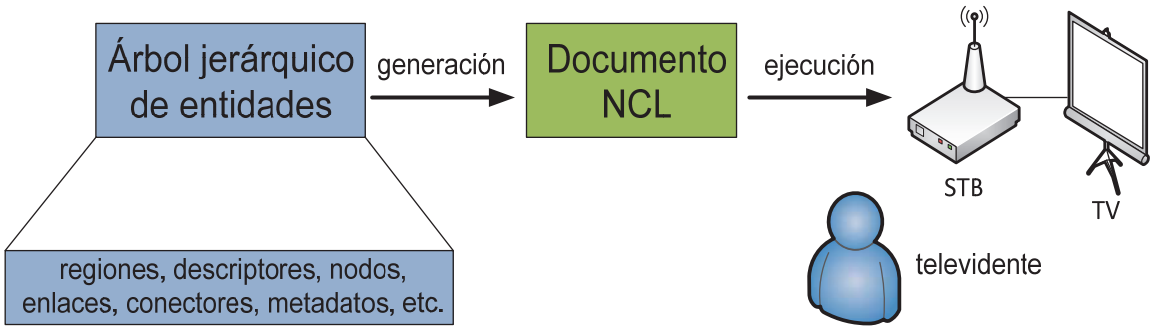


Figura 1.7 Lógica de trabajo de NCL Composer

El IDE NCL Composer está compuesto de tres partes:

- **Composer-core:** Implementa las reglas de negocio para la generación del documento NCL.
- **Composer-gui:** Implementa la interfaz gráfica general de usuario en la que se cargan los *plug-ins*.
- **Composer-plugins:** Comprende el conjunto de *plug-ins* a ser compilados con su interfaz gráfica y hacen uso de la lógica implementada en el composer-core.

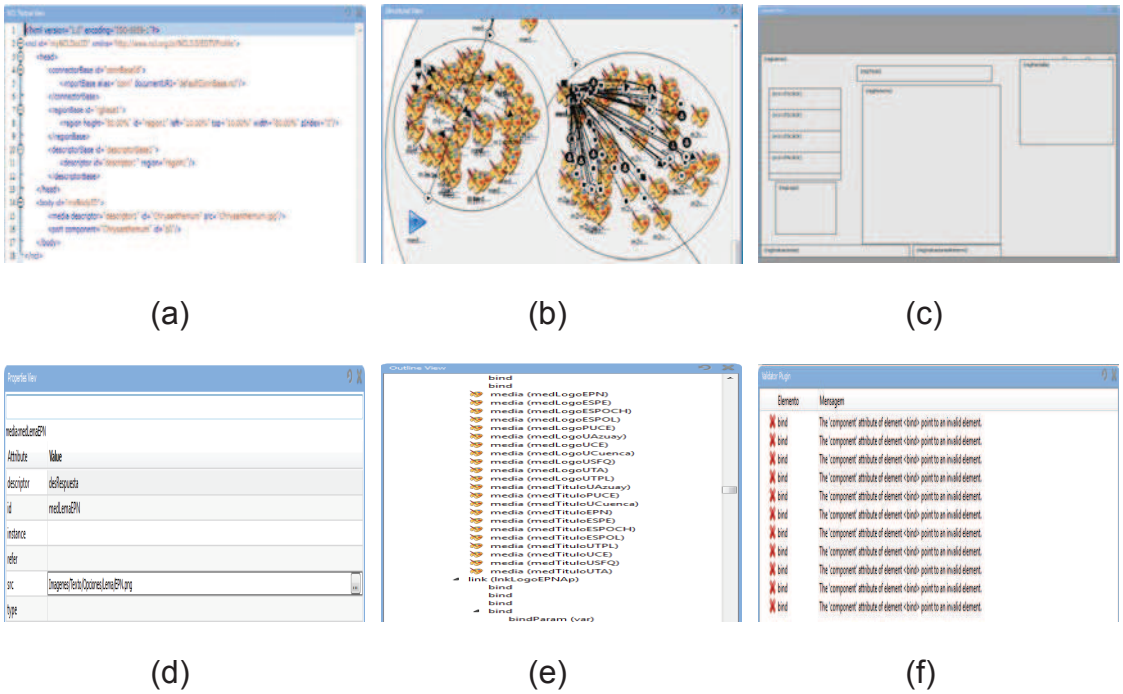
Un *plug-in* es un componente de *software* que añade una característica o un servicio específico a un sistema más grande. Cada *plug-in* puede gestionar el árbol jerárquico de entidades de tal modo que permite al usuario generar el documento NCL, evitándole la tarea de programar directamente el código NCL y brindándole una interfaz gráfica amigable.

La Tabla 1.1 expone los *plug-ins* oficiales existentes a la fecha[16] de la redacción de este documento con los que cuenta NCL Composer y su respectiva funcionalidad. En la Figura 1.8 se presenta la interfaz gráfica de cada uno de los *plug-ins*.

[16] Documento redactado en septiembre de 2013.

Tabla 1.1 *Plug-ins* oficiales del IDE NCL Composer [20]

Plug-in	Funcionalidad
NCL Textual View	Permite la interacción con el contenido textual NCL.
NCL Structural View	Permite la interacción visual con la estructura lógica del documento. En NCL, la estructura lógica es representada por objetos media, contextos y entidades de enlace.
NCL Layout View	Permite al usuario interactuar visualmente con la región donde un objeto media será presentado.
NCL Properties View	Permite al usuario el fácil acceso (y cambio) del contenido de las propiedades de un elemento seleccionado.
NCL Outline View	Presenta la estructura del documento como una vista de árbol. Esta vista permite navegar entre los elementos.
NCL Validator Plugin	Valida documentos NCL cada vez que se modificó el modelo de núcleo, dando información al usuario sobre los errores en el código fuente del documento que se está editando.



(a) (b) (c)

(d) (e) (f)

Figura 1.8 (a) NCL Textual View (b) NCL Structural View (c) NCL Layout View (d) NCL Properties View (e) NCL Outline View (f) NCL Validator Plugin

El código fuente de NCL Composer desarrollado por el laboratorio TeleMídia está disponible en [19] y se lo puede versionar mediante el *software* de control de versiones git [5].

1.2.4.2 NCL Eclipse

NCL Eclipse es un editor presentado como un *plug-in* para el IDE Eclipse, que ofrece diversas funcionalidades para la creación de aplicaciones interactivas basadas en NCL de manera sencilla [14]. Una de las principales funcionalidades del desarrollo de aplicaciones con NCL Eclipse es que permite la identificación de errores en el momento de la creación de la aplicación sin que el programador tenga que ejecutarla para encontrar los posibles errores de sintaxis o semánticos[17].

Para que NCL Eclipse pueda ser utilizado debe ser instalado siguiendo el patrón de instalación de *plug-ins* para el IDE Eclipse [14].

1.3 CANAL DE RETORNO [9], [14]

La principal función del canal de retorno, también llamado canal de interactividad, es permitir la comunicación del receptor del televidente con un servidor. Para este fin, el canal de retorno puede comprender cualquier red de comunicaciones que provea conectividad entre el receptor y el servidor. En la actualidad, Internet es la red más empleada como parte del canal de retorno y tiene toda la infraestructura de acceso.

Un sistema de televisión digital puede operar sin necesidad de emplear el canal de retorno. En este primer nivel de interactividad, también llamado interactividad local [14], las aplicaciones únicamente emplean los datos transmitidos por difusión hacia los receptores de los televidentes [9].

La interactividad en la que se hace uso del canal de retorno, es también denominada interactividad remota y, dependiendo del tipo de comunicación, puede tener varios niveles [14].

El canal de retorno puede ser unidireccional, permitiendo únicamente al receptor enviar datos hacia el servidor. Este segundo nivel de interactividad permite al

[17] Un error de sintaxis se produce cuando se escribe código de una forma no admitida por las reglas del lenguaje mientras que un error semántico se produce cuando la sintaxis del código es correcta, pero la funcionalidad del código no es la que se pretendía [12].

televidente realizar acciones como solicitudes de compra de un determinado producto, votaciones en línea, responder a encuestas, etc.

Un tercer nivel de interactividad consiste en emplear un canal de retorno bidireccional en el que el televidente determina el contenido que quiere visualizar generando una petición al servidor y este le responde con el contenido para ser mostrado al televidente. En este esquema, el canal de retorno es empleado para transmitir datos de interés específico al usuario, evitando así que estos datos sean transmitidos para el público en general con el contenido televisivo [14].

Sin embargo, un canal de retorno bidireccional también puede permitir el envío de datos en banda ancha (*upload*). En este cuarto nivel de interactividad, también llamado interactividad plena, el receptor pasa a actuar como una pequeña emisora y permite brindar servicios como *"social TV"* o *"community TV"* [9].

El sistema brasileño de televisión digital permite los cuatro niveles de interactividad, así como los sistemas: americano, europeo y japonés [9].

1.4 LENGUAJE DE PROGRAMACIÓN NESTED CONTEXT LANGUAGE (NCL) [2], [4], [9], [10], [11], [14]

El ambiente declarativo de Ginga hace uso del lenguaje declarativo *Nested Context Language* (NCL), o lenguaje de contexto anidado, para la creación de aplicaciones interactivas.

NCL es un lenguaje basado en XML (*eXtensible Markup Language*) que hace uso de etiquetas jerárquicas para definir elementos NCL [14]. Un elemento NCL definido a través de una etiqueta puede estar contenido (anidado) dentro de otros elementos padres, así como contener a otros elementos hijos.

1.4.1 MODELO NESTED CONTEXT MODEL (NCM)

El modelo *Nested Context Model* (NCM), o Modelo de Contexto Anidado, es un modelo conceptual empleado para representar y manipular los elementos involucrados en documentos hipermedia [11]. Al ser un modelo conceptual, el modelo NCM establece las estructuras de datos y las relaciones entre estas

estructuras, así como las reglas y operaciones que se pueden utilizar para manipularlas [4].

NCM se basa en el concepto de entidades, las cuales pueden ser principalmente de dos tipos: nodos o enlaces. Los nodos representan información mientras que los enlaces son empleados para establecer las relaciones entre los nodos. Además, el modelo establece otros tipos de entidades útiles para la creación de documentos hipermedia como las regiones y descriptores.

Cada entidad NCM posee un conjunto de propiedades que la identifican. Todas las entidades NCM poseen las propiedades básicas `unique identifier` (`id`), `name`, `description`, `date of creation` y `owner` [4]. De estas propiedades, únicamente la propiedad `id` es obligatoria. Además, cada entidad puede tener una lista de propiedades extendidas definidas por la etiqueta `<property>` para permitir establecer más propiedades de una entidad no solo a través de la herencia[18].

1.4.1.1 Regiones

Para que un nodo en un documento hipermedia pueda ser presentado, es necesario determinar un área para exhibirlo. Para este fin, el modelo NCM define una entidad denominada región, la cual permite determinar dónde presentar un nodo [14].

Una región es definida mediante la etiqueta `<region>` dentro de una base de regiones definida por la etiqueta `<regionBase>` y tiene las siguientes propiedades [14]:

- **`id:`** Identificador único de la región.
- **`left:`** Define la coordenada horizontal a la izquierda de la región.
- **`top:`** Define la coordenada vertical superior de la región.
- **`right:`** Define la coordenada horizontal a la derecha de la región.
- **`bottom:`** Define la coordenada vertical inferior de la región.

[18] La herencia es un mecanismo de programación orientada a objetos a través del cual una clase se deriva de otra que extiende su funcionalidad.

- **`width:`** Define la dimensión horizontal de la región.
- **`height:`** Define la dimensión vertical de la región.
- **`zIndex:`** Define la sobreposición de las regiones. Consiste en un número entre 0 y 255. Una región se presentará por encima de otras regiones sobrelapadas que tengan un valor de `zIndex` menor y por debajo de otras con un valor de `zIndex` mayor.

1.4.1.2 Descriptores

Un descriptor es una entidad NCM que describe parámetros de cómo debe ser exhibido un nodo, además de la región en donde debe ser presentado [9].

Un descriptor es definido mediante la etiqueta `<descriptor>` dentro de una base de descriptores definida por la etiqueta `<descriptorBase>` y tiene dos propiedades básicas [14]:

- **`id:`** Identificador único del descriptor.
- **`region:`** Asocia el descriptor con una región que se definió anteriormente.

Además de estas propiedades, un descriptor puede establecer propiedades adicionales dependiendo del nodo de contenido u objeto media con el que se asocie. Así por ejemplo, un descriptor puede determinar la transparencia de una imagen, el volumen de un video y en general la duración de la presentación de un nodo, entre otros parámetros [14].

1.4.1.3 Nodos

Los nodos representan información y pueden ser de dos tipos: nodos de contenido y nodos de composición.

1.4.1.3.1 Nodos de Contenido

Un nodo de contenido, también llamado nodo media u objeto media, es una entidad NCM que contiene información para mostrarse al televidente. Dependiendo del tipo de información que el nodo contiene se puede especializar

en subclases pudiendo ser un nodo de texto, imagen, audio, video, aplicación, tiempo o configuración [11].

Un nodo de contenido es definido por la etiqueta `<media>` y tiene las siguientes propiedades [14]:

- **`id:`** Identificador único del nodo de contenido.
- **`src:`** Define la localización del contenido del nodo.
- **`descriptor:`** Define un descriptor que controla cómo debe ser presentado el nodo.
- **`type:`** Define el tipo de media (texto, audio, video, etc).

La Tabla 1.2 muestra los tipos de media utlizados por NCL. Un nodo de contenido está conformado por una colección de unidades de información denominadas anclas [11]. La definición de una unidad de información depende de la especialización del nodo. De manera general, cada ancla es un subcojunto de las unidades de información que conforman el nodo y, particularmente, cada nodo posee un ancla que contiene todas las unidades de información de manera que a través de ella se pueda visualizar el contenido del nodo por completo[19] [11].

Tabla 1.2 Tipos de media utilizados por NCL [14]

Tipo de media	Extensión del archivo media
texto/html	.htm, .html
texto/css	.css
texto/xml	.xml
imagen/bmp	.bmp
imagen/png	.png
imagen/gif	.gif
imagen/jpeg	.jpg
audio/basic	.wav
audio/mp3	.mp3
audio/mp2	.mp2
audio/mpeg4	.mp4, .mpg4
video/mpeg	.mpeg, .mpg
aplicación/x-ginga-NCLua	.lua

[19] Un ancla de este tipo es el nodo de contenido en sí definido por la etiqueta `<media>`.

Un ancla está definida mediante la etiqueta `<area>` dentro de un elemento `<media>` y posee las siguientes propiedades [14]:

- **`id:`** Identificador único del ancla.
- **`begin:`** Determina el inicio del ancla en un nodo de contenido continuo[20] y puede ser utilizado con el formato `segundo.fracción` u `hora:minuto:segundo.fracción`.
- **`end:`** Determina el fin del ancla en un nodo de contenido continuo. Es usado en conjunto con el atributo `begin`.
- **`label:`** Define una descripción para el ancla.

El Código 1.1 presenta un ejemplo en el que se define un video a través de la etiqueta `<media>` y se han definido dos anclas a través de la etiqueta `<area>`. La primera ancla presenta el video desde el segundo 2 hasta el segundo 5 mientras que la segunda ancla presenta el mismo video desde el segundo 5 hasta el segundo 10.

```
<media id="video" src="video.avi" descriptor="dpVideo"
        <area id="parte01" begin="2s" end="5s">
        <area id="parte02" begin="5s" end="10s">
</media>
```

Código 1.1 Ejemplo de definición de un nodo de contenido y anclas con NCL [14]

1.4.1.3.2 Nodos de Composición

Un nodo de composición es un nodo que contiene recursivamente a otros nodos, ya sean nodos de contenido o de composición [11]. Como un nodo de composición es un contenedor de más nodos, requiere, como propiedad, de una entidad NCM denominada puerta que permite acceder a las entidades contenidas en el nodo. Un nodo de composición puede tener una puerta o una lista ordenada de puertas.

[20] Como por ejemplo un video.

Una puerta está compuesta por un nodo contenido dentro del nodo de composición y una interfaz de dicho nodo. En este enfoque, se entiende por interfaz a una propiedad de un nodo o un ancla (en el caso de un nodo de contenido) o una puerta o conjunto de puertas `switchPort` (en el caso de un nodo de composición) como se muestra en la Figura 1.9.

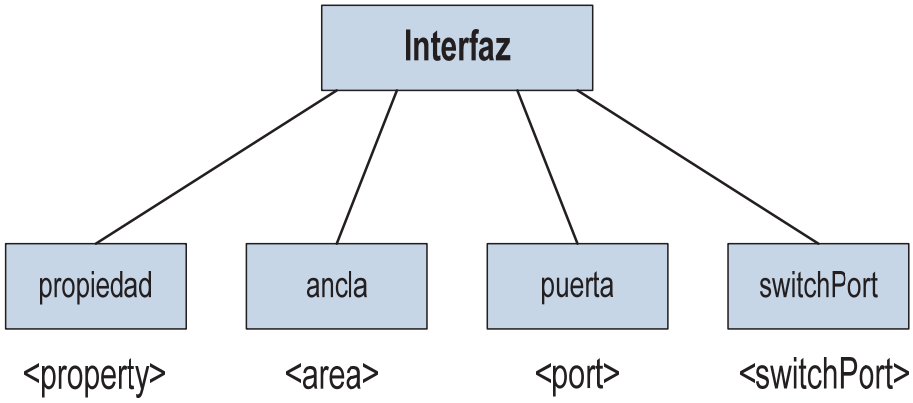


Figura 1.9 Interfaces de un nodo NCM [9]

Una puerta se define mediante la etiqueta `<port>` dentro de la definición del nodo de composición y tiene las siguientes propiedades [14]:

- **`id:`** Identificador único de la puerta.
- **`component:`** Identificador (`id`) del nodo asociado a la puerta**.**
- **`interface:`** Interfaz del nodo asociada a la puerta.

El modelo NCM establece cinco tipos de nodos de composición: `contexto`, `switch`, `trail`, `public hyperbase` y `private base` [11]. Sin embargo, los nodos de contexto y `switch` son los de mayor uso en la creación de un documento hipermedia.

Un nodo de contexto contiene recursivamente uno o más nodos de contenido, otros nodos de contexto o nodos `switch` [11]. Un nodo de contexto se define por la etiqueta `<context>` y permanece activo siempre que al menos uno de sus elementos esté activo.

La principal función de un nodo de contexto es determinar la estructura lógica, jerárquica o no, de un documento hipermedia [11]. Por lo tanto, los nodos de

contexto ayudan a organizar los elementos que conforman el documento hipermedia y permiten establecer diferentes vistas de dicho documento.

Un nodo `switch` es otra especialización de nodos de composición [11]. Este tipo de nodo puede contener recursivamente uno o más nodos de contexto y nodos de contenido, y es definido mediante la etiqueta `<switch>` [14].

El funcionamiento de un nodo `switch` se basa en la evaluación de reglas. Una regla es una entidad NCM definida por un elemento `<rule>` dentro de una base de reglas definida por la etiqueta `<ruleBase>` y retorna un valor booleano (verdadero o falso) en base a la evaluación de una expresión lógica . En caso de que la expresión se cumpla, el valor booleano retornado es verdadero y en caso contrario es falso. La Tabla 1.3 muestra los comparadores lógicos que pueden ser empleados por las reglas NCL para la evaluación de expresiones lógicas.

Tabla 1.3 Comparadores lógicos empleados en reglas NCL [14]

Comparador	Significado
eq	Igual a
ne	Diferente de
gt	Mayor que
ge	Mayor o igual que
lt	Menor que
le	Menor o igual que

La expresión lógica puede ser simple o compuesta. En este último caso se hace uso de los operadores lógicos AND y OR para unir dos o más expresiones simples [11].

Una regla define las siguientes propiedades [14]:

- **`id:`** Identificador único de la regla.
- **`var:`** Indica la propiedad que será evaluada.
- **`comparator:`** Indica el operador relacional empleado para la evaluación.
- **`value:`** Indica el valor a compararse con el que posee la propiedad en cuestión.

Para que un nodo `switch` seleccione un nodo a ser presentado, es necesario asociar una regla con un nodo a través de un elemento `<bindRule>` [14] dentro de la definición del nodo `switch`. Puede darse el caso de que más de una regla sea verdadera, pero estas son evaluadas de forma secuencial, de manera que la primera regla verdadera evaluada determinará el nodo que debe ser activado. Opcionalmente, se puede definir un nodo `default` que será activado en caso de que ninguna de las reglas evaluadas resulten verdaderas. Si todas las reglas son falsas y no se tiene un nodo `default`, entonces ninguno de los nodos será seleccionado [11].

En caso de que un nodo `switch` contenga nodos de contexto, deberá tener un elemento `<switchPort>` [14] para realizar el *mapping* de los nodos de contexto con sus respectivas puertas, de manera que cuando un nodo de contexto resulte ser elegido se active a través de la puerta indicada.

1.4.1.4 Enlaces

Un enlace es una entidad NCM que tiene dos propiedades adicionales: un conector y un conjunto de uniones [11]. El propósito de un enlace es el de relacionar los nodos en un documento hipermedia determinando cuándo deben ser presentados [4].

Un enlace está definido por la etiqueta `<link>` y está vinculado a un conector a través de la propiedad `xconnector` y define una unión a través de la etiqueta `<bind>` [14].

1.4.1.4.1 Conectores

Un conector representa el tipo de relación entre diferentes nodos participantes. Un mismo conector podría ser empleado varias veces por distintos participantes siempre que la relación entre ellos sea del mismo tipo y está conformado por un conjunto de roles, que son condiciones o acciones que se analizan o ejecutan sobre una interfaz de un determinado nodo [11].

La Figura 1.10 ilustra este concepto. Un mismo conector R es empleado dos veces, en primer lugar, para asociar las interfaces de los nodos A, B y C y en segundo para asociar las interfaces de los nodos B, C y D. Los roles del conector pueden ser condiciones o acciones que se ejecutan sobre las interfaces de los nodos involucrados (participantes) asociando cada interfaz con un rol del conector a través de una unión.

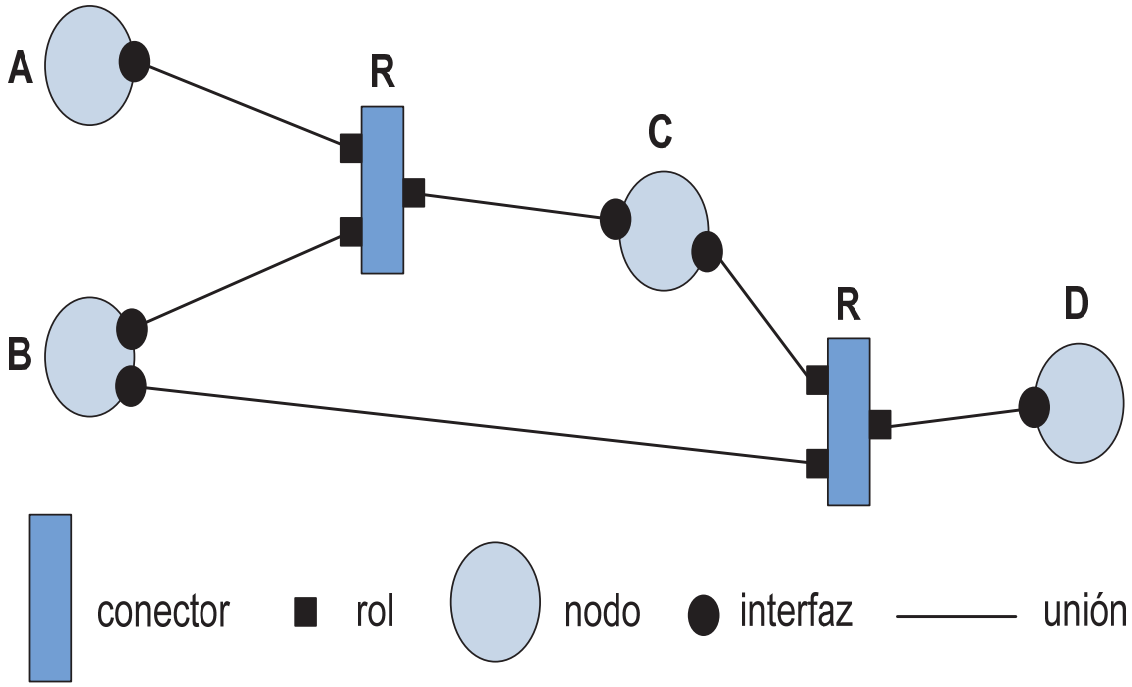


Figura 1.10 Enlaces entre nodos NCM [11]

Un conector puede ser de tipo causal o de restricción, pero los de tipo causal son los más utilizados en presentaciones hipermedia [11]. Un conector de tipo causal establece una condición que debe ser cumplida para que se ejecute una o más acciones [14] y está definido mediante la etiqueta `<causalConnector>` dentro de una base de conectores definida por la etiqueta `<connectorBase>` [14].

Una condición simple se define mediante la etiqueta `<simpleCondition>` dentro del conector y establece una condición simple que debe ser cumplida para que se ejecuten una o más acciones. Este elemento define, a través de su propiedad `role`, un nombre o condición que posteriormente se asociará a un participante a través de una unión.

Estas condiciones son [14]:

- **onBegin:** Se cumple cuando la presentación del participante asociado es iniciada.
- **onEnd:** Se cumple cuando la presentación del participante asociado es terminada.
- **onAbort:** Se cumple cuando la presentación del participante asociado es abortada.
- **onPause:** Se cumple cuando la presentación del participante asociado es pausada.
- **onResume:** Se cumple cuando la presentación del participante asociado es reiniciada después de haber sido pausada.
- **onSelection:** Se cumple cuando el usuario presiona una tecla determinada del control remoto.
- **onBeginAtribution:** Se cumple inmediatamente antes de que un valor sea atribuido a una propiedad del participante asociado.
- **onEndAtributtion:** Se cumple inmediatamente después de que un valor sea atribuido a una propiedad del participante asociado.

Además de una condición simple, un conector causal puede definir una condición compuesta a través de la etiqueta `<compoundCondition>`. Este elemento contiene dos o más condiciones simples unidas a través de los operadores lógicos AND y OR [11].

La etiqueta `<simpleAction>` define una acción a ser ejecutada cuando se activa un conector. Este elemento, a través de su propiedad `role`, define la acción o evento a ser ejecutado sobre un determinado participante asociado a través de una unión. NCL posee un conjunto de acciones que son [14]:

- **start:** Inicia la presentación del participante asociado.
- **stop:** Termina la presentación del participante asociado.
- **pause:** Pausa la presentación del participante asociado.
- **resume:** Reinicia la presentación del participante asociado.
- **set:** Establece un valor a una propiedad del participante asociado.

Además de una acción simple, un conector puede definir una acción compuesta a través de la etiqueta `<compoundAction>` [14]. Cuando se define una acción compuesta, se debe establecer una propiedad `operator` que recibe los valores `par` o `seq`, los cuales determinan si las acciones deben ser ejecutadas paralelamente o en secuencia, una detrás de otra respectivamente [11].

1.4.1.4.2 Uniones

La función de una unión es asociar un punto final (interfaz de un nodo) a un rol definido en el conector [11].

Se define una unión mediante la etiqueta `<bind>` y tiene las siguientes propiedades [14]:

- **`role:`** Identifica la condición o acción, establecida en el conector, siendo utilizada.
- **`component:`** Identifica, a través de su `id`, al nodo participante en la relación.
- **`interface:`** Identifica una propiedad o un ancla en un nodo de contenido, o una puerta o conjunto de puertas `switchPort` en un nodo de contexto.

1.4.1.5 Metadatos

Un metadato es un dato empleado para describir otro dato [2]. El uso de metadatos se basa en el uso de índices que permiten identificar y localizar recursos dentro de una estructura, de manera análoga a como el uso de fichas bibliográficas ayudan a describir y localizar libros dentro de una biblioteca.

En NCL, los metadatos funcionan como “datos sobre datos”, es decir, datos que ayudan a identificar, describir y localizar recursos dentro de un documento NCL [9]. NCL permite dos tipos de metadatos definidos por las etiquetas `<meta>` y `<metadata>` [14].

La entidad `<meta>` se emplea para metadatos simples y posee como atributos el nombre del dato a identificar (propiedad `name`) y una lista de valores asociados

con el dato que el metadato describe (propiedad `content`). La entidad `<metadata>` se emplea para establecer metadatos más estructurados. Estos metadatos son representados como un árbol[21], donde el elemento `<metadata>` actúa como la raíz [9].

En un documento NCL, los metadatos pueden ser asociados a una aplicación NCL como un todo o pueden también ser asociados únicamente a nodos de contexto.

1.4.2 ESTRUCTURA DE UN DOCUMENTO NCL

Todo el contenido de un documento NCL está anidado en la etiqueta raíz `<ncl>`, y su estructura está dividida en dos grandes partes: el encabezado y el cuerpo.

1.4.2.1 Encabezado

El encabezado del documento NCL define las características de presentación del documento a través de regiones, descriptores, reglas y conectores que serán utilizados para la sincronización de los nodos [14].

El encabezado está definido por la etiqueta `<head>` y puede tener como hijos los elementos definidos por las etiquetas `<importedDocumentBase>`, `<ruleBase>`, `<transitionBase>`, `<regionBase>`, `<descriptorBase>`, `<connectorBase>`, `<meta>` y `<metadata>` [10].

1.4.2.2 Cuerpo

En el cuerpo del documento NCL se definen, básicamente, los elementos y la sincronización entre ellos. Para mejorar su estructura y organización se recomienda establecer nodos de composición [14].

El cuerpo se define por la etiqueta `<body>` y puede tener como hijos los elementos definidos por las etiquetas `<port>`, `<property>`, `<media>`, `<context>`, `<switch>` y `<link>` [10].

---

[21] También llamado árbol RDF (*Resource Description Framework*). RDF es un modelo de metadatos flexible basado en XML [2].

Se considera al cuerpo como un tipo especial de nodo de contexto [14]. Una vez que se define el elemento `<body>` se crea automáticamente un nodo de contexto. Por lo tanto, para presentar un documento NCL, será necesario hacer uso de por lo menos una puerta que permita acceder a los nodos contenidos dentro del cuerpo del documento.

1.5 LENGUAJE DE PROGRAMACIÓN LUA [1], [14], [15], [16], [17], [21], [23]

Lua es un lenguaje de *scripting* imperativo, estructurado, de extensión y multiplataforma, diseñado con el objetivo principal de ser liviano y extensible para cualquier aplicación [15].

Al ser un lenguaje imperativo y estructurado, Lua emplea librerías, sentencias de control[22] e implementación de funciones para dar lugar a la solución de uno o más problemas específicos, haciendo uso de una sintaxis fácil de entender y bastante semejante a otros lenguajes de *scripting* [14]. Lua como lenguaje de extensión no tiene un programa principal sino que funciona embebido dentro de otra aplicación[23]. Finalmente, su característica multiplataforma le permite ser ejecutado sobre distintas arquitecturas de ordenadores y sistemas operativos [15].

El ambiente declarativo Ginga-NCL permite embeber objetos media de tipo Lua, también llamados objetos NCLua [14], dentro del documento NCL de manera que el programador tiene mayor control sobre la aplicación.

La diferencia entre un programa Lua puro y un programa NCLua radica en que este último es controlado por el formateador del documento NCL que establece cuándo debe ser ejecutado el código Lua [14].

1.5.1 MÓDULOS LUA

Las bibliotecas disponibles para Lua están organizadas de manera general en cuatro módulos. Cada módulo realiza tareas específicas a través de un conjunto de funciones [14]. Estos módulos son:

---

[22] Las sentencias de control permiten implementar algoritmos. Lua posee las sentencias `if then else`, `while`, `repeat`, `for` numérico y `for` genérico [14].
[23] También llamada programa anfitrión o simplemente anfitrión (*host*).

- **`Módulo Event:`** Permite la comunicación de los objetos NCLua con el formateador NCL y otras entidades externas a través de eventos, tales como los generados con el control remoto o a través del canal de retorno [14].
- **`Módulo Settings:`** Permite acceder a las variables definidas en un nodo de contenido de configuración (*settings*) definido en el documento NCL [14].
- **`Módulo Persistent:`** Ofrece una API para exportar una tabla con variables persistentes definidas en un área reservada para la manipulación de objetos imperativos [14].
- **`Módulo Canvas:`** Ofrece una API para diseñar objetos gráficos [21] y es uno de los más utilizados en *scripts* de Lua.

1.5.1.1 Módulo Canvas

Lua permite realizar operaciones gráficas durante la presentación de una aplicación tales como dibujar texto, líneas, imágenes y gráficas de forma general a través de su módulo `Canvas` [21].

Cuando se inicia un objeto NCLua, se crea de manera automática un objeto gráfico asignado a la variable global `canvas` [14]. Para ello, el objeto NCLua debe estar asignado previamente a una región específica porque de lo contrario el valor asignado a la variable `canvas` será de `nil`[24].

La Tabla 1.4 muestra algunas de las principales funciones del módulo `Canvas`. Una vez creado el objeto gráfico se puede hacer uso de estas funciones gráficas a través de su invocación mediante la variable global `canvas`, el operador dos puntos y el nombre de la función seguido de sus respectivos argumentos (indicados entre paréntesis), en caso de que la función así los requiera [21].

[24] En Lua un tipo de dato nulo (*null*) se denomina `nil` [15].

Tabla 1.4 Algunas funciones del módulo `Canvas` [14], [21]

Función	Propósito	Parámetros
`canvas:new(width,height)`	Crea un nuevo objeto gráfico del tamaño especificado.	`witdth`: [number] largo `height`: [number] ancho
`canvas:new(image_path)`	Una imagen es pasada como argumento.	`image_path`: [string] ruta hacia la imagen
`canvas:attrColor(R,G,B,A)`	Determina el color a través de sus componentes RGB y transparencia (parámetro A). Las componentes RGB pueden ser reemplazadas por el nombre de un color específico.	`R`: [number] comprendido entre 0-255 `G`: [number] comprendido entre 0-255 `B`: [number] comprendido entre 0-255 `A`: [number] comprendido entre 0-255
`canvas:attrFont(face,size,style)`	Establece el tipo de fuente.	`face`: [string] nombre de fuente `size`: [number] tamaño de fuente `style`: [string] estilo de fuente
`canvas:attrSize()`	Retorna las dimensiones de la región.	
`canvas:drawLine(x1,y1,x2,y2)`	Genera una línea.	`x1`: [number] componente en x del primer punto `y1`: [number] componente en y del primer punto `x2`: [number] componente en x del segundo punto `y2`: [number] componente en y del segundo punto
`canvas:drawText(x,y,text)`	Genera cómo será escrito un texto.	`x`: [number] coordenada horizontal `y`: [number] coordenada vertical `text`: [string] texto
`canvas:measureText(text)`	Retorna las dimensiones del texto.	`text`: [string] texto
`canvas:flush()`	Presenta en pantalla las operaciones especificadas con `Canvas`.	

1.5.2 CLIENTE WEB LUA

Lua puede ser utilizado para convertir al receptor del televidente en un cliente web capaz de solicitar determinados datos desde un servidor web remoto para visualizarlos, usando el canal de retorno y el protocolo HTTP (*HyperText Transfer Protocol*).

La Figura 1.11 muestra el mecanismo de comunicación entre un servidor web y el STB del usuario, actuando como un cliente web gracias a la funcionalidad implementada con Lua. En este esquema, el canal de retorno es un canal bidireccional en el que el cliente web realiza una petición del contenido que desea y el servidor web responde con dicho contenido. Como Lua es un lenguaje imperativo, el contenido puede ser adaptado en el receptor antes de ser presentado al televidente.

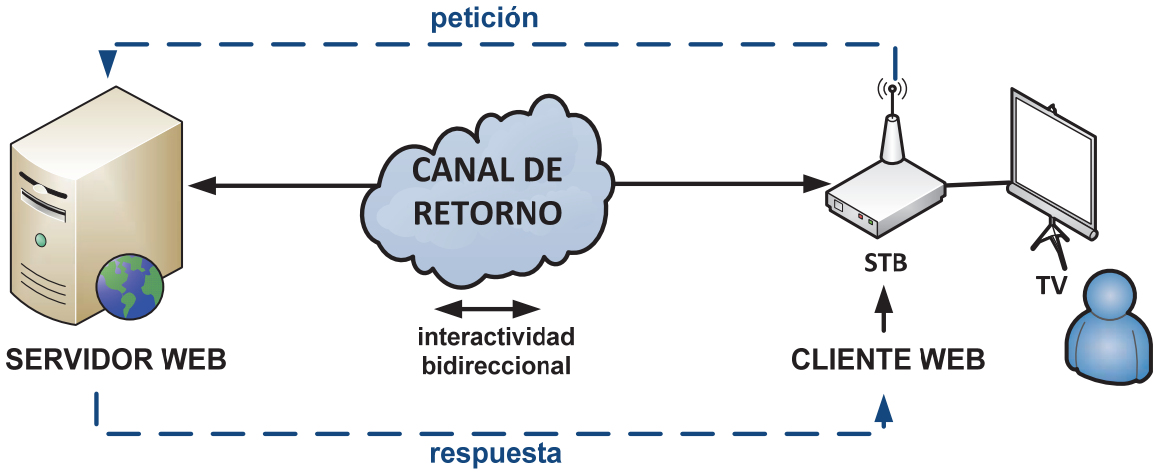


Figura 1.11 Comunicación entre servidor y cliente web Lua

1.5.2.1 Librería socket.http

La librería `socket.http` de Lua permite la comunicación a través del protocolo HTTP con un servidor web remoto convirtiendo a la aplicación Lua en un cliente web [1]. La Figura 1.12 muestra el mecanismo de comunicación entre cliente y servidor empleando la librería `socket.http`. La librería crea un *socket* TCP (*Transmission Control Protocol*) de manera que la aplicación pueda enviar peticiones HTTP y recibir la información correspondiente desde el servidor web.

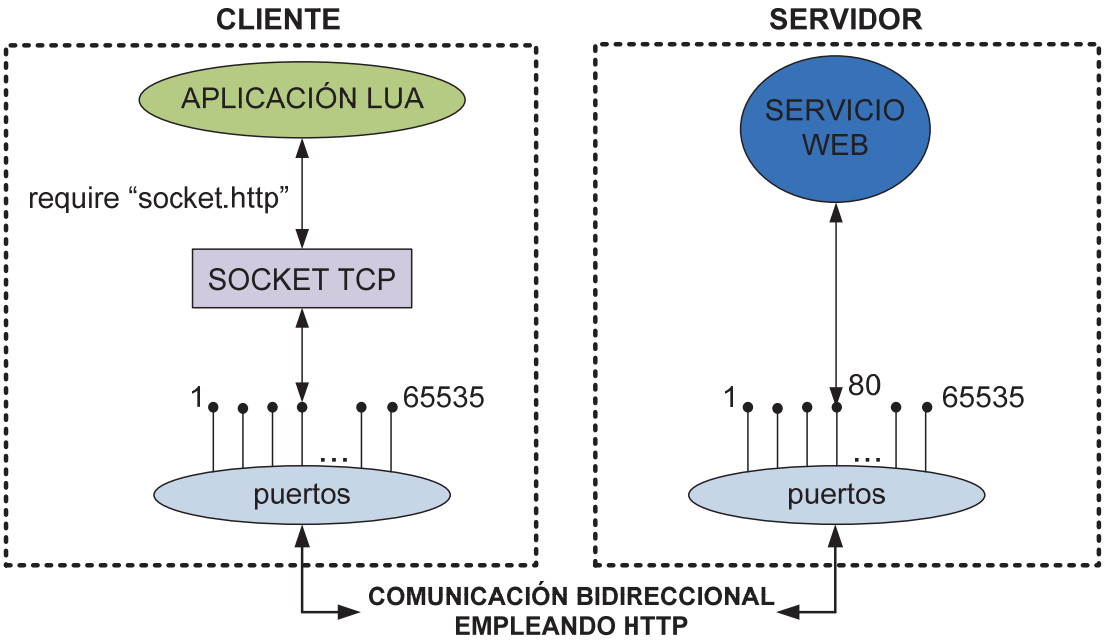


Figura 1.12 Comunicación entre cliente y servidor mediante la librería `socket.http`

Para poder hacer uso de esta librería, se debe cargar la misma a través de la función `require` de Lua [17]. El argumento de esta función es el nombre de la librería que se desea cargar para emplearla en el *script* Lua. Mediante la sentencia `require “socket.http”` se crea un objeto correspondiente a un *socket* TCP que se puede manipular mediante su asignación a una variable local [1].

Las peticiones HTTP se pueden realizar por medio de la función `request` del *socket* TCP [1]. La forma más sencilla de usar esta función es colocar como argumento de la misma, el URL (*Uniform Resource Locator*) del recurso al que se desea acceder en el servidor web remoto, de manera que se realice una operación HTTP GET [1]. La función `request` enviará la petición HTTP en busca del recurso indicado y retornará como resultado de la acción tres valores: el primer valor corresponde al recurso buscado, que en caso de no poder ser encontrado se establece en el valor `nil` [1]; el segundo valor corresponde al código de estado HTTP resultante de la operación [1], la Tabla 1.5 presenta los códigos de estado que se pueden obtener al intentar realizar una operación con HTTP; finalmente, el tercer valor retornado contiene los encabezados HTTP de la respuesta del servidor [1].

Tabla 1.5 Códigos de estado HTTP [23]

	Código	Descripción
1XX MENSAJES	100/111	Conexión rechazada
2XX OPERACIÓN EXITOSA	200	OK
	201-203	Información no oficial
	204	Sin contenido
	205	Contenido para recargar
	206	Contenido parcial
3XX REDIRECCIÓN	301	Mudado permanentemente
	302	Encontrado
	303	Vea otros
	304	No modificado
	305	Utilice un proxy
	307	Redirección temporal
4XX ERROR POR PARTE DEL CLIENTE	400	Solicitud incorrecta
	401	No autorizado
	402	Pago requerido
	403	Prohibido
	404	No encontrado
	409	Conflicto
	410	Ya no disponible
	412	Falló precondición
5XX ERROR DEL SERVIDOR	500	Error interno
	501	No implementado
	502	Pasarela incorrecta
	503	Servicio no disponible
	504	Tiempo de espera de la pasarela agotado
	505	Versión de HTTP no soportada

1.5.3 MANEJO DE EXCEPCIONES

Una excepción es un error que se produce en tiempo de ejecución. Cuando una aplicación interactiva ejecuta código Lua y se produce una excepción, la aplicación finaliza inmediatamente. Sin embargo, es posible evitar esta finalización repentina mediante el manejo de excepciones.

Lua permite manejar excepciones de manera explícita mediante la función `pcall`, para encapsular código que al ser ejecutado puede producir excepciones [16].

La función `pcall` tiene como argumento un segmento de código Lua a ser monitoreado mientras se ejecuta [16]. Este segmento de código puede ser pasado a `pcall` como el nombre de una función [14]. Si el segmento de código indicado

se ejecuta sin ningún error, `pcall` retorna un valor booleano de `true` y los valores que la función indicada retorne a través de la sentencia `return` [16]. Si por el contrario se produce un error al ejecutar el segmento de código Lua, `pcall` retornará un valor booleano de `false` y el respectivo mensaje de error permitiendo al programador establecer las acciones necesarias para controlar el error generado [16].

1.6 SERVICIOS WEB WINDOWS COMMUNICATION FOUNDATION (WCF) [3], [7], [13]

Un servicio web es un mecanismo de comunicación distribuido en el que una aplicación cliente accede a las funcionalidades ofrecidas por una aplicación servidor a través de un protocolo sobre una red de comunicaciones [3].

El funcionamiento de un servicio web se basa en una tecnología denominada *Remote Procedure Call* (RPC) o Invocación Remota de Métodos y se describe en la Figura 1.13. La aplicación cliente envía al servicio web peticiones de ejecución de métodos con sus argumentos. El servicio web interpreta la petición, ejecuta el método pedido con los argumentos enviados y retorna los resultados obtenidos a la aplicación cliente. Por lo tanto, la funcionalidad se implementa por completo en el lado del servidor y el cliente se limita tan solo a realizar peticiones de la funcionalidad que desea que el servicio realice.

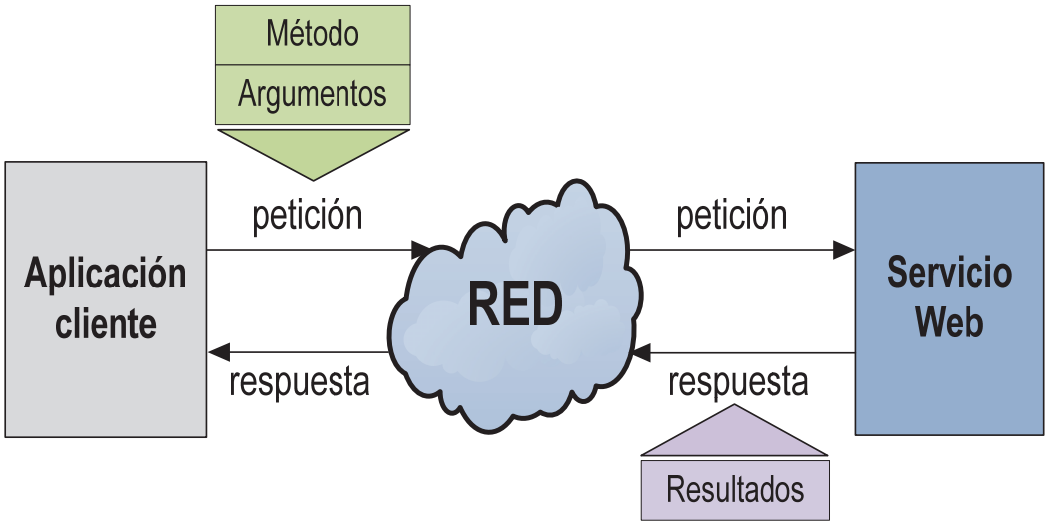


Figura 1.13 Interacción entre aplicación cliente y servicio web a través de RPC

Existe una amplia gama de herramientas para poder construir servicios web. *Windows Communication Foundation* (WCF) de Microsoft es una de ellas. Si bien es posible crear servicios web con muchas otras tecnologías, propietarias o no de Microsoft, la implementación de servicios web con WCF es mucho más sencilla pues nació para brindar un enfoque manejable, unificando las tecnologías de Microsoft[25] y permitiendo la interoperabilidad con otras tecnologías de terceros [7].

WCF es una plataforma que permite la creación de servicios web empleando tecnologías de comunicación como SOAP (*Simple Object Access Protocol*) y REST (*Representational State Transfer*) [13].

1.6.1 COMPONENTES DE UN SERVICIO WEB WCF

Un servicio web WCF tiene tres componentes: *address*, *binding* y *contract*[26].

- ***Address* (Dirección):** Una dirección representa el punto final a través del cual se accede al servicio web [7]. Este punto de acceso incluye el protocolo de comunicaciones empleado (por ejemplo HTTP) y la dirección de red del servidor que brinda el servicio [13].
- ***Binding* (Unión):** Una unión especifica cómo un cliente web se comunica con el servicio (a través de SOAP, REST o ambos) [13]. Además, una unión puede ser utilizada para establecer restricciones, como por ejemplo en lo que concierne a seguridad [7].
- ***Contract* (Contrato):** Un contrato es una interfaz que representa los métodos que el servicio web implementa y los tipos de datos retornados por cada uno de dichos métodos [13]. Gracias al contrato un cliente web puede interactuar con el servicio web [7].

1.6.2 SIMPLE OBJECT ACCESS PROTOCOL (SOAP)

SOAP es un protocolo que emplea XML como formato de datos para realizar la invocación remota de métodos típicamente mediante HTTP [13].

---

[25] Ejemplos de estas tecnologías de Microsoft son ASP.NET, *Enterprises Services* (COM+) y *Web Service Enhacements* (WSE) [7].
[26] Llamados el ABC de un servicio WCF [13].

Cada petición y respuesta son envueltas en un mensaje SOAP en formato XML, legibles para el ordenador y para el usuario e independientes de la plataforma. Sin embargo, la envoltura SOAP es demasiado compleja lo cual puediese limitar el uso de esta tecnología para la comunicación entre cliente y servidor [7].

La principal ventaja de SOAP es que facilita la implementación de la aplicación cliente que consume el servicio web a través de un *proxy* que permite ejecutar métodos remotos como si se ejecutasen localmente. El *proxy* es un objeto que se crea en el lado de la aplicación cliente como una referencia al servicio web, de manera que cuando se requiera invocar a un método del servicio, el programador únicamente debe acceder a dicho método a través del objeto *proxy*, al igual como se accede a métodos de cualquier otro objeto local, y el *proxy* se encarga de realizar la invocación remota del método al servicio web y de recibir la respuesta de manera totalmente transparente, como se muestra en la Figura 1.14.

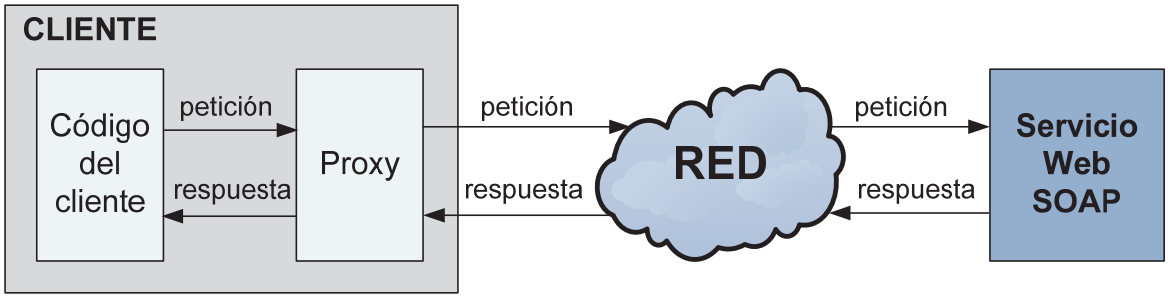


Figura 1.14 Interacción entre un cliente y un servicio web basado en SOAP [13]

La mayoría de *firewalls* en Internet permiten el tráfico HTTP, por lo que los servicios web basados en SOAP pueden enviar y recibir mensajes SOAP sobre conexiones HTTP con pocas limitaciones.

1.6.3 REPRESENTATIONAL STATE TRANSFER (REST)

REST es una arquitectura utilizada para implementar servicios web. Los servicios creados en base a esta arquitectura son comúnmente denominados servicios RESTful [13]. En un servicio RESTful cada método del servicio web es identificado por un URL único.

A diferencia de los servicios web implementados con SOAP, los servicios RESTful no requieren de una envoltura especial para la comunicación y además no están

limitados a retornar datos únicamente en formato XML sino que pueden hacer uso de otros formatos como JSON (*JavaScript Object Notation*), HTML (*HyperText Markup Language*), texto plano e inclusive archivos multimedia [13]. Estas características hacen que los servicios RESTful sean ideales para la comunicación con el receptor del usuario a través del canal de retorno. Sin embargo, la implementación de la aplicación cliente puede llegar a ser complicada puesto que se requiere manejar espacios de nombres URL para la invocación de los métodos.

REFERENCIAS

[1] D. Nehab. (2006, Abril) LuaSocket:HTTP Support. [Online]. Disponible en: http://w3.impa.br/~diego/software/luasocket/http.html (Consultado el 18 de septiembre de 2013).

[2] E. Méndez Rodríguez, "RDF: Un modelo de metadatos flexible", Dpto. de Biblioteconomía y Documentación Universidad Carlos III de Madrid, Madrid.

[3] G. González. Servicios Web. [Online]. Disponible en: http://kalistog.wordpress.com/servicios-web (Consultado el 20 de septiembre de 2013).

[4] Ginga Bolivia. NCM 3.0 (Nested Context Model, Modelo de Contexto Anidado). [Online]. Disponible en: http://ginga.softwarelibre.org.bo/lib/exe/fetch.php?media=ncm.pdf (Consultado el 8 de septiembre de 2013).

[5] Git Foundaton. Git distributed even if your workflow isnt Documentation. [Online]. Disponible en: http://git-scm.com/documentation (Consultado el 12 de septiembre de 2013).

[6] J. Blanchette y M. Summerfield, *C++ GUI Programming with Qt 4*, Primera edición ed., Trolltech, Ed. Stoughton, USA:Prentice Hall, 2006.

[7] J. Conesa Caralt, A. Rius Gavidia, J. Ceballos Villach, y D. Gañán Jiménez, *Introducción a .NET*, Primera edición ed., Sónia Poch, Ed,: UOC, 2010.

[8] J. Torres. (2011, Junio) Comunidad Ginga Ecuador. [Online]. Disponible en: http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html (Consultado el 8 de septiembre de 2013).

[9] L. F. Gomes Soares y S. D. Junqueira Barbosa, *Programando em NCL 3.0*, Segunda edición ed., PUC-Rio, Ed. Río de Janeiro, Brasil, 2012.

[10] L. F. Gomes Soares, R. Ferreira Rodrigues, R. Rezende Costa, y M. Ferreira Moreno, "Nested Context language 3.0 Part 9 - NCL Live Editing Commands", Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Monografía en Ciencia de Computación ISSN 0103-9741.

[11] L. Gomes y R. Ferreira, "Nested Context Model 3.0 Part 1 - NCM Core", Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Monografía en Ciencia de Computación ISSN 0103-9741.

[12] Microsoft. (2013) Fundamentos de la depuración. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/aa290881(v=vs.71).aspx (Consultado el 20 de septiembre de 2013).

[13] P. Deitel y H. Deitel, *C# 2010 for programmers*, Cuarta edición ed., RR Donnelley, Ed. Crawfordsville, USA: Prentice Hall, 2010.

[14] R. Carvalho, J. A. Ferreira dos Santos, J. Ribeiro Damasceno, J. Varanda Da Silva, y D. C. Muchaluat Saade, *Introducao ás Linguagens NCL e Lua: Desenvolvendo Aplicacoes Interactivas para TV Digital*, Primera edición ed., Universidad Federal Fluminense, Ed. Brasil: Laboratorio MídiaCom y Peta5, 2009.

[15] R. Ierusalimschy, L. Henrique de Figueiredo, y W. Celes. (2011, Septiembre) Manual de Referencia de Lua 5.1. [Online]. Disponible en: http://www.lua.org/manual/5.1/es/manual.html (Consultado el 18 de septiembre de 2013).

[16] R. Ierusalimschy. Lua Programming Error Handling and Exceptions. [Online]. Disponible en: http://www.lua.org/pil/8.4.html (Consultado el 19 de septiembre de 2013).

[17] R. Ierusalimschy. Lua Programming The require Function. [Online]. Disponible en: http://www.lua.org/pil/8.1.html (Consultado el 19 de septiembre de 2013).

[18] Superintendencia de Telecomunicaciones del Ecuador. (2011, Noviembre) Preguntas frecuentes sobre Televisión Digital. [Online]. Disponible en: http://www.supertel.gob.ec/index.php?option=com_content&view=article&id=211:preguntas-frecuentes-sobre-television-digital&catid=61:articulos-recomendados&Itemid=311 (Consultado el 6 de septiembre de 2013).

[19] TeleMídia (PUC-Rio). How to: build NCL Composer from source code. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/doc/tutorial/how_to_build_ncl_composer_from_source_code (Consultado el 14 de septiembre de 2013).

[20] TeleMídia (PUC-Rio). NCL Composer Plugins. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/plugins/start (Consultado el 14 de septiembre de 2013).

[21] TeleMídia (PUC-Rio). Lua Módulo canvas. [Online]. Disponible en: http://www.telemidia.puc-rio.br/~francisco/nclua/referencia/canvas.html (Consultado el 16 de septiembre de 2013).

[22] TeleMídia (PUC-Rio). Welcome to NCL Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/start?redirect=1#welcome_to_ncl_composer (Consultado el 14 de septiembre de 2013).

[23] Wikipedia. HypertextTransferProtocol. [Online]. Disponible en: http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol (Consultado el 16 de septiembre de 2013).

CAPÍTULO 2

DISEÑO E IMPLEMENTACIÓN DE UN PLUG-IN DE GENERACIÓN AUTOMÁTICA DE MENÚS PARA EL IDE NCL COMPOSER

En este capítulo se especifica el diseño e implementación de un *plug-in,* para el Entorno de Desarrollo Integrado (IDE) NCL Composer, capaz de generar menús con características y funcionalidades personalizables. El *plug-in* ha sido nombrado Menu Creator y se desarrolló empleando la biblioteca multiplataforma Qt con su lenguaje nativo C++.

El *plug-in* brinda una interfaz gráfica amigable al usuario para diseñar menús y genera el código en lenguaje NCL de los menús diseñados de forma automática. De esta manera, el usuario puede crear aplicaciones interactivas para televisión digital que requieran menús de manera simple, en menor tiempo y sin necesidad de tener conocimientos avanzados del lenguaje NCL.

2.1 IDE NCL COMPOSER [26], [33]

NCL Composer es el IDE para el desarrollo de aplicaciones interactivas con Ginga-NCL y fue creado haciendo uso de la biblioteca multiplataforma Qt y el lenguaje de programación C++.

Para facilitar el trabajo de usuario, NCL-Composer hace uso de *plug-ins*, los cuales son bibliotecas de enlace dinámico[27] que se cargan dentro de la ventana principal del IDE en tiempo de ejecución. Cada *plug-in* hace uso de una subventana que se presenta dentro de la ventana principal de NCL Composer, la cual brinda una interfaz gráfica al usuario para realizar determinadas acciones en la creación de una aplicación interactiva, como se presenta en la Figura 2.1.

[27] Una biblioteca de enlace dinámico es un archivo con código ejecutable que se carga bajo demanda de un programa [33]. En Windows este archivo es de extensión .dll (*dynamic-link library*) mientras que en Linux y Macintosh es de extensión .so (*shared object*).

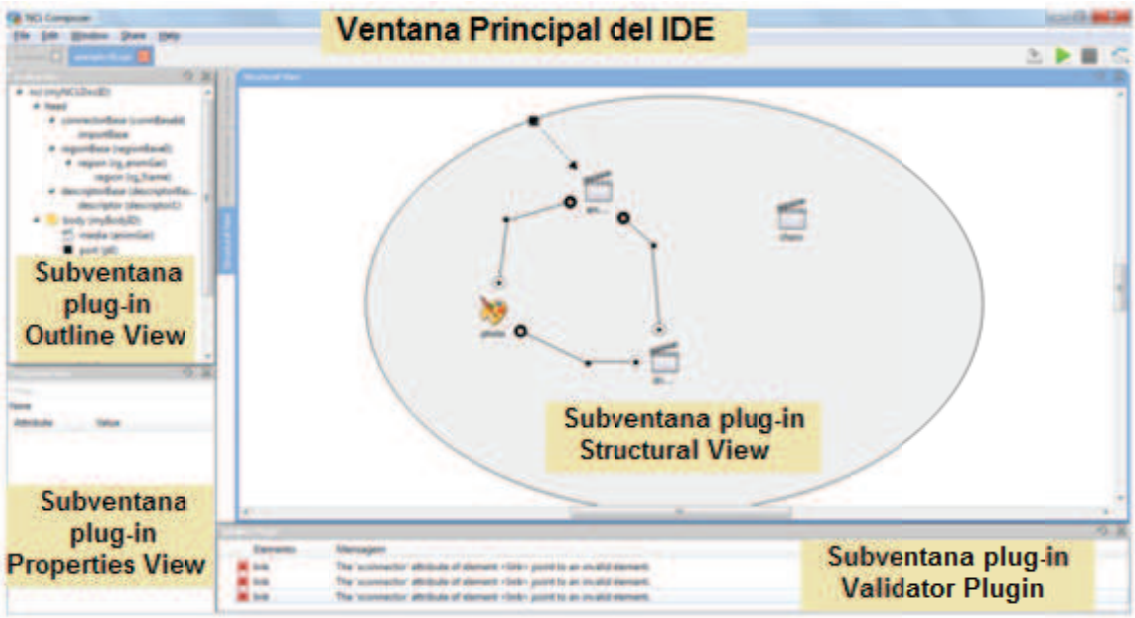


Figura 2.1 Ventana principal de NCL Composer y subventanas de *plug-ins* [26]

El usuario puede observar los *plug-ins* que han sido cargados en el IDE, así como activar o desactivar cada uno de ellos, desde el diálogo *Help/About Plugins* que se muestra en la Figura 2.2.

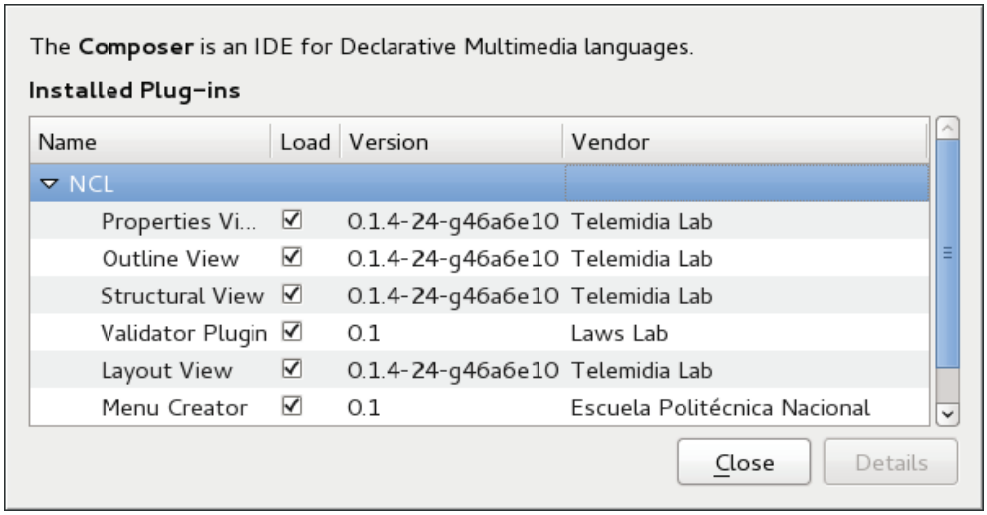


Figura 2.2 Diálogo *Help/About Plugins* de NCL Composer

2.1.1 COMPILACIÓN E INSTALACIÓN [9], [14], [18], [35]

Como se mencionó en el Capítulo 1, el IDE NCL Composer es de código abierto y sus archivos fuente están disponibles en [18].

NCL Composer requiere para su compilación la biblioteca multiplataforma Qt, con su lenguaje nativo C++, que son los recursos con los que fue desarrollado originalmente. Si el IDE va a ser compilado en Windows, se requiere instalar además MinGW[28] para dar soporte a C++; se recomienda la descarga de Qt SDK que ofrece todo el *framework* de instalación para desarrolladores de Qt en un solo paso. El SDK incluye la biblioteca Qt y una versión adecuada de MinGW para trabajar con la versión de Qt que instala el paquete. Este *software* se puede descargar de la página oficial de Nokia disponible en [9] previo registro.

La Figura 2.3 muestra una imagen del instalador de Qt SDK en un solo paso.

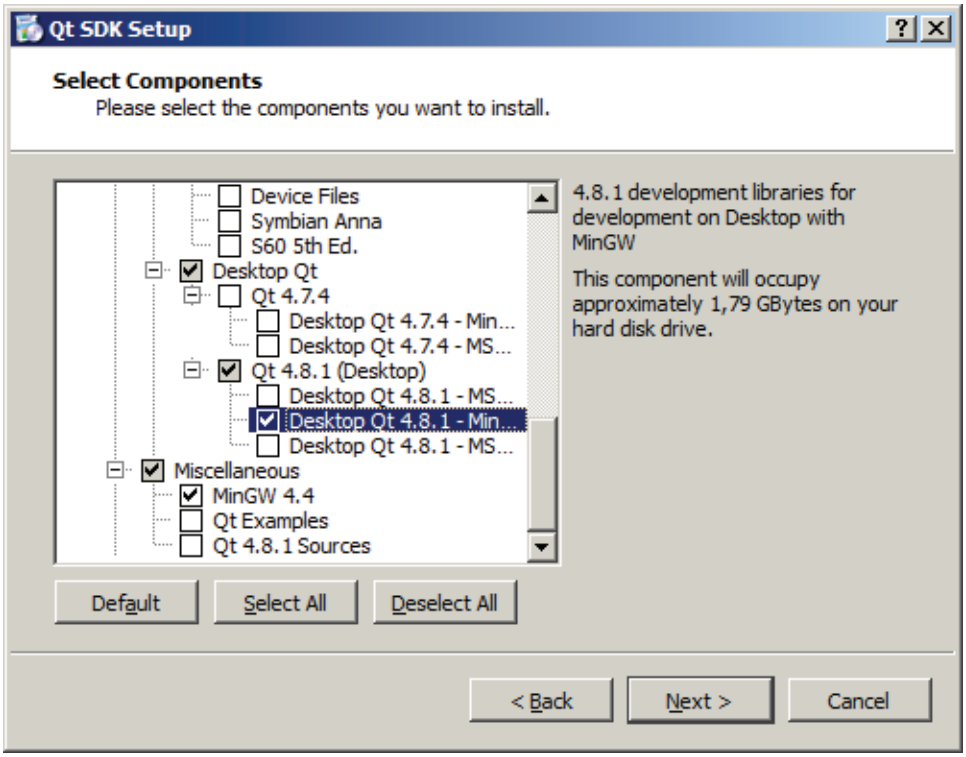


Figura 2.3 Instalador de Qt SDK

Para hacer uso de un proyecto desarrollado en Qt, desde la línea de comandos y situado en el directorio que contiene el proyecto, se debe ejecutar en un orden específico los comandos que permitan configurar, compilar e instalar dicho proyecto.

[28] MinGW (*Minimalist GNU for Windows*) es una implementación de los compiladores GCC (*GNU Compiler Collection*) para la plataforma Win32, que permite migrar la capacidad de este compilador a entornos Windows [35].

En Linux, estos comandos se presentan en el Código 2.1. Puede ser necesario hacer uso del último comando con `sudo`.

```
$ qmake
$ make
$ [sudo] make install
```

Código 2.1 Comandos para compilar e instalar NCL Composer en Linux [14]

Para proporcionar el soporte del lenguaje C++ en Windows, los dos últimos comandos cambian, como lo sugiere el Código 2.2.

```
$ qmake
$ mingw32-make
$ mingw32-make install
```

Código 2.2 Comandos para compilar e instalar NCL Composer en Windows [14]

En Macintosh, el primer comando se cambia y puede ser necesario, al igual que en Linux, hacer uso del último comando con `sudo` como sugiere el Código 2.3.

```
$ qmake –spec macx-g++
$ make
$ [sudo] make install
```

Código 2.3 Comandos para compilar e instalar NCL Composer en Macintosh [14]

El módulo fuente de NCL Composer viene dado por tres proyectos desarrollados en Qt:

- composer-core
- composer-gui
- composer-plugins

Cada componente se ubica en una carpeta distinta; por lo tanto, cada uno de estos componentes debe ser compilado e instalado ejecutando los comandos anteriormente descritos.

El composer-core comprende un proyecto compuesto por dos subproyectos:

- **core:** Es el núcleo de NCL Composer, cuya funcionalidad es gestionar los *plug-ins*, el árbol jerárquico de entidades, los proyectos de usuario y demás componentes del IDE. Sin embargo, no provee el analizador que permite dar soporte al lenguaje declarativo (NCL, SMIL o XHTML) a ser empleado por el IDE.
- **NCLLanguageProfile:** Contiene un analizador que da soporte al lenguaje declarativo NCL. El IDE NCL Composer trabaja con el lenguaje NCL por defecto, pudiéndose extender su funcionalidad para trabajar con otros lenguajes declarativos como SMIL y XHTML.

La ejecución de los comandos antes descritos sobre el proyecto composer-core, compila e instala ambos subproyectos y se obtiene como resultado dos bibliotecas de enlace dinámico. La primera corresponde al *core* y la segunda al analizador del lenguaje NCL. En la Figura 2.4 se muestran las dos bibliotecas obtenidas en Windows 7 después del proceso de compilación e instalación.

(a) (b)

Figura 2.4 Bibliotecas obtenidas por compilación e instalación del composer-core en Microsoft Windows 7: (a) Core (b) Analizador del lenguaje NCL

Por otro lado, al compilar e instalar la interfaz gráfica de usuario de NCL Composer (proyecto composer-gui), se obtiene el archivo de programa ejecutable composer.exe. Además, se compila el soporte del cliente SSH[29] para permitir a un usuario conectarse a un simulador de Ginga vía una máquina virtual, e instala el archivo defaultConnBase.ncl que ofrece varios conectores causales a disposición del usuario. En la Figura 2.5 se presenta el resultado obtenido después de la compilación e instalación del proyecto composer-gui.

[29] *Secure Shell* (SSH) es un protocolo que permite el acceso remoto a máquinas, reales o virtuales, empleando técnicas de cifrado que logran que la información viaje segura.

(a) (b)

Figura 2.5 Resultados obtenidos de la compilación e instalación del composer-gui: (a) Archivo ejecutable composer.exe (b) Archivo defaultConnBase.ncl

Finalmente, cada *plug-in* debe compilarse e instalarse para obtener una biblioteca de enlace dinámico. Para los *plug-ins* originales se ofrece la opción de compilación e instalación en un solo paso en la carpeta del proyecto composer-plugins. Esta opción consiste en una única llamada de compilación e instalación de todos los *plug-ins* contenidos en el proyecto composer-plugins. El resultado es un conjunto de bibliotecas de enlace dinámico, una por cada *plug-in*, que al instalarse se copian en la carpeta *extensions* de NCL Composer. Cuando el IDE se ejecute reconocerá automáticamente los *plug-ins* instalados.

Finalmente, en la Figura 2.6 se detalla la estructura del IDE NCL Composer. El *core* gestiona los *plug-ins* y los analizadores de lenguaje implementados. El árbol jerárquico de entidades es parte del *core* y es el único componente que puede gestionar el árbol directamente. El *core* es además usado por la interfaz gráfica de usuario (composer-gui), que corresponde a la ventana principal del IDE en la que se cargan los *plug-ins*. La interfaz gráfica de cada *plug-in* (subventana dentro de la ventana principal de NCL Composer) debe ser implementada dentro del código de cada *plug-in* pues no es parte del composer-gui.

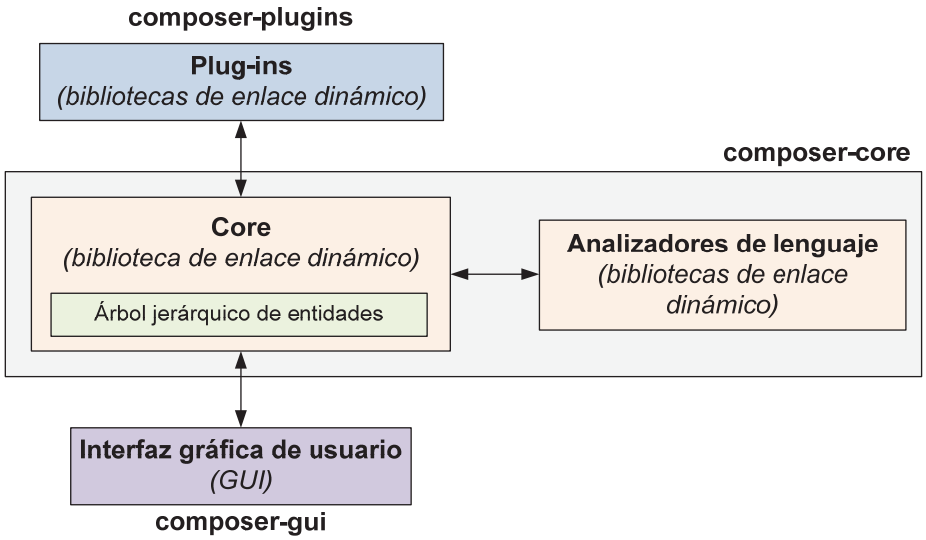


Figura 2.6 Estructura de NCL Composer

2.2 BIBLIOTECA MULTIPLATAFORMA Qt [3], [34]

La biblioteca multiplataforma Qt es un *framework* de desarrollo escrito en C++[30] para desarrolladores multiplataforma [3]. Su API cuenta con métodos que permiten crear interfaces gráficas de usuario, acceder a bases de datos mediante sentencias SQL, uso de XML, gestión de hilos, comunicación en red, manipulación de archivos, entre otras funcionalidades [34].

La implementación del *framework* se basa en un conjunto de clases que se asocian mediante un mecanismo de herencia. La clase `QObject` es la clase base de la que heredan todas las demás. Este mecanismo permite mayor versatilidad sobre la programación del *framework* debido a que las clases que heredan pueden hacer uso de los métodos de sus clases antecesoras, lo que favorece el reuso de código en el API. En la Figura 2.7 se presenta el modelo de clases de Qt. Por ejemplo, la clase `QPushButton` (que implementa un botón gráfico) no tiene un método `show()` para mostrarlo en pantalla, pero su clase antecesora `QWidget` sí implementa este método, por lo que puede ser usado por objetos de la clase `QPushButton` y por otros objetos gráficos (*Widgets*) de clases como `QSpinBox`, `QSlider`, `QLabel`, entre otras [3].

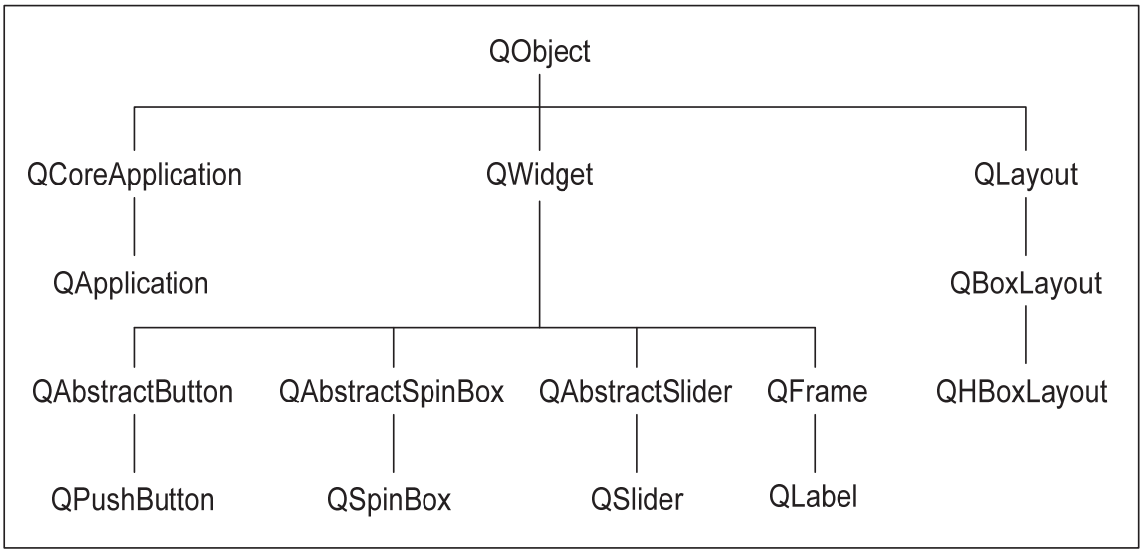


Figura 2.7 Modelo de clases de la biblioteca multiplataforma Qt [3]

[30] Qt hace uso del lenguaje de programación C++ de forma nativa y extiende su funcionalidad para dar soporte a otros lenguajes, como Phyton, Perl, PHP y otros mediante el uso de *bindings* [34].

2.2.1 MECANISMO SIGNALS AND SLOTS [3], [4], [8], [12]

El mecanismo *signals and slots* es la característica central de la biblioteca Qt y probablemente su principal distintivo de otros *frameworks* de desarrollo de aplicaciones [8]. Este mecanismo permite al programador de la aplicación comunicar objetos (instancias de distintas clases) sin la necesidad de que dichos objetos sepan uno del otro [3]. Aunque este mecanismo puede ser utilizado como se requiera, es principalmente usado para la comunicación entre la interfaz gráfica de usuario y las clases que implementan su funcionalidad (programación dirigida por eventos).

El mecanismo *signals and slots* es una de las maneras más flexibles de comunicar objetos y hace el código fácil de diseñar y de reusar [4]. Cuando un objeto requiere comunicarse con otros basta con que este emita (`emit`) una señal (*signal*) que es recibida por otros objetos a la escucha que ejecutan un *slot* al recibir la señal. La Figura 2.8 ilustra la comunicación entre objetos empleando el mecanismo *signals and slots*. Los objetos 1 y 3 emiten señales (objetos emisores) que son recibidas por los objetos 2 y 4 (objetos receptores). Tal como sugiere la figura, para que el mecanismo sea factible, las clases de los objetos emisores deben declarar las señales que emiten, y las clases de los objetos receptores deben declarar e implementar los *slots* que se ejecutan en respuesta a las señales que se reciben.

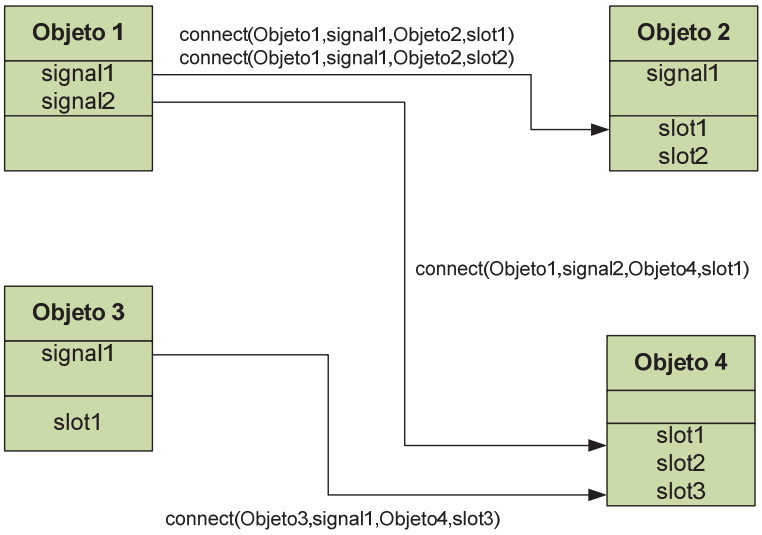


Figura 2.8 Comunicación entre objetos con el mecanismo *signals and slots* [12]

El objeto que recibe la señal sabe qué *slot* ejecutar mediante la sentencia de conexión (`connect`), la cual asocia al objeto emisor y su señal con el objeto receptor y su *slot*. Las sentencias `connect` no necesariamente deben ser declaradas en la clase que implementa el *slot* sino en cualquier clase, ya que cuando una señal es emitida, esta es receptada por todos los objetos y procesadas únicamente por aquellos que implementan una conexión (`connect`) para dicha señal.

Si no se especifica un objeto receptor en la sentencia `connect`, se asume que este objeto corresponde a la instancia de la clase en la que se implementa la sentencia (representado por la palabra reservada `this`). Por otra parte, el mecanismo permite especificar un objeto receptor de una clase diferente a la que implementa la sentencia `connect`.

Tanto las señales como los *slots* se declaran igual que un método[31], mientras que la declaración de la conexión (`connect`) especifica, para las señales y *slots*, únicamente el tipo de dato. Cuando se hace uso de la sentencia `emit` desde la clase emisora de la señal, debe detallarse el nombre de la señal que se emite y los argumentos correspondientes a la señal. La Figura 2.9 muestra un ejemplo.

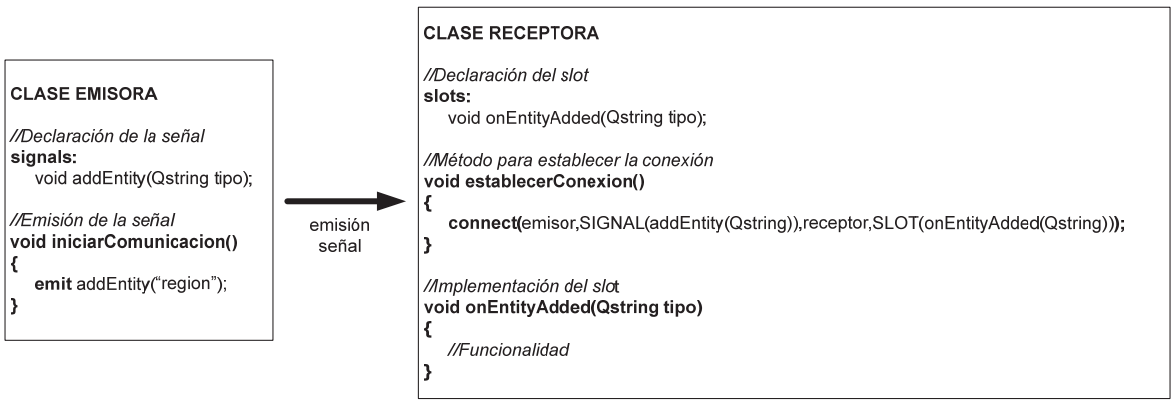


Figura 2.9 Ejemplo de implementación del mecanismo *signals and slots*

Existe cierta flexibilidad a la hora de implementar el mecanismo *signals and slots* pudiendo tenerse entre otras posibilidades [3]:

[31] Al ser declaradas al igual que un método, las señales y *slots* deben especificar sus parámetros, identificados a través de un nombre de variable, y el tipo de dato de dichos parámetros.

- Una señal puede ser conectada a muchos *slots*.
- Diferentes señales pueden ser conectadas al mismo *slot*.
- Una señal puede ser conectada a otra señal, de manera que cuando la primera se emite provoca la emisión de la segunda. Esto se realiza especificando en lugar de un *slot* una señal.

La comunicación entre los objetos puede ser eliminada mediante la sentencia `disconnect`, seguida de los mismos parámetros de creación de la conexión. Esta sentencia es poco utilizada, pues la conexión se elimina automáticamente cuando se destruye un objeto que interviene en ella [3].

Existen algunas consideraciones a tener en cuenta a la hora de implementar el mecanismo *signals and slots*. Para que una señal se conecte a un *slot*, ambos deben tener los mismos tipos de parámetros declarados en el mismo orden. El mecanismo permite una excepción, en la que si una señal posee un número mayor de parámetros de los que implementa el *slot*, los parámetros restantes se omiten [3]. Esto es una ventaja ya que evita una dependencia en el número de parámetros de señales y *slots*, a diferencia del caso tradicional de comunicación entre objetos, en el cual para que un objeto de la clase emisora invoque directamente a un método de un objeto de la clase receptora, debe cumplir con el número de parámetros del método que pretende invocar.

2.2.2 IMPLEMENTACIÓN DE CLASES EN C++ [2], [5], [10], [32]

C++ es un lenguaje de programación que mejora las características del lenguaje C al brindar soporte al paradigma orientado a objetos [10].

Una clase en C++ se implementa mediante dos archivos: un archivo de cabecera (de extensión .h) y un archivo de implementación de funcionalidad (de extensión .cpp).

Los archivos de cabecera, conocidos también como ficheros de inclusión, contienen una declaración de la clase a implementarse (atributos y métodos) [32], además de otros identificadores tales como sentencias de preprocesador [2]. Es

por ello que es común referirse a estos archivos como interfaces, ya que definen la estructura con la cual se implementan las clases.

Un archivo de extensión .cpp implementa la funcionalidad de la clase. Sin embargo, no está obligado a implementar todos los métodos declarados en su archivo cabecera.

Los archivos de cabecera también pueden incluir la declaración explícita de los métodos, permitiendo transferir estas definiciones a los archivos de extensión .cpp. Si se define un método de una clase como tipo `inline` se evita que el archivo de extensión .cpp pueda sobreescribir la definición ya implementada en el archivo de cabecera [5].

2.3 COMPOSER-CORE

El composer-core es el elemento principal del IDE NCL Composer ya que implementa las reglas de negocio para la generación del documento NCL.

2.3.1 MECANISMOS DE COMUNICACIÓN CON EL COMPOSER-CORE

2.3.1.1 Mecanismo Signals and Slots

Un *plug-in* en NCL Composer debe ser capaz de gestionar el árbol jerárquico de entidades para permitir al usuario generar el documento NCL evitándole la tarea de programar directamente el código y brindándole una interfaz gráfica amigable. Sin embargo, un *plug-in* bajo ninguna circunstancia puede acceder al árbol jerárquico de entidades y manipularlo directamente, ya que estas tareas únicamente las puede realizar el composer-core.

Para que un *plug-in* pueda gestionar el árbol jerárquico de entidades debe invocar la lógica implementada en el composer-core mediante el mecanismo *signals and slots*.

Al abrir el IDE, NCL Composer crea una instancia del composer-core y una instancia del composer-gui. La instancia del composer-core contiene toda la funcionalidad para gestionar varios proyectos Composer creados por el usuario,

mientras que la instancia del composer-gui ofrece la interfaz gráfica al usuario (ventana principal del IDE).

Cuando se crea un nuevo proyecto en NCL Composer, se crea una instancia de cada *plug-in* activado con el fin de que el usuario pueda invocar su funcionalidad haciendo uso de la interfaz gráfica de dicho *plug-in*. Además, se crea un archivo (documento NCL) en donde se almacenará el código NCL que es generado haciendo uso de los *plug-ins*, de acuerdo a las indicaciones del usuario. Por otro lado, también se crea un archivo en donde se almacenarán los datos del proyecto Composer con el trabajo del usuario y el estado de los *plug-ins* utilizados (archivo de extensión .cpr).

El mecanismo *signals and slots* es utilizado por NCL Composer para permitir que el objeto que desea comunicarse (en este caso una instancia del *plug-in*) pueda emitir (`emit`) una señal (*signal*) y el objeto que recibe dicha señal (en este caso una instancia del composer-core) pueda conectar (`connect`) dicha señal al *slot* correspondiente, como se presenta en la Figura 2.10.

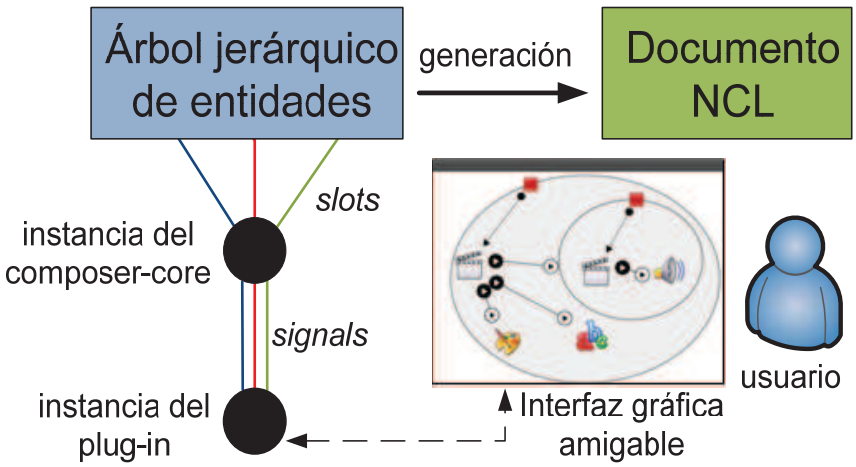


Figura 2.10 Comunicación entre una instancia de un *plug-in* con una instancia del composer-core a través del mecanismo *signals and slots*

En este esquema, un *slot* se refiere a un método que realiza una funcionalidad y que será invocado cuando la instancia del composer-core recibe la señal, de tal manera que los *plug-ins* gestionan el árbol jerárquico de entidades sin manipular directamente dichas entidades, sino que lo hacen de forma indirecta a través de la emisión de señales dirigidas al composer-core, el cual realiza una funcionalidad

(*slot*) determinada. Como sugiere la Figura 2.10, el usuario haciendo uso de la interfaz gráfica del *plug-in* indica las acciones que desea hacer sobre el árbol jerárquico de entidades, de tal manera que la instancia del *plug-in* emita las señales correspondientes a la instancia del composer-core y sea este el que manipule el árbol para generar el documento NCL.

La comunicación entre las distintas instancias de los *plug-ins* que están siendo empleados en un proyecto Composer con la instancia del composer-core es a manera de una estrella, en la que el punto central es la instancia del composer-core, como se presenta en la Figura 2.11.

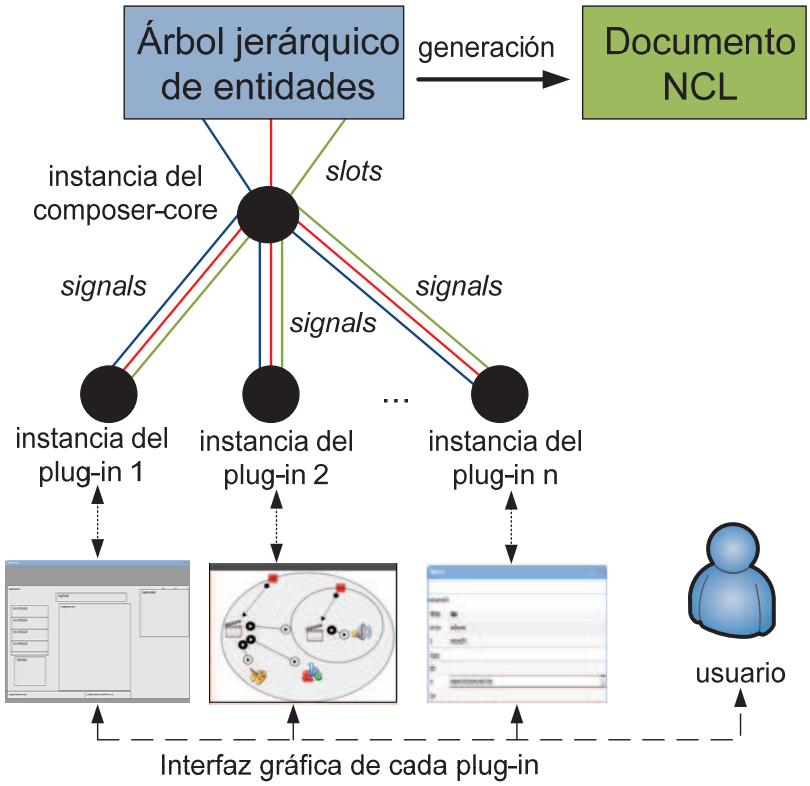


Figura 2.11 Comunicación entre instancias de varios *plug-ins* con una instancia del composer-core

2.3.1.2 Mensajes de control

El mecanismo *signals and slots* se emplea únicamente como un mecanismo de comunicación unidireccional desde los *plug-ins* hacia el composer-core. Sin embargo, es necesaria una comunicación bidireccional que surge de la necesidad de que la instancia del composer-core informe mediante mensajes de control de

las acciones realizadas sobre el árbol jerárquico de entidades a las instancias de los *plug-ins*.

En las versiones iniciales de NCL Composer se empleaba el mecanismo *signals and slots* como una comunicación bidireccional, en la que se permitía que el composer-core emita señales hacia los *plug-ins* para que ejecuten los *slots* correspondientes, y de esta manera se conseguía informar fácilmente a los *plug-ins* las acciones realizadas por el composer-core sobre el árbol jerárquico de entidades. Sin embargo, esto obligaba a implementar las conexiones (`connect`) necesarias para poder invocar a los *slots* de los *plug-ins*, lo cual implicaba una mayor complejidad en la programación para gestionar dichas conexiones[32].

En versiones posteriores de NCL Composer se solucionó este problema haciendo que únicamente los *plug-ins* puedan emitir señales (mecanismo *signals and slots* unidireccional), y se implementó en la clase `MessageControl` la funcionalidad que posibilita el envío de mensajes de control desde la instancia del composer-core a las instancias de los *plug-ins*, para invocar directamente a los *slots*, evitando la necesidad del mecanismo *signals and slots* bidireccional. La Figura 2.12 ilustra este concepto.

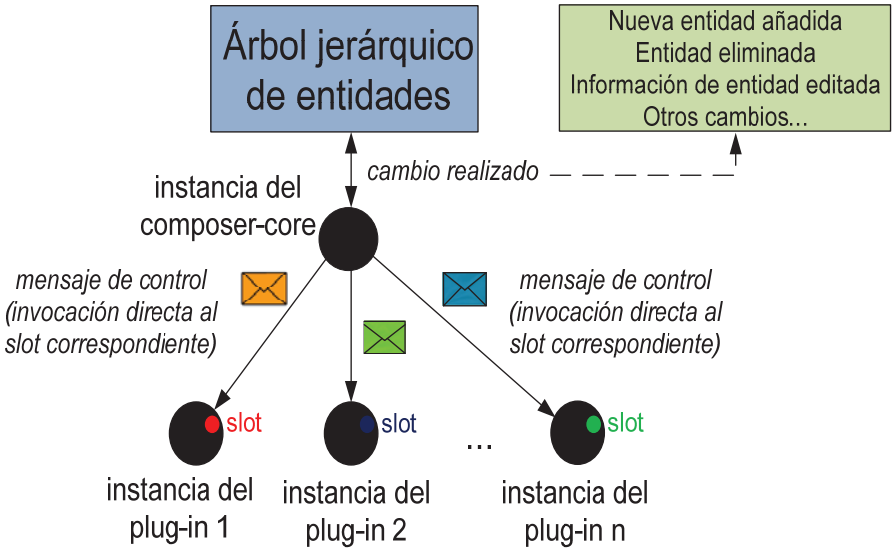


Figura 2.12 Comunicación entre una instancia del composer-core e instancias de varios *plug-ins* con la clase `MessageControl`

[32] Dado que ninguna instancia es destruida durante la comunicación, las conexiones además de ser iniciadas requieren ser destruidas, lo cual implica una mayor complejida en el manejo de las conexiones.

2.3.1.3 Mensajes de broadcast

Existe un tercer mecanismo de comunicación con el composer-core, el cual se ilustra en la Figura 2.13. Este mecanismo consiste en que una instancia de un *plug-in* emita una señal `sendBroadcastMessage` al composer-core, para que este ejecute un *slot* que, haciendo uso de la clase `PluginControl`, genere un mensaje de *broadcast* que se envíe a todos los *plug-ins* (incluyendo al *plug-in* que envió la señal al composer-core inicialmente). Cuando el composer-core recibe la señal de enviar el mensaje de *broadcast* (mediante el mecanismo *signals and slots*), invoca directamente a un *slot* de cada *plug-in* mediante un mensaje de control[33].

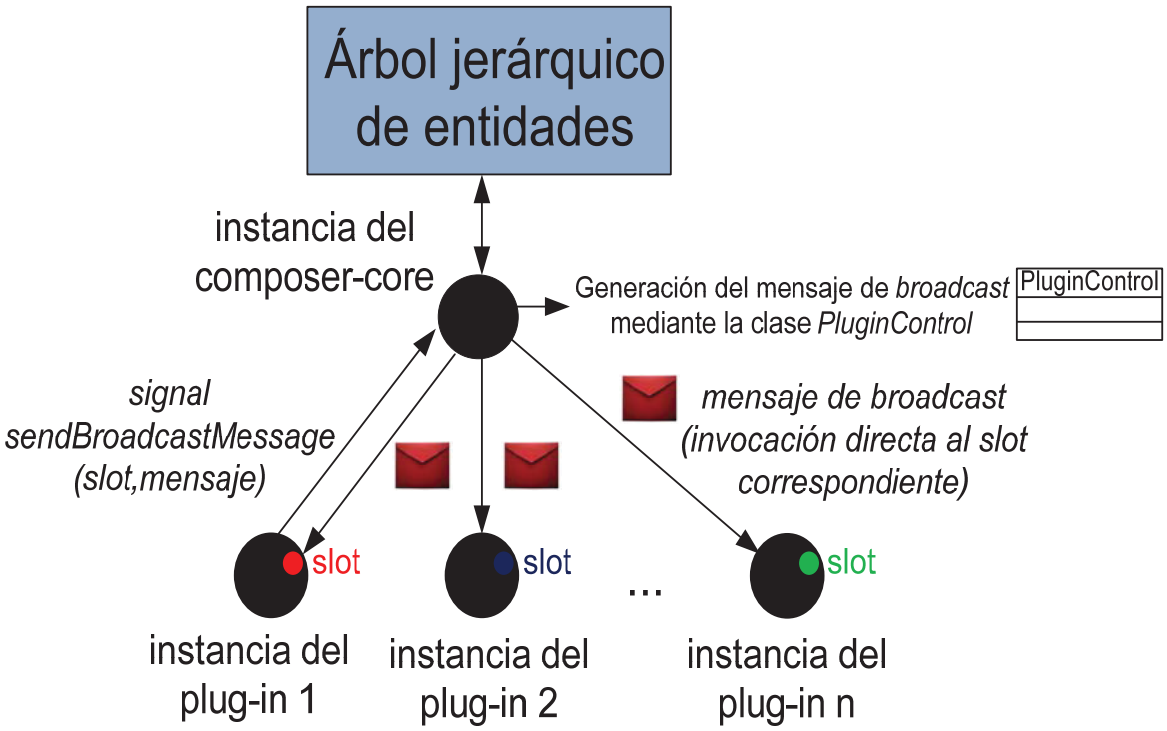


Figura 2.13 Comunicación entre una instancia del composer-core e instancias de varios *plug-ins* con la clase `PluginControl`

Este mecanismo puede ser usado para extender los mensajes del composer-core desde el código fuente de un *plug-in*, permitiendo la comunicación entre *plug-ins* usando al composer-core como un intermediario que retransmite mensajes.

[33] En este sentido un mensaje de *broadcast* puede entenderse como un conjunto de mensajes de control, los cuales son todos iguales. Cada *plug-in* recibirá uno de estos mensajes de control.

2.3.2 ESTRUCTURA DEL COMPOSER-CORE

El composer-core, por motivos de organización, ha sido dividido en cuatro secciones que componen su estructura. La Tabla 2.1 muestra las cuatro secciones que lo conforman.

Tabla 2.1 Secciones que conforman el composer-core

Sección	Función
Extensions	Contiene las clases necesarias para el funcionamiento global de NCL Composer y las interfaces para la vinculación de los *plug-ins* al IDE.
Model	Contiene las clases necesarias para la estructuración del código NCL (entidades) y gestión del documento del proyecto Composer (archivo .cpr).
Modules	Contiene la funcionalidad del composer-core que permite crear un nexo entre los *plug-ins* y el modelo.
Util	Contiene utilidades de uso general.

2.3.2.1 Sección Extensions [19], [20], [21], [22]

Esta sección comprende cuatro interfaces empleadas en la creación de un nuevo *plug-in*. La finalidad de un *plug-in* es extender la funcionalidad de NCL Composer de manera que permita al usuario realizar acciones que el IDE originalmente no puede realizar.

Estas interfaces son:

- **`IPluginFactory:`** Contiene información sobre el *plug-in* y permite crear y eliminar instancias del mismo [21].
- **`IPlugin:`** Contiene la funcionalidad del *plug-in* [22].
- **`ILanguageProfile:`** Esta interfaz debe ser implementada por un *plug-in* cuyo fin sea permitir que NCL Composer trabaje con otros lenguajes declarativos usados en televisión digital como SMIL o XHTML [20].
- **`IDocumentParser:`** Un *plug-in* implementa esta interfaz para crear un analizador de documentos (*parser*) [19]. El objetivo del analizador es permitir leer e interpretar un archivo con el código de la aplicación

interactiva en un lenguaje declarativo (como NCL, SMIL o XHTML) y estructurar a partir de este archivo el árbol jerárquico de entidades [19].

Las interfaces `IPlugin` e `IPluginFactory` se utilizan para crear *plug-ins* cuya finalidad principal es la de gestionar el árbol jerárquico de entidades para obtener así el documento NCL a ser ejecutado en el STB.

Por otro lado, las interfaces `ILanguageProfile` e `IDocumentParser` sirven para crear *plug-ins* que contienen analizadores de lenguajes, de manera que permitan a NCL Composer trabajar con otros lenguajes declarativos y no tan solo con NCL. Los *plug-ins* que se crean para este propósito no se muestran al usuario (no poseen interfaz gráfica).

El composer-core posee un *plug-in* que implementa las interfaces `ILanguageProfile` e `IDocumentParser` para trabajar con el lenguaje NCL por defecto.

2.3.2.2 Sección Model

Esta sección modela el entorno lógico con el que trabaja NCL Composer. Se compone de las siguientes clases:

- **`Entity:`** Representa una entidad NCL.
- **`Project:`** Representa un proyecto Composer que el usuario crea.
- **`Excepciones:`** Comprende un conjunto de clases para la gestión de errores que se producen en tiempo de ejecución.

2.3.2.2.1 Clase Entity [17]

La clase `Entity` es la principal herramienta empleada por el composer-core para construir el código XML de la aplicación interactiva [17].

Cada una de las etiquetas XML que estructura el composer-core representa una entidad del modelo NCM. Las entidades se anidan de manera jerárquica, de manera que una entidad puede contener entidades hijas y a la vez ser hija de otra entidad padre.

La Figura 2.14 muestra el diagrama de la clase `Entity`, en el que se han considerado únicamente los atributos y métodos de mayor importancia.

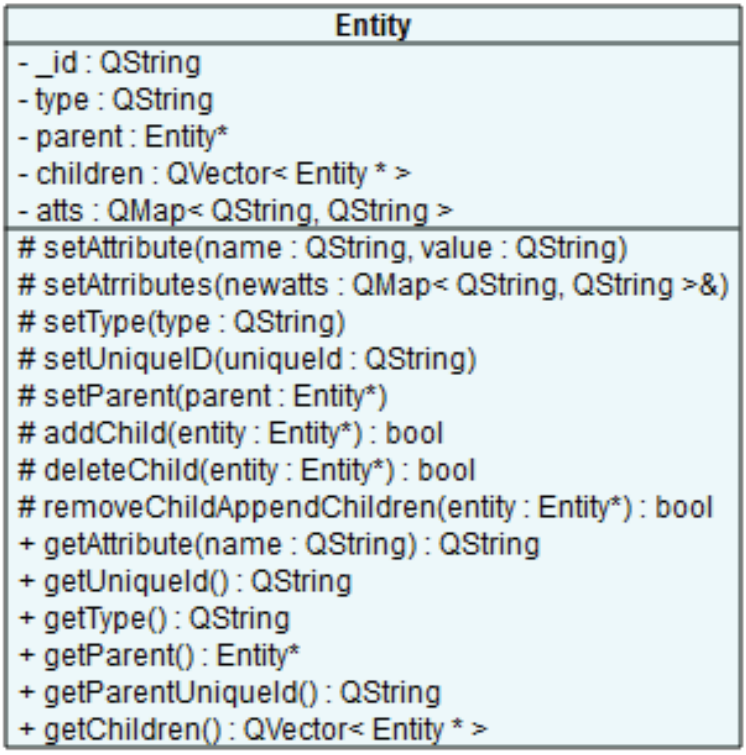


Figura 2.14 Clase `Entity`

La Tabla 2.2 describe los principales atributos de la clase `Entity`. La clase posee un identificador único (atributo `_id`) de tipo `QString` que la identifica unívocamente permitiendo diferenciarla de otras entidades[34]. El atributo `type` de tipo `QString` permite determinar el tipo de entidad[35]. Las propiedades de la etiqueta XML se expresan como un mapa (`QMap`) de pares de tipo `QString` (atributo `atts`), en el que se especifican las parejas `<propiedad, valor>` (`<QString, QString>`) que describen a la entidad. La clase `Entity` posee además una referencia a su entidad padre (atributo `parent`) y un vector de entidades (`QVector`) en el que se almacenan las referencias a las entidades hijas (atributo `children`).

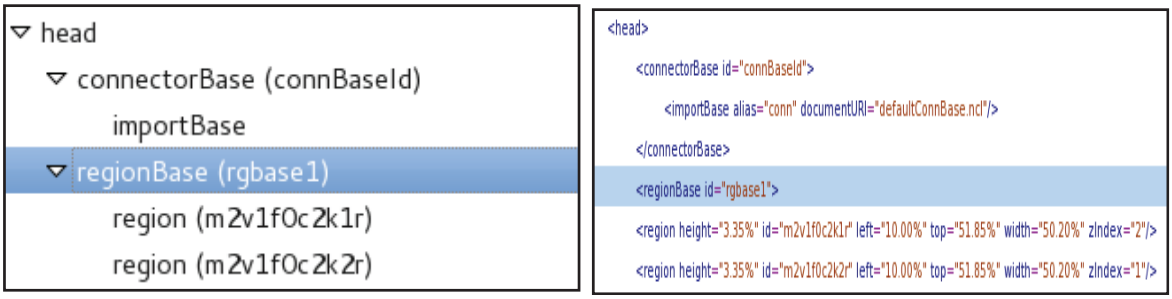
[34] Este identificador no tiene el mismo valor que el de la propiedad `id` de la etiqueta XML que define la entidad.

[35] Así por ejemplo, una región tendrá como valor de este atributo la palabra `region`, un descriptor la palabra `descriptor`, un nodo de contenido la palabra `media` y un nodo de contexto la palabra `context`.

Tabla 2.2 Principales atributos de la clase `Entity`

Atributo	Tipo	Descripción
_id	`QString`	Identifica una entidad de manera única.
type	`QString`	Determina el tipo de entidad de la que se trata.
atts	`QMap<QString, QString>`	Establece pares `<propiedad,valor>` con la información de las propiedades de la etiqueta XML que define a la entidad.
parent	`Entity*`	Referencia a la entidad padre.
children	`QVector<Entity*>`	Vector de entidades hijas.

En la Figura 2.15 se presenta el *plug-in* NCL Outline View mostrando el árbol jerárquico de entidades de una aplicación interactiva. Para la entidad de tipo `regionBase` su entidad padre es la entidad `head` y sus entidades hijas son las entidades de tipo `region`. En el código NCL generado automáticamente por el composer-core se puede observar la creación de las entidades con la anidación correcta.



(a) (b)

Figura 2.15 (a) *plug-in* NCL Outline View mostrando el árbol jerárquico de entidades (b) Código NCL generado

Los métodos de la clase `Entity` permiten obtener información sobre una entidad determinada, así como establecer sus propiedades [17]. De igual manera, desde esta clase se puede obtener la referencia a la entidad padre de la entidad y un conjunto de referencias a las entidades hijas.

Los métodos para establecer el valor de los atributos (*setters*) tienen una visibilidad de tipo `protected`. Esto obliga a los *plug-ins*, al no heredar directamente de esta clase, a emitir señales (*signals*) al composer-core para

poder gestionar entidades. Por el contrario, los métodos para obtener el valor de los atributos (*getters*) son de visibilidad `public`, por lo que los *plug-ins* pueden hacer uso de estos métodos mediante invocación directa.

2.3.2.2.2 Clase Project [1], [22], [29]

La clase `Project` representa el proyecto Composer en el que trabaja el usuario. A la vez, sirve como un repositorio que contiene toda la información referente a las entidades y *plug-ins* en un mismo recurso [29]. Esto permite al programador obtener información referente al proyecto y a las entidades empleadas para generar el documento NCL. La clase `Project` no permite establecer estado (alterar los valores de sus atributos), porque es una clase de solo lectura que ofrece información.

La Figura 2.16 muestra el diagrama de la clase `Project`, en el que se han considerado únicamente los atributos y métodos de mayor importancia.

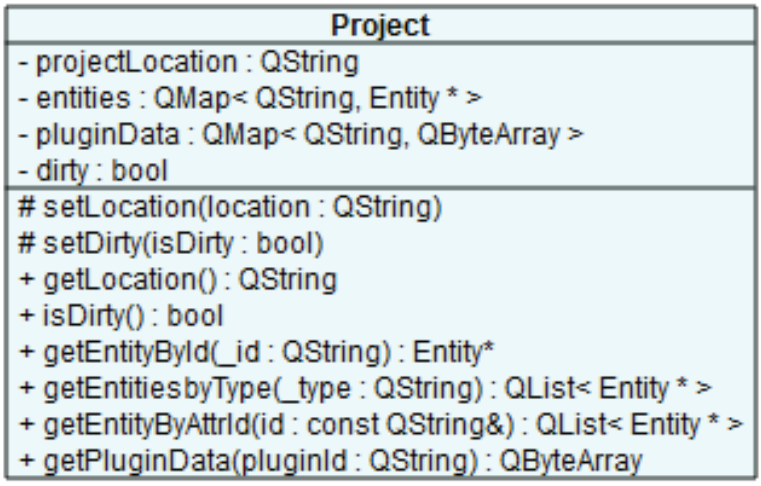


Figura 2.16 Clase `Project`

La Tabla 2.3 describe los principales atributos de la clase `Project`. La clase tiene la ubicación del proyecto actual (atributo `projectLocation` de tipo `QString`), un mapa (`QMap`) de entidades (atributo `entities`), en el que se especifican los pares `<id, entidad>` (`<QString, Entity*>`), un mapa de datos del *plug-in* (atributo `pluginData`), en el que se especifica información referente al *plug-in*

mediante los pares <id_plug-in,bytes_plug-in> (<QString, QByteArray>). La información de estos atributos se guardan en el archivo .cpr.

Tabla 2.3 Principales atributos de la clase `Project`

Atributo	Tipo	Descripción
projectLocation	QString	Ubicación del proyecto actual.
entities	QMap<QString, Entity*>	Establece pares <id,entidad> para describir a las entidades usadas en el proyecto.
pluginData	QMap<QString, QByteArray>	Establece pares <id_plug-in,bytes_plug-in> con la información de los *plug-ins* usados en el proyecto.
dirty	bool	Determina si el proyecto ha sufrido cambios o no.

Un último atributo de la clase `Project` llamado `dirty` de tipo `bool` almacena un valor (`true` o `false`) dependiendo de si el proyecto ha sufrido cambios o no. El valor de `true` indica que el proyecto ha sufrido modificaciones y que se debería guardar los cambios realizados para no perder el trabajo del usuario. El valor de `false` indica que el proyecto no ha sufrido cambios.

Los métodos de la clase `Project` permiten obtener información como la localización del proyecto actual, buscar entidades por su `id` definido como atributo de la clase `Entity`, por su propiedad `id` de la etiqueta XML o por el tipo de entidad de que se trate; así como obtener los datos guardados por el *plug-in* en el archivo .cpr [29].

Los métodos para establecer el valor de los atributos (*setters*) tienen una visibilidad de tipo `protected`, mientras que los métodos para obtener el valor de los atributos (*getters*) son de visibilidad `public`, al igual que la clase `Entity`. Esto permite al programador únicamente tener acceso directo a los métodos *getters* y obliga a emitir señales (*signals*) para establecer el valor de los atributos. Esto se debe a que cuando se usa el mecanismo *signals and slots* para establecer el valor de un atributo se tienen ventajas sobre el uso de la invocación directa a los métodos *setters*, como por ejemplo se evita la dependencia en el número de parámetros de los métodos, y además cualquier clase que implemente

un *slot* y `connect` adecuados puede recibir la señal emitida, lo que da libertad al programador para realizar cambios.

Para poder realizar las llamadas a los métodos de la clase `Project`, cada uno de los *plug-ins* necesita una referencia a dicha clase. Por ello, el archivo de cabecera `IPlugin` contenido dentro del composer-core, provee un método que se hereda al implementar la funcionalidad del *plug-in* [22]. Este método se llama `getProject()` y es declarado como tipo `inline` [1].

2.3.2.3 Sección Module [23], [24], [25], [27], [28], [30], [31]

La sección *module* está compuesta de cinco clases que permiten implementar la funcionalidad del composer-core, creando de esta manera un nexo con los *plug-ins*.

Las clases que componen esta sección son las siguientes:

- **`LanguageControl:`** Administra los analizadores de lenguajes declarativos instalados (por ejemplo NCL, SMIL o XHTML) [23]. Por lo tanto, esta clase trabaja con las interfaces `ILanguageProfile` e `IDocumentParser` que se han creado para dar soporte a otros lenguajes además del NCL.

- **`MessageControl:`** Modela un controlador de comunicaciones que administra los mensajes de control que se envían desde el composer-core hacia los *plug-ins* y viceversa [24] e implementa los *slots* correspondientes a las señales (*signals*) que emiten las instancias de los *plug-ins* para gestionar el árbol jerárquico de entidades y el proyecto Composer actual [25].

  Por cada proyecto Composer, el composer-core crea una instancia de la clase `MessageControl`.

- **`PluginControl:`** Administra los *plug-ins* de NCL Composer. Algunas de las funciones que cumple son: reconocer los *plug-ins* instalados en el

sistema, cargar los *plug-ins* que han sido reconocidos, conectar cada *plug-in* con su respectivo controlador de mensajes (instancia de la clase `MessageControl`) y llamar al método `saveSubsession()` de cada *plug-in* para archivar su información cuando el usuario guarda el proyecto Composer [27].

Cuando se abre el IDE, NCL Composer crea una instancia de la clase `PluginControl` para controlar todas las instancias de los *plug-ins* de los proyectos Composer. Esta instancia contiene, como una colección, referencias a las instancias de la clase `MessageControl` [28].

Desde la clase `PluginControl` se declaran las conexiones (`connect`) que son el punto de recepción de las señales (*signals*) que emiten las instancias de los *plug-ins*, e invoca al *slot* de la instancia que corresponda de la clase `MessageControl`.

La Figura 2.17 muestra cómo una instancia de un *plug-in* obtiene información del estado del proyecto Composer mediante la llamada directa a los métodos *getters* de la clase `Project` y emplea el mecanismo *signals and slots* para comunicarse con el controlador de comunicaciones (implementado en la clase `MessageControl`) y así gestionar el árbol jerárquico de entidades. Las sentencias `connect` son definidas en la clase `PluginControl`.

- **`ProjectControl:`** Administra los proyectos Composer que se encuentran abiertos [30]. Como en el IDE NCL Composer es posible abrir varios proyectos a la vez, se crean varias instancias de la clase `Project` y de los *plug-ins* en tiempo de ejecución (una instancia de cada *plug-in* por proyecto). La clase `ProjectControl` administra todos los proyectos implementando un sistema de control total del IDE.

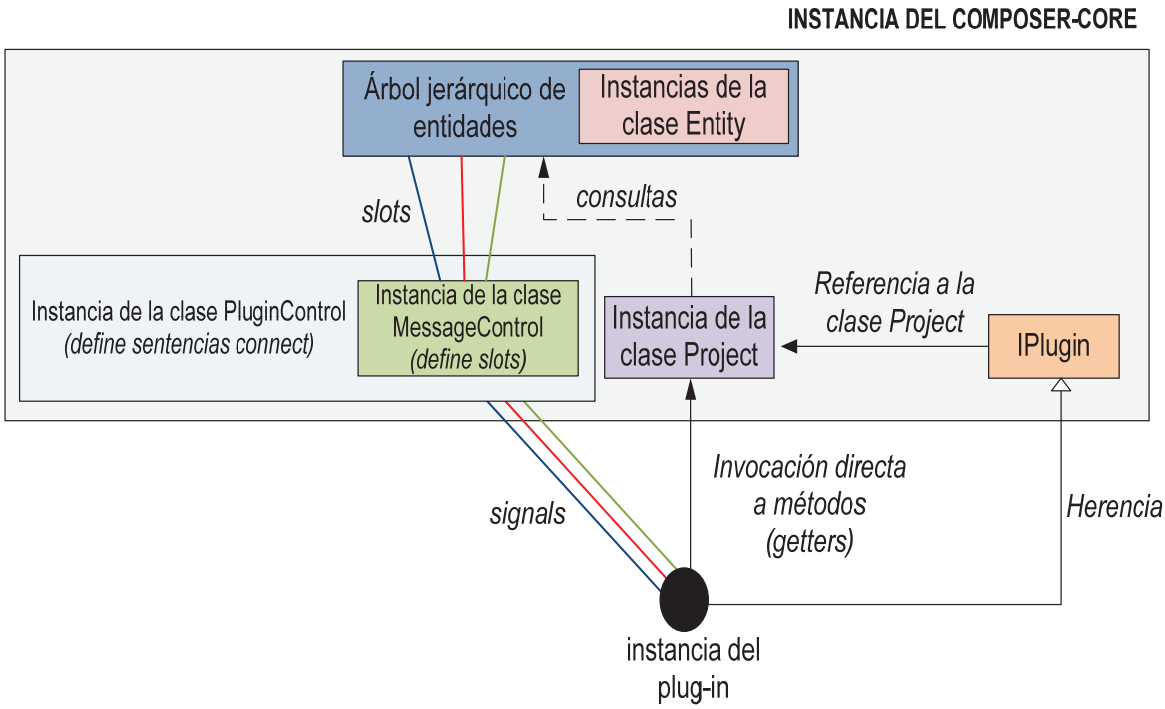


Figura 2.17 Manejo de las clases `Entity`, `Project`, `MessageControl` y `PluginControl` del composer-core

- **`ProjectReader:`** Esta clase es responsable de analizar el proyecto Composer para volver a crear el objeto de la clase `Project` que se generó cuando el usuario creó el proyecto [31]. Cuando se deba abrir un proyecto previamente guardado, esta clase lee la información del archivo .cpr y genera nuevamente el objeto de la clase `Project` que contiene la información de los datos del proyecto, de los *plug-ins* empleados y de las entidades utilizadas para la generación del documento NCL.

2.3.2.4 Sección Util [11], [13], [15], [16]

La sección *util* comprende cuatro clases que poseen métodos de interés general que sirven como herramientas de reuso de código para los programadores.

Las clases que componen esta sección son las siguientes:

- **`AttributeReferences:`** Guarda información sobre el par (elemento, atributo) que puede referenciar un valor de otro par (elemento, atributo) [13]. Esta clase es principalmente usada cuando se está definiendo la

sintaxis del lenguaje (NCL, SMIL o XHTML). Solo se requiere si se va a realizar un nuevo analizador de lenguaje.

- **`Commands:`** Contiene comandos de uso interno para añadir, editar y eliminar entidades, así como para permitir acciones de *undo* y *redo* (deshacer y rehacer). Estos comandos permiten a las instancias de la clase `MessageControl` realizar un histórico de la gestión del árbol jerárquico de entidades.
- **`ComposerSettings:`** Provee configuraciones adicionales que son necesarias para garantizar que NCL Composer sea multiplataforma [15]. Para ello, hace uso mediante herencia de los métodos de la clase `QSettings` de la biblioteca Qt, lo que permite una configuración del IDE de manera persistente e independiente de la plataforma [11].
- **`Utilities:`** Implementa un conjunto de métodos de gran utilidad tanto para programar el composer-core así como para programar un *plug-in* [16]. Los métodos que implementa esta clase permiten, entre otras cosas: obtener el lenguaje de programación correspondiente a una extensión (NCL, SMIL, XHTML) y viceversa, obtener la ruta relativa al proyecto actual dada la ruta absoluta (lo cual es útil para definir las propiedades `src` de las entidades NCL), gestionar rutas de diálogos de archivos, entre otras funcionalidades.

2.4 CREACIÓN DE UN NUEVO PLUG-IN PARA NCL COMPOSER

Para crear un *plug-in* en NCL Composer se debe hacer uso de dos interfaces, cuyos métodos deben ser implementados en dos clases distintas[36]. Estas interfaces son `IPluginFactory` e `IPlugin`.

2.4.1 INTERFAZ IPLUGINFACTORY [21]

La interfaz `IPluginFactory` define métodos que permiten especificar información sobre el *plug-in*, así como crear y eliminar instancias del mismo [21].

[36] Una clase que herede de estas interfaces no está obligada a implementar todos los métodos, sino únicamente los que se requiera.

Sin esta interfaz no sería posible registrar el *plug-in* para que sea utilizado por el IDE como el resto de *plug-ins* originales.

La información sobre el *plug-in* que se especifica con esta interfaz se presenta en el diálogo *Help/About Plugins/Details* de NCL Composer. Esta información comprende datos generales del *plug-in* que se presentan en la Tabla 2.4. De estos datos, únicamente el identificador del *plug-in* y la lista de tipos de lenguajes son de uso interno, por lo que no se muestran al usuario.

Tabla 2.4 Datos de un *plug-in* que permite especificar `IPluginFactory`

Dato	Descripción
ID (identificador)	Cadena de texto que identifica unívocamente al *plug-in*.
Nombre registrado	Nombre con el cual el *plug-in* fue registrado en NCL Composer.
Versión	Versión del *plug-in*.
Versión compatible	Versión de NCL Composer a partir de la cual es compatible con el *plug-in*.
Propietario	Nombre o razón social del propietario.
Copyright	Derechos de autor.
Licencia	Tipo de licencia del *plug-in*.
Descripción	Texto que describe la funcionalidad del *plug-in*.
URL (*Uniform Resource Locator*)	Dirección donde el usuario puede obtener más información del *plug-in*.
Categoría	Indica la funcionalidad principal para la que fue creado el *plug-in*. Generalmente este valor es "NCL", debido a que los *plug-ins* son generadores y gestores de código NCL por defecto.
Tipos de lenguajes	Listado de los lenguajes que el *plug-in* puede emplear (ej: NCL, SMIL, XHTML). Este dato es de tipo `QList`.

La interfaz también permite crear nuevas instancias del *plug-in* y eliminarlas [21]. Para ello, la interfaz especifica dos métodos, uno para crear la instancia del *plug-in* y otro para eliminarla. Dichos métodos se detallan en la Tabla 2.5.

La instancia de un *plug-in* se declara como un objeto de tipo `IPlugin`, que es la interfaz de la que una clase hereda para implementar la funcionalidad del *plug-in*.

Tabla 2.5 Métodos para crear y eliminar instancias de un *plug-in*

Método	Funcionalidad
`IPlugin* createPluginInstance()`	Este método retorna una instancia del *plug-in*.
`void realeasePluginInstance (IPlugin* plugin)`	Este método elimina de la memoria principal la instancia del *plug-in* que se recibe como argumento.

2.4.2 INTERFAZ IPLUGIN [22]

La interfaz `IPlugin` especifica métodos que implementan la funcionalidad del *plug-in* [22]. Estos métodos también permiten la comunicación con el composer-core mediante el uso del mecanismo *signals and slots* y la recepción de mensajes de control.

Esta interfaz se enlaza con una instancia del proyecto, por lo que sus métodos serán llamados cada vez que el proyecto sufra un cambio [22]. De igual manera, cuando el *plug-in* requiera realizar cambios en el proyecto se realizará la invocación de estos métodos.

Las señales que emite la instancia del *plug-in* son enviadas por la clase que implementa los métodos de `IPlugin` a la instancia del composer-core empleando el mecanismo *signals and slots*. Una vez que la instancia del composer-core recibe la señal, esta procesa el requerimiento, ejecuta el *slot* correspondiente y envía un mensaje para comunicar la acción realizada a los demás *plug-ins* asociados a NCL Composer. Además de los métodos de la interfaz, el programador del *plug-in* debe implementar en la clase que hereda de `IPlugin` los *slots* correspondientes para poder capturar los mensajes de control enviados por la instancia del composer-core.

La Tabla 2.6 especifica los métodos de la interfaz `IPlugin` y su funcionalidad.

Al crear un nuevo proyecto de NCL Composer, se crea una instancia de cada uno de los *plug-ins* y se llama a su método `init()`, para cargar los datos en caso de que sea un proyecto ya existente, y al método `getWidget()` para obtener su interfaz gráfica. Para guardar los cambios de un proyecto, NCL Composer llama al

método `saveSubsession()` de cada uno de los *plug-ins* y guarda en el archivo .cpr los datos que cada *plug-in* requiera guardar, diferenciándolos por el identificador (`ID`) especificado en la clase que implementa los métodos de la interfaz `IPluginFactory`.

Tabla 2.6 Métodos de la interfaz `IPlugin`

Método	Funcionalidad
`void init()`	Se ejecuta tras instanciar el *plug-in* y permite cargar los datos guardados en el archivo .cpr del proyecto.
`QWidget* getWidget()`	Obtiene el objeto que corresponde a la interfaz gráfica de usuario del *plug-in*.
`bool saveSubsession()`	Método que se ejecuta cuando se guarda el proyecto o sus cambios.
`inline Project* getProject()`	Obtiene la instancia de la clase `Project` correspondiente al proyecto actual.
`inline void setProject(Project* project)`	Asigna la instancia de la clase `Project` correspondiente al proyecto Composer, a la instancia del *plug-in*.
`inline QString getPluginInstanceID()`	A cada *plug-in* se le asigna un identificador único cada vez que se crea una instancia. Este método retorna el `id` de la instancia del *plug-in*.
`inline void setPluginInstanceID(QString pluginInstID)`	Establece el valor del identificador (`id`) de la instancia del *plug-in*.
`inline ILanguageProfile* getLanguageProfile()`	Obtiene el analizador de lenguaje utlizado por el *plug-in*.
`inline void setLanguageProfile(ILanguageProfile* languageProfile)`	Establece el analizador de lenguaje que debe usar el *plug-in*.

En la Tabla 2.7 se presentan las señales que un *plug-in* puede emitir y la funcionalidad del *slot* implementado en el composer-core para dicha señal.

Tabla 2.7 Señales a las que responde el composer-core

Signal	Parámetros	Funcionalidad del slot
`void addEntity (QString type, QString parentEntityId, QMap <QString, QString> atts, bool force)`	**`type:`** Tipo de entidad NCL (XML *tagname*). Ej. `media`, `region`, `descriptor`. **`parentEntityId:`** Identificador de la entidad padre dentro del árbol jerárquico de entidades. **`atts:`** Propiedades de la entidad. **`force:`** Tiene un valor de `true` si se debe añadir la entidad aunque se produzca un error dentro del árbol jerárquico.	Añade una nueva entidad NCL al árbol jerárquico.
`void setAttributes (Entity* entity, QMap <QString, QString> atts, bool force)`	**`entity:`** Referencia a la entidad cuyos atributos serán establecidos. **`atts:`** Propiedades de la entidad. **`force:`** Tiene un valor de `true` si se deben establecer las propiedades aunque se produzca un error dentro del árbol jerárquico.	Establece las propiedades de la entidad NCL (propiedades de la etiqueta XML).
`void removeEntity (Entity* entity, bool force)`	**`entity:`** Referencia a la entidad a ser eliminada. **`force:`** Tiene un valor de `true` si se debe eliminar la entidad aunque se produzca un error dentro del árbol jerárquico.	Elimina una entidad NCL del árbol jerárquico.
`void setListenFilter (const QStringList &entityTypes)`	**`entityTypes:`** Lista de tipo `QString` que indica el tipo de entidades a escuchar.	Funciona como un filtro que indica al composer-core que el *plug-in* está interesado únicamente en las entidades del tipo pasado como argumento.
`void sendBroadcastMessage (const char* msg, void* obj)`	**`msg:`** Nombre del *slot* del lado de los *plug-ins* al que se debe llamar. **`obj:`** Cualquier `QObject` (objeto de Qt) que se desee transmitir en *broadcast*.	Puede ser usado para extender los mensajes del composer-core. Permite la comunicación entre *plug-ins*. El composer-core retransmite el mensaje de *broadcast* sin ninguna validación.
`void setPluginData (QByteArray data)`	**`data:`** Arreglo de *bytes* con información del *plug-in*.	Guarda los datos del *plug-in* en el archivo de proyecto Composer (.cpr).
`void setCurrentProjectAsDirty()`		Establece que se han realizado cambios en el proyecto y de ser necesario habilita el botón `Guardar` en el composer-gui.

Si el composer-core requiere comunicar algo al *plug-in* realizará la invocación directamente al *slot* implementado en el *plug-in*. En la Tabla 2.8 se muestran los *slots* que deben implementarse en el *plug-in* para receptar los mensajes de control enviados por el composer-core. Estos *slots* no necesitan de la implementación de `connects` ya que son invocados directamente por el composer-core.

Tabla 2.8 *Slots* para recepción de mensajes de control del composer-core

Slot	Parámetros	Funcionalidad
`void onEntityAdded (QString pluginID, Entity* entity)`	**`pluginID:`** Identificador del *plug-in* que ha añadido la entidad. **`entity:`** Instancia de la entidad que ha sido añadida.	Permite indicar a los *plug-ins* que una entidad ha sido añadida al árbol jerárquico.
`void onEntityChanged (QString pluginID, Entity* entity)`	**`pluginID:`** Identificador del *plug-in* que ha cambiado la entidad. **`entity:`** Instancia de la entidad que ha sido cambiada.	Permite indicar a los *plug-ins* que una entidad ha sido cambiada.
`void onEntityRemoved (QString pluginID, QString entityID)`	**`pluginID:`** Identificador del *plug-in* que ha eliminado la entidad. **`entityID:`** Identificador de la entidad que ha sido eliminada.	Permite indicar a los *plug-ins* que una entidad ha sido eliminada del árbol jerárquico.
`void updateFromModel()`		Permite actualizar la información de los *plug-ins* en base al estado actual del composer-core.
`void errorMessage (QString error)`	**`error:`** Una descripción del error.	Permite indicar sobre algún error que ha ocurrido.

2.5 PLUG-IN DE GENERACIÓN AUTOMÁTICA DE MENÚS: "MENU CREATOR"

El *plug-in* lleva por nombre Menu Creator y permite al usuario crear uno o varios menús de manera sencilla generando automáticamente el código NCL respectivo.

El resultado que se obtiene al crear un menú empleando el *plug-in* es un nodo de contexto. Dicho nodo contiene todas las facilidades de interacción que un menú puede prestar al diseñador de la aplicación NCL tales como uso de estilos gráficos, menús con distintas propiedades de diseño (número de filas, número de columnas, dimensiones del menú, entre otras), división de menús en vistas (una vista es un fragmento de un menú), uso de viñetas de navegación (símbolos gráficos que indican al televidente cómo desplazarse entre las distintas vistas), creación de menús con filas y columnas de distinto tamaño, ingreso del texto a presentar, navegación entre los elementos de menús e inserción de enlaces NCL.

2.5.1 CASOS DE USO

Para el análisis de los casos de uso se considera un único actor llamado *Diseñador NCL*, el cual tiene ciertos conocimientos del lenguaje NCL y está en la capacidad de implementar aplicaciones interactivas haciendo uso del IDE NCL Composer y del *plug-in* Menu Creator.

El diseño del *plug-in* se basa en tres casos de uso generales que se presentan en la Figura 2.18.

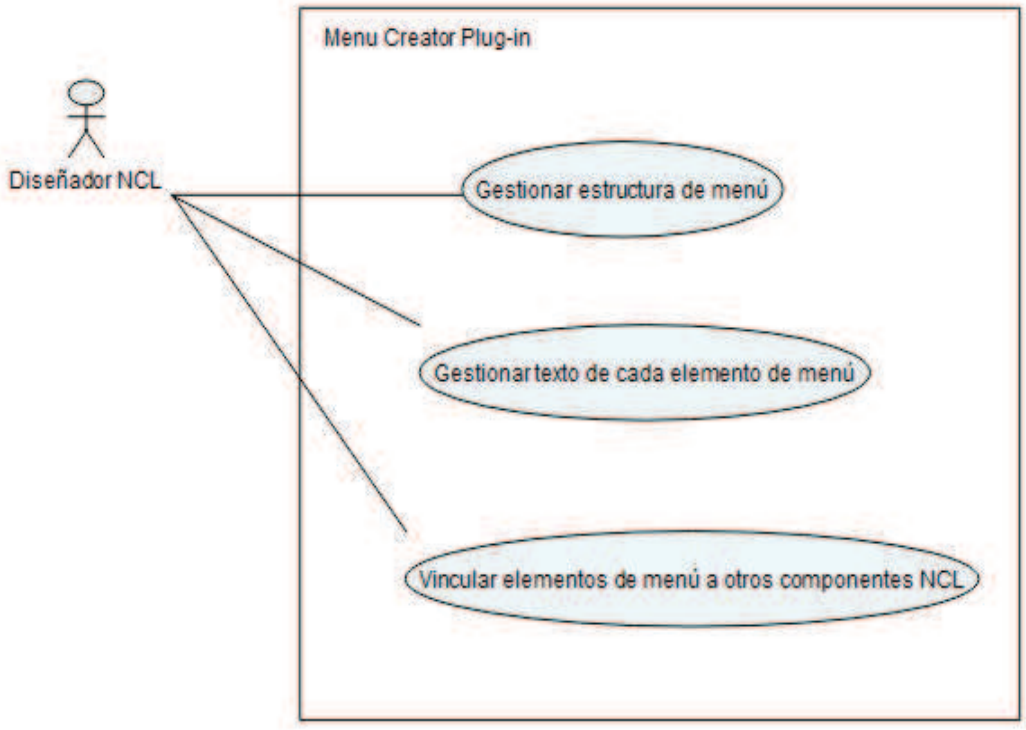


Figura 2.18 Diagrama de casos de uso general

Estos casos de uso son:

- **Gestionar estructura de menú:** Comprende acciones que permiten trabajar con menús y su código NCL.
- **Gestionar texto de cada elemento de menú:** Comprende acciones que permiten generar imágenes PNG con el texto de los elementos de un menú determinando sus propiedades de diseño.
- **Vincular elementos de menú a otros componentes NCL:** Comprende acciones que permiten insertar enlaces NCL para asociar un menú creado con otros componentes empleados en una aplicación interactiva, tales como imágenes, videos u otros menús creados con el *plug-in.*

2.5.1.1 Gestionar estructura de menú

Para la gestión de la estructura de menú se especifican los casos de uso que se presentan en la Figura 2.19.

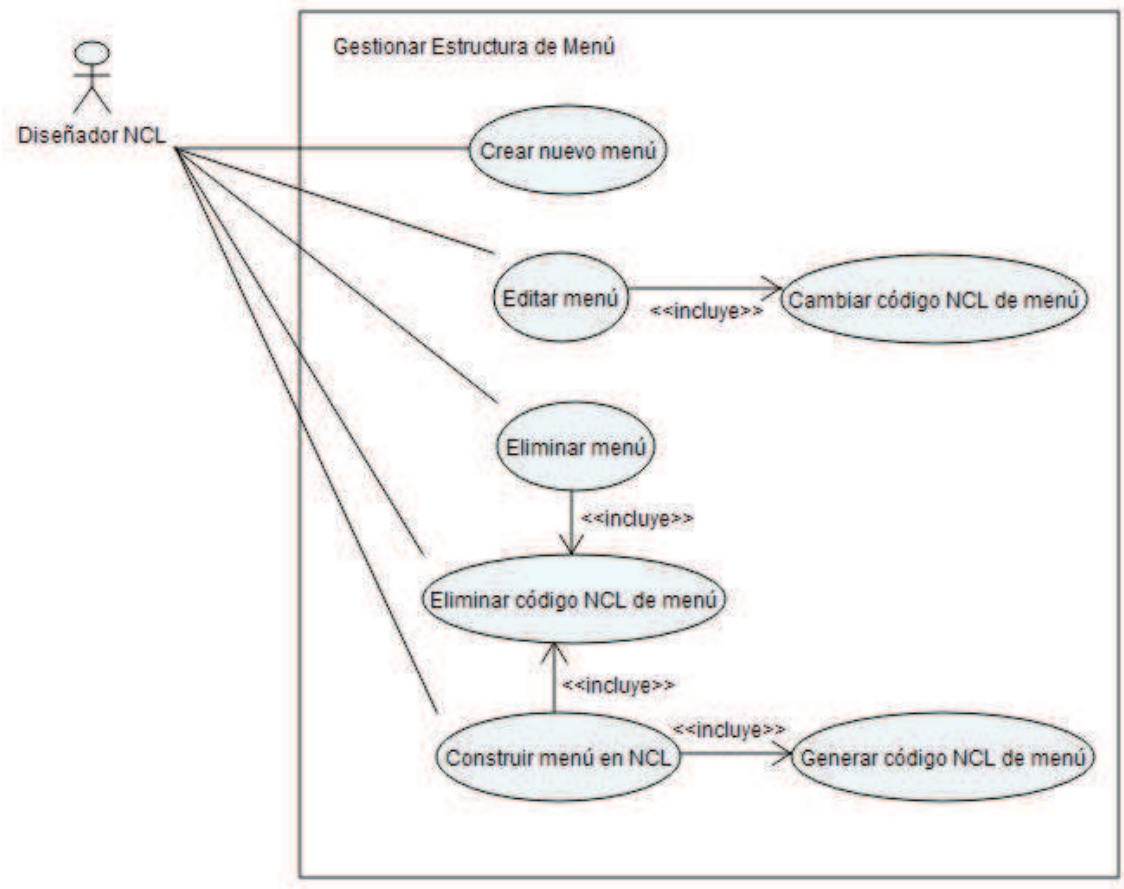


Figura 2.19 Diagrama de casos de uso para gestionar estructura de menú

Estos casos de uso de uso son:

- **Crear nuevo menú:** Permite diseñar un nuevo menú estableciendo sus parámetros de diseño, sin aún generar el código NCL.
- **Editar menú:** Permite cambiar los parámetros de diseño de un menú. Si el código NCL ya ha sido generado, los cambios se realizan automáticamente en el código de acuerdo a los nuevos parámetros establecidos.
- **Cambiar código NCL de menú:** Cambia el código NCL de un menú cuyos parámetros de diseño han sido modificados.
- **Eliminar menú:** Elimina el diseño y el código NCL generado para el menú. Si el código NCL del menú aún no ha sido generado, únicamente elimina su diseño.
- **Eliminar código NCL de menú:** Elimina el código NCL de un menú, pero mantiene los parámetros de diseño del mismo.
- **Construir menú en NCL:** Genera el código NCL de un menú diseñado de acuerdo a los parámetros establecidos por el diseñador. Si el código para dicho menú ya existía, lo elimina para volver a generar el código otra vez.
- **Generar código NCL de menú:** Genera el código NCL de un menú diseñado de acuerdo a los parámetros establecidos por el diseñador.

2.5.1.1.1 Editar menú

Para la edición de un menú se emplearon los casos de uso que se detallan en la Figura 2.20, y están orientados a establecer los parámetros de diseño de un menú.

Estos casos de uso son:

- **Establecer estilo (plantilla):** Permite seleccionar la plantilla con las imágenes PNG a partir de las cuales el *plug-in* estructura el menú diseñado.
- **Seleccionar conector de navegación entre elementos de menú:** Permite seleccionar el tipo de conector NCL causal para posibilitar la navegación entre los distintos elementos del menú.

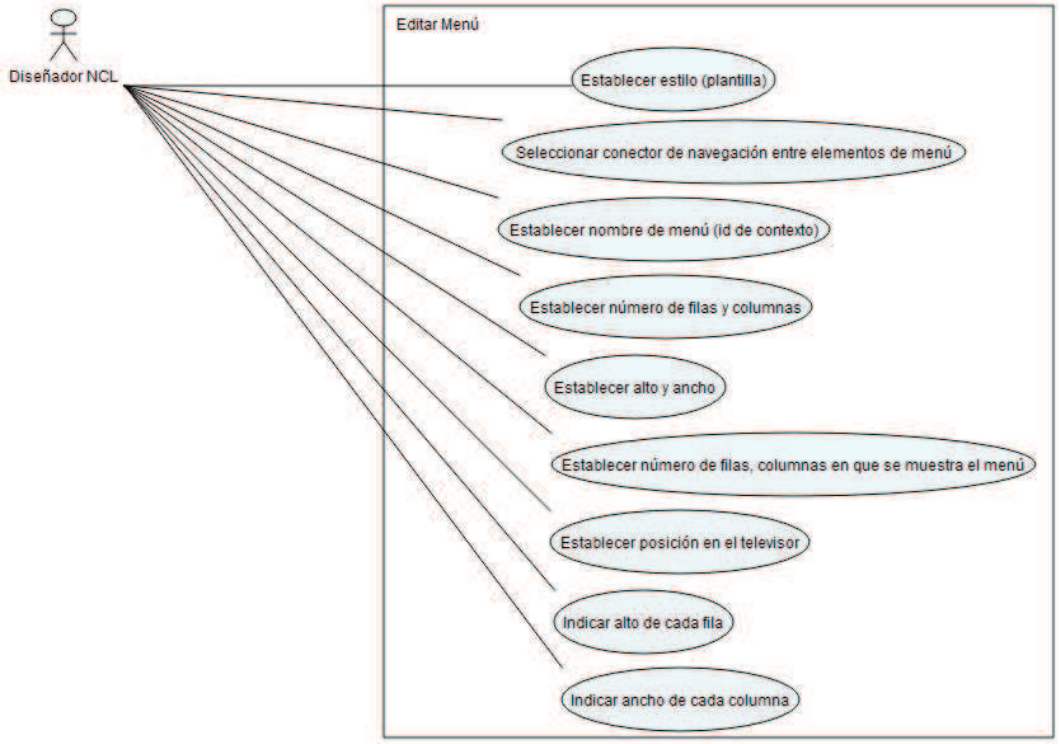


Figura 2.20 Diagrama de casos de uso para editar estructura de menú

- **Establecer nombre de menú (`id` de contexto):** Permite establecer el valor de la propiedad `id` con el que se generará el nodo de contexto del menú diseñado.
- **Establecer número de filas y columnas:** Permite establecer las dimensiones del menú, indicando el número de filas y de columnas que lo componen.
- **Establecer alto y ancho:** Permite establecer las dimensiones del alto y ancho del menú en relación a las dimensiones de la pantalla del televisor.
- **Establecer número de filas, columnas en que se muestra el menú:** Permite determinar el número de filas y de columnas que conforman una vista de un menú.
- **Establecer posición en el televisor:** Permite determinar la posición en la que se ubicará el menú diseñado en la pantalla del televisor del televidente.
- **Indicar alto de cada fila:** Permite establecer el alto de una determinada fila de un menú como un porcentaje respecto de las dimensiones de la pantalla del televisor.

- **Indicar ancho de cada columna:** Permite establecer el ancho de una determinada columna de un menú como un porcentaje respecto de las dimensiones de la pantalla del televisor.

2.5.1.2 Gestionar texto de los elementos de menús

La gestión del texto de los elementos de un menú se basa en los casos de uso que se detallan en la Figura 2.21. Aún cuando el objetivo del *plug-in* dentro del sistema de búsqueda, almacenamiento y procesamiento de información es generar menús sin texto, se ha diseñado al *plug-in* con las opciones que permiten también ingresar texto en los menús, de manera que el *plug-in* sea una herramienta independiente capaz de trabajar sin necesidad del Subsistema de Adquisición y Procesamiento de Datos (SAPDa).

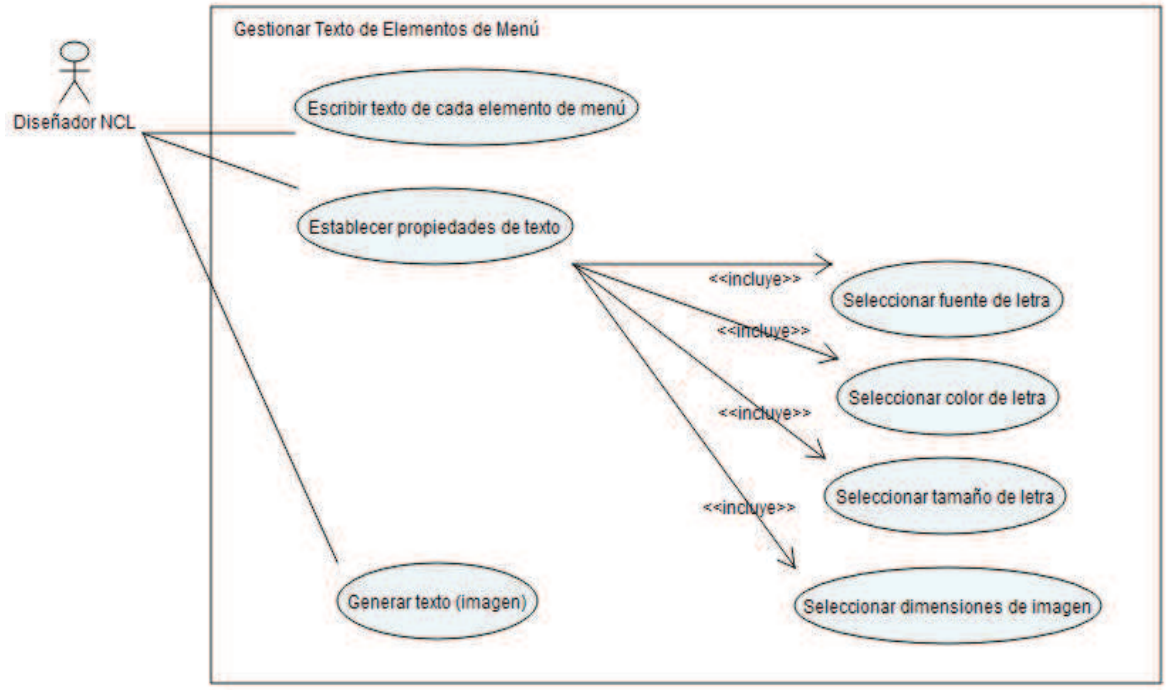


Figura 2.21 Diagrama de casos de uso para gestionar el texto de los elementos del menú

Estos casos de uso son:

- **Escribir texto de cada elemento de menú:** Permite al diseñador ingresar el texto que se mostrará en cada elemento del menú.

- **Establecer propiedades de texto:** Permite determinar las características del texto y de la imagen que se generará.
- **Seleccionar fuente de letra:** Establece el tipo de letra para el texto.
- **Seleccionar color de letra:** Establece el color de letra para el texto.
- **Seleccionar tamaño de letra:** Establece el tamaño de letra para el texto.
- **Seleccionar dimensiones de imagen:** Establece el valor del ancho y el alto en píxeles de la imagen PNG (dimensiones del lienzo), correspondientes al texto de un elemento de un menú.
- **Generar texto (imagen):** Genera la imagen PNG correspondiente al texto con las características establecidas por el diseñador.

2.5.2 ESTRUCTURA JERÁRQUICA DE CLASES [7]

Para facilitar el trabajo de diseño de los menús y la posterior generación de las entidades NCL que conforman los menús diseñados, se ha modelado una estructura jerárquica de clases que se presenta en la Figura 2.22, haciendo uso de asociaciones binarias *one way* en las que un objeto de una clase de nivel superior posee una colección de referencias a objetos de clases de nivel inferior.

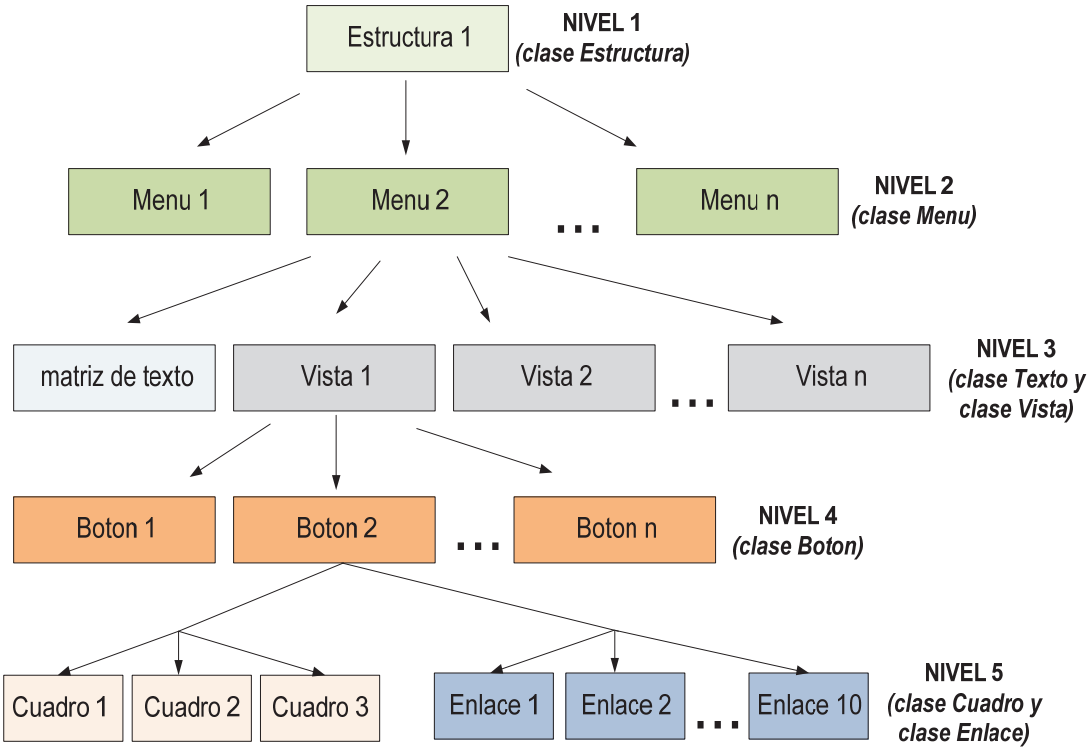


Figura 2.22 Estructura jerárquica de clases del *plug-in* Menu Creator

El primer nivel contiene la clase `Estructura`. Una instancia de esta clase tiene referencias a una o más instancias de la clase `Menu` del segundo nivel que a su vez tienen como atributos las propiedades generales de un menú que ha diseñado el usuario tales como el número de filas, número de columnas, porcentaje de ocupación del menú en la pantalla (ancho y alto), número de filas y de columnas que se muestran por cada vista y la posición que ocupa el menú en la pantalla. Además, cada instancia de tipo `Menu` tiene referencias a instancias de la clase `Texto`, correspondientes al texto de cada uno de los elementos que conforman el menú, y una o más instancias de la clase `Vista`; todos estos objetos se ubican en el nivel 3.

En el nivel 4 se encuentran instancias de la clase `Boton`, que representan a cada uno de los elementos que conforman una vista determinada de un menú. Cada una de estas instancias contiene tres instancias del nivel 5 de tipo `Cuadro` correspondientes al texto del elemento y dos imágenes de fondo, una para cuando el elemento no está seleccionado (fondo) y la otra para cuando el televidente selecciona el elemento (foco).

La clase `Cuadro` es la abstracción de tres entidades NCL [7]: una región que indica en donde se posicionará el elemento del menú y sus dimensiones; un nodo de contenido asociado con la imagen a abrir; y, un descriptor para enlazar la región con el nodo de contenido. Además, cada instancia de la clase `Boton` tiene referencias a diez instancias de la clase `Enlace` del nivel 5, correspondientes a las posibles acciones que realiza el televidente a través del control remoto: ir arriba, ir abajo, ir derecha, ir izquierda, presionar botón ROJO, VERDE, AMARILLO, AZUL, ESC y OK. Cada instancia de tipo `Enlace` representa una entidad NCL de tipo `link` [7] que permite la navegación a través del menú.

La estructura jerárquica de clases permite que todas las entidades NCL requeridas para los menús diseñados por el usuario sean añadidas al árbol jerárquico de entidades de NCL Composer, de manera que, posteriormente, el usuario pueda generar con el *plug-in* el código NCL de cada menú, como se muestra en la Figura 2.23. Para ello, la instancia del *plug-in* (creada gracias a la clase que implementa la interfaz `IPluginFactory`) emite las señales

correspondientes a una instancia del composer-core mediante el mecanismo de *signals and slots* para generar el documento NCL.

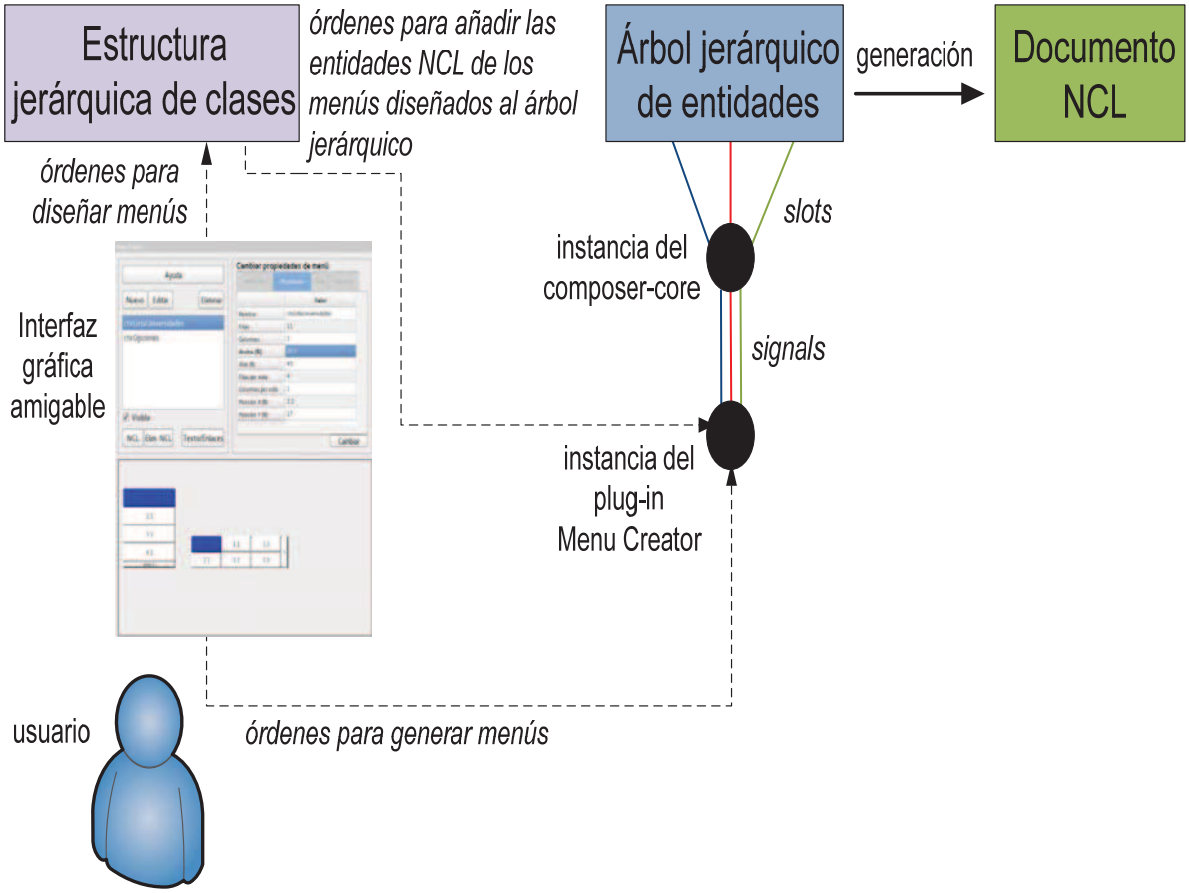


Figura 2.23 Mecanismo de trabajo de Menu Creator

La Figura 2.24 presenta el diagrama de clases del plug-in Menu Creator. Cada clase, a excepción de la clase Estructura, posee como parte de sus atributos un identificador alfanumérico único que permite identificar a las instancias de cada clase de manera única.

2.5.2.1 Clase Estructura

La clase `Estructura` se instancia una sola vez durante la ejecución del *plug-in* por lo que no requiere de un identificador alfanumérico.

Los atributos de esta clase son:

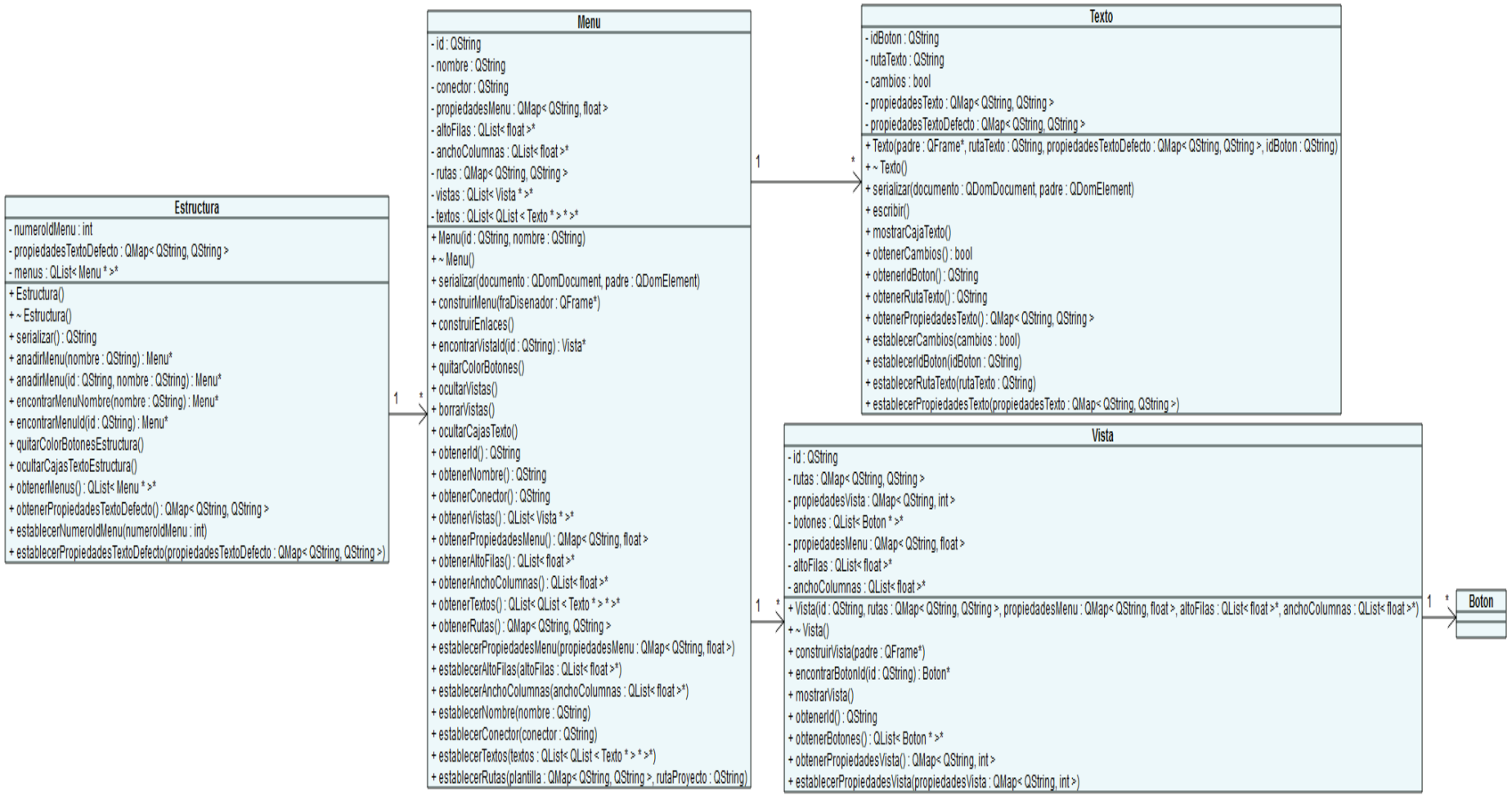


Figura 2.24 Diagrama de clases del *plug-in* Menu Creator (primera parte)

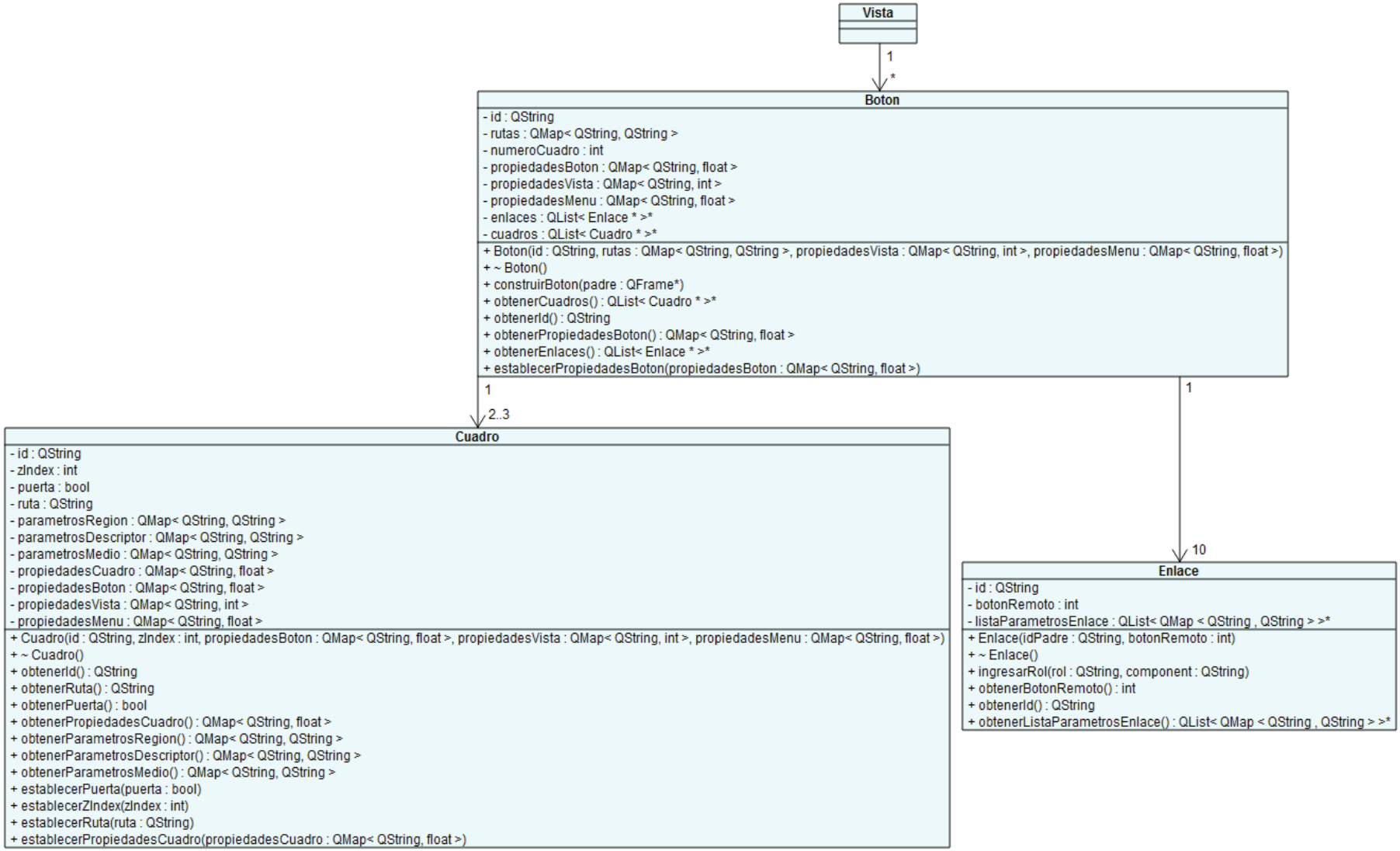


Figura 2.24 Diagrama de clases del *plug-in* Menu Creator (segunda parte)

- **numeroIdMenu:** Atributo de tipo `int`. Es un contador incremental en pasos de uno a partir del cual se construye el identificador alfanumérico de cada instancia de la clase `Menu`. Cada vez que el diseñador crea un menú, este contador se incrementa en uno.
- **propiedadesTextoDefecto:** Atributo de tipo `QMap<QString,QString>`. Almacena los valores por defecto correspondientes a la fuente, color, tamaño, ancho del lienzo y alto de lienzo para la generación de la imagen de texto. Estos valores se presentan en la Tabla 2.9.

Tabla 2.9 Valores por defecto de las propiedades para la generación de texto

Propiedad	Valor por defecto
fuente	Toma la primera fuente encontrada en el sistema operativo.
color	Verde (#00FF00)
tamanio	50
anchoLienzo	600 píxeles
altoLienzo	200 píxeles

- **menus:** Atributo de tipo `QList<Menu*>*`. Consiste en una colección con referencias a instancias de la clase `Menu`.

El objetivo principal de la clase `Estructura` es agrupar las instancias de la clase `Menu`, permitiendo gestionar todos los menús que cree el usuario de manera más sencilla y facilitando la serialización XML de todo el trabajo del usuario mediante la invocación a un solo método, para guardar la estructura jerárquica de clases en el archivo .cpr del proyecto Composer.

2.5.2.2 Clase Menu

La clase `Menu` modela un menú diseñado por el usuario. Los atributos de esta clase son:

- **id:** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único que permite distinguir las distintas instancias creadas de esta clase.

Este código inicia con la letra m seguida del valor del contador `numeroIdMenu` de la instancia de la clase `Estructura`. Así, el primer menú diseñado por el usuario tendrá el código m1, el segundo m2 y así sucesivamente.

- **`nombre:`** Atributo de tipo `QString`. Corresponde al nombre del menú que se emplea como `id` del nodo de contexto.
- **`conector:`** Atributo de tipo `QString`. Establece el `id` del conector causal empleado para permitir la navegación entre los elementos del menú.
- **`propiedadesMenu:`** Atributo de tipo `QMap<QString,float>`. Almacena, en tiempo de ejecución, los pares `<nombre_propiedad,valor>` que se presentan en la Tabla 2.10.

Tabla 2.10 Propiedades del atributo `propiedadesMenu` de la clase `Menu`

Propiedad	Descripción
`filas`	Número de filas del menú.
`columnas`	Número de columnas del menú.
`ancho`	Ancho en porcentaje (%) que ocupa el menú respecto de la pantalla del televisor.
`alto`	Alto en porcentaje (%) que ocupa el menú respecto de la pantalla del televisor.
`vFilas`	Número de filas que se muestran por vista. Este valor es siempre menor o igual al número de filas del menú.
`vColumnas`	Número de columnas que se muestran por vista. Este valor es siempre menor o igual al número de columnas del menú.
`posX`	Posición en porcentaje (%) que ocupa el menú desde el borde lateral izquierdo del televisor respecto del ancho del televisor.
`posY`	Posición en porcentaje (%) que ocupa el menú desde el borde superior del televisor respecto del alto del televisor.

- **`altoFilas:`** Atributo de tipo `QList<float>*`. Almacena, en tiempo de ejecución, el alto de cada fila que conforma el menú.
- **`anchoColumnas:`** Atributo de tipo `QList<float>*`. Almacena, en tiempo de ejecución, el ancho de cada columna que conforma el menú.

- **`rutas:`** Atributo de tipo `QMap<QString,QString>`. Almacena la ruta de la carpeta de la plantilla[37] que será usada para estructurar el menú y la ruta de la carpeta de texto donde el *plug-in* almacenará las imágenes PNG con el texto de los elementos del menú.
- **`vistas:`** Atributo de tipo `QList<Vista*>*`. Consiste en una colección con referencias a instancias de la clase `Vista`.
- **`textos:`** Atributo de tipo `QList<QList<Texto*>*>*`. Consiste en una colección matricial con referencias a instancias de la clase `Texto`.

2.5.2.3 Clase Vista

La clase `Vista` representa un fragmento de un menú. Esta clase permite que se presenten al televidente solo determinadas filas y columnas de un menú, permitiendo optimizar el espacio disponible en el televisor.

Los atributos de esta clase son:

- **`id:`** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único que es asignado por la instancia de la clase `Menu` a la que pertenece la vista. El identificador alfanumérico único de esta clase viene dado por el identificador de la instancia de la clase `Menu` de nivel superior (m#) al que la vista corresponde, y se añade una letra v seguida de un valor numérico (m#v#) correspondiente al número de vista del menú. Las vistas son numeradas empezando desde uno. Así, la primera vista del primer menú diseñado por el usuario tendrá el identificador m1v1, la segunda vista el identificador m1v2, y así consecutivamente. De igual manera, la primera vista del segundo menú diseñado tendría el identificador m2v1, la segunda el identificador m2v2, etc. Este código además de permitir identificar el objeto de manera unívoca, permite determinar la posición de la vista dentro del menú.

[37] Una plantilla consiste en un conjunto de imágenes PNG que el *plug-in* utiliza para generar un menú diseñado por el usuario. Ver sección 2.6.1.

- **rutas:** Atributo de tipo `QMap<QString,QString>`. Referencia a la colección de rutas de la instancia de la clase `Menu` (atributo `rutas`) a la que corresponde la instancia de la clase `Vista`.
- **propiedadesVista:** Atributo de tipo `QMap<QString,int>`. Almacena, en tiempo de ejecución, los pares `<nombre_propiedad,valor>` que se presentan en la Tabla 2.11.

Tabla 2.11 Propiedades del atributo `propiedadesVista` de la clase `Vista`

Propiedad	Descripción
iX	Número de columna del menú desde la que inicia la vista.
fX	Número de columna del menú en que termina la vista aumentado en una unidad.
iY	Número de fila del menú desde la que inicia la vista.
fY	Número de fila del menú en que termina la vista aumentado en una unidad.
superior	Asume el valor 0 si la vista no posee la viñeta para navegar hacia arriba, y el valor 1 si la posee.
inferior	Asume el valor 0 si la vista no posee la viñeta para navegar hacia abajo, y el valor 1 si la posee.
izquierda	Asume el valor 0 si la vista no posee la viñeta para navegar hacia la izquierda, y el valor 1 si la posee.
derecha	Asume el valor 0 si la vista no posee la viñeta para navegar hacia la derecha, y el valor 1 si la posee.

- **botones:** Atributo de tipo `QList<Boton*>*`. Consiste en una colección con referencias a instancias de la clase `Boton`.
- **propiedadesMenu:** Atributo de tipo `QMap<QString,float>`. Referencia a la colección de propiedades de la instancia de la clase `Menu` (atributo `propiedadesMenu`) a la que corresponde la instancia de la clase `Vista`.
- **altoFilas:** Atributo de tipo `QList<float>*`. Referencia a la colección de la altura de cada fila de la instancia de la clase `Menu` (atributo `altoFilas`) a la que corresponde la instancia de la clase `Vista`.

- **anchoColumnas:** Atributo de tipo `QList<float>*`. Referencia a la colección del ancho de cada columna de la instancia de la clase `Menu` (atributo `anchoColumnas`) a la que corresponde la instancia de la clase `Vista`.

La Figura 2.25 presenta un ejemplo de la distribución de las vistas en un menú y cómo se numeran de manera secuencial. Para la Vista 1 se muestran las propiedades `iX`, `fX`, `iY` y `fY`, cuyos valores se almacenarían en la colección `propiedadesVista` para determinar los límites de la vista. Así, `iX` corresponde a la primera columna, `fX` a la tercera columna, `iY` a la primera fila y `fY` a la tercera fila.

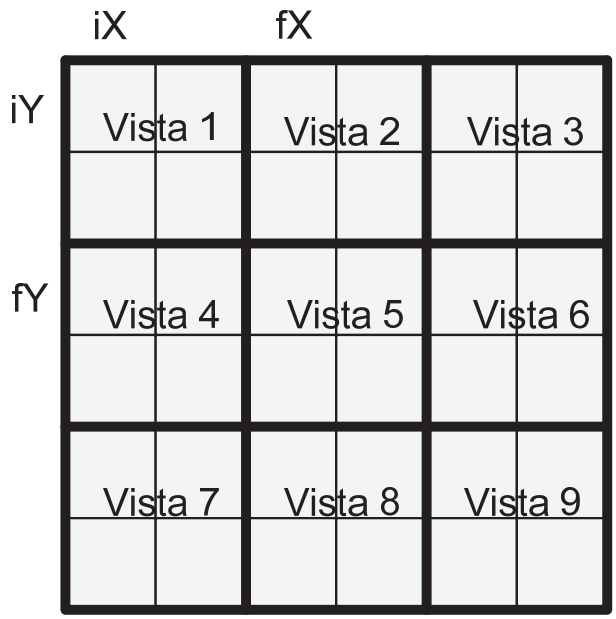


Figura 2.25 Ejemplo de división de un menú en vistas: Menú de seis filas y seis columnas dividido en vistas de dos filas y dos columnas

2.5.2.4 Clase Boton

La clase `Boton` modela un componente de un menú, sea este una viñeta de navegación o un elemento representando una alternativa que el televidente puede seleccionar.

Los atributos de esta clase son:

- **id:** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único que es asignado por la instancia de la clase `Vista` a la que pertenece el botón. Este identificador corresponde al identificador alfanumérico de la instancia de la clase `Vista` añadida la letra f, seguida de un valor numérico (que inicia desde 1) que representa la fila que ocupa el elemento en el menú. Luego, se añade la letra c seguida de un valor numérico (que inicia desde uno) que indica la columna que ocupa el elemento dentro del menú. Así, se obtiene un identificador con el formato m#v#f#c#. Para el caso especial de las viñetas de navegación, el número correspondiente a la fila (número después de la letra f) se establece en cero y el valor de la columna (número después de la letra c) indica si se trata de la viñeta superior (1), inferior (2), izquierda (3) o derecha (4).
- **rutas:** Atributo de tipo `QMap<QString,QString>`. Referencia a la colección de rutas de la instancia de la clase `Vista` (atributo `rutas`) a la que corresponde la instancia de la clase `Boton`.
- **numeroCuadro:** Atributo de tipo `int`. Es un contador incremental en pasos de uno a partir del cual se construye el identificador alfanumérico de cada instancia de la clase `Cuadro`.
- **propiedadesBoton:** Atributo de tipo `QMap<QString,float>`. Almacena, en tiempo de ejecución, los pares `<nombre_propiedad,valor>` que se presentan en la Tabla 2.12.

Tabla 2.12 Propiedades del atributo `propiedadesBoton` de la clase `Boton`

Propiedad	Descripción
ancho	Ancho del botón en porcentaje (% relativo a la vista).
alto	Alto/altura del botón en porcentaje (% relativo a la vista).
posX	Posición horizontal del botón en porcentaje (% relativo a la vista) establecido desde el borde izquierdo.
posY	Posición vertical del botón en porcentaje (% relativo a la vista) establecido desde la parte superior.

- **propiedadesVista:** Atributo de tipo `QMap<QString,int>`. Referencia a la colección de propiedades de la instancia de la clase `Vista` (atributo `propiedadesVista`) a la que corresponde la instancia de la clase `Boton`.

- **propiedadesMenu:** Atributo de tipo `QMap<QString,float>`. Referencia a la colección de propiedades de la instancia de la clase `Vista` (atributo `propiedadesMenu`) a la que corresponde la instancia de la clase `Boton`.
- **enlaces:** Atributo de tipo `QList<Enlace*>*`. Consiste en una colección de diez referencias a instancias de la clase `Enlace`.
- **cuadros:** Atributo de tipo `QList<Cuadro*>*`. Consiste en una colección con dos referencias a instancias de la clase `Cuadro`, si el botón se trata de una viñeta de navegación, o una colección con tres referencias a instancias de la clase `Cuadro`, si el botón representa un elemento del menú con una alternativa que el televidente puede seleccionar.

Las viñetas de navegación se han diseñado para que ocupen 7% del alto de la vista, si se trata de las viñetas de navegación superior o inferior; y, 7% del ancho de la vista, si se trata de las viñetas de navegación hacia la izquierda o derecha. De esta manera se garantiza que las viñetas de navegación tengan un tamaño adecuado acorde a las dimensiones de la vista.

El porcentaje de ocupación de las viñetas se resta del tamaño de la vista, tanto en alto como ancho, para así distribuir el espacio restante entre los distintos elementos. La Figura 2.26 ilustra la distribución de las dimensiones de los elementos y viñetas de navegación en una vista.

2.5.2.5 Clase Cuadro

La clase `Cuadro` modela una sola de las tres imágenes PNG que conforman un elemento de un menú, o una de las dos imágenes PNG que conforman una viñeta de navegación.

Para poder mostrar una imagen PNG en el televisor, es necesario crear: una región que defina la ubicación y dimensiones del área en donde se localizará la imagen; un nodo de contenido asociado con la imagen a mostrarse en pantalla; y, un descriptor que sirva como nexo entre la región y el nodo de contenido, es decir, que indique qué nodo de contenido colocar en una región determinada.

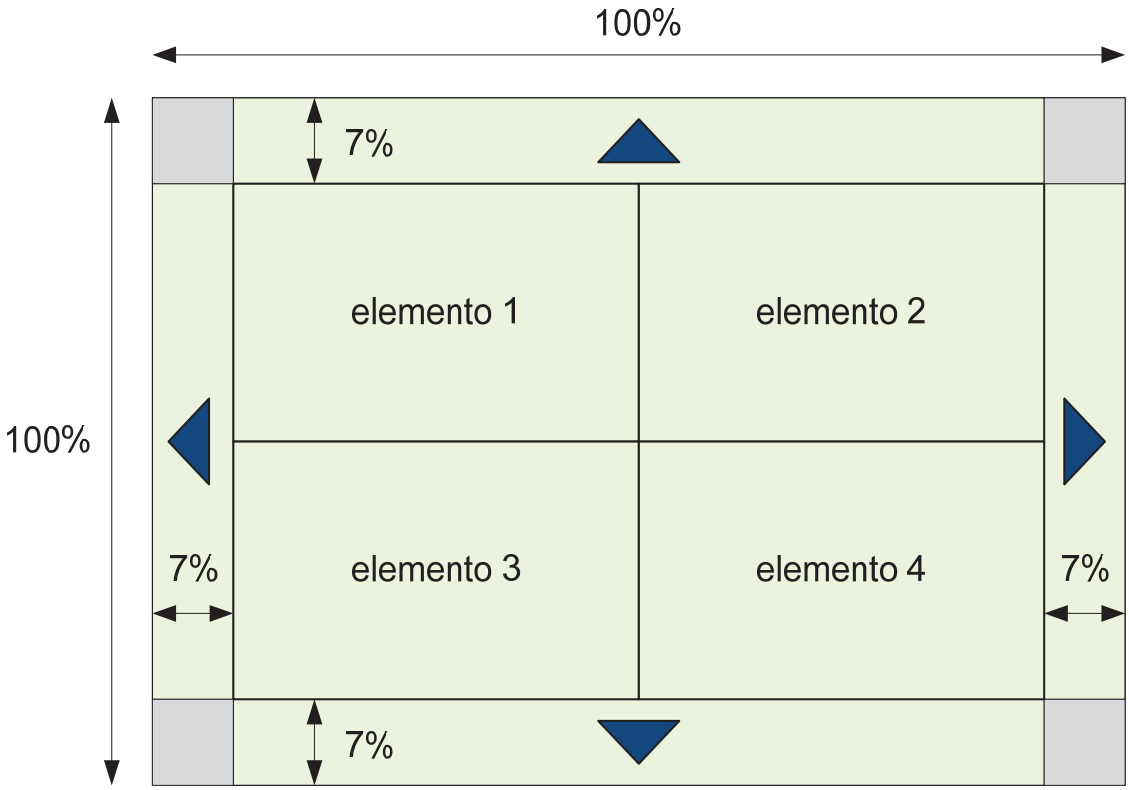


Figura 2.26 Distribución de las dimensiones de los elementos y viñetas de navegación en una vista

Cada uno de los elementos que conforman un menú diseñado por el usuario necesita de tres imágenes:

- **Imagen de fondo:** Aparece en el fondo del elemento del menú, detrás del texto.
- **Imagen de foco:** Aparece sobre la imagen de fondo y bajo la imagen de texto cuando el televidente se posiciona sobre el elemento del menú a través del control remoto.
- **Imagen de texto:** Corresponde al texto que aparece en el elemento del menú. Esta es una imagen completamente transparente a excepción del texto que contiene[38].

Cada imagen se ubica en una región distinta, pero las regiones de las tres imágenes correspondientes a un mismo elemento de un menú están

[38] Por esta razón se emplea el formato PNG, ya que permite obtener imágenes conformadas por píxeles con transparencia.

sobrelapadas, es decir, tienen las mismas dimensiones y posición en la pantalla del televisor.

Para la conformación y correcto funcionamiento de cada uno de los elementos del menú, se deben posicionar las tres imágenes de cada elemento haciendo uso del atributo `zIndex` de las regiones. La imagen de fondo será la que se encuentre debajo de otras, por lo que la región para esta imagen tendrá un valor de `zIndex` igual a 1, luego se encontrará la imagen de foco cuya región tendrá un valor de `zIndex` igual a 2. De esta manera, cuando el televidente se posicione en un determinado elemento del menú aparecerá la imagen foco superpuesta a la imagen de fondo indicando el elemento que puede seleccionar. Finalmente, la imagen de texto con transparencia se ubica sobre ambas imágenes asignando a su región un valor de `zIndex` igual 3.

De esta manera, mientras el televidente navega entre los elementos del menú, la imagen de fondo y de texto no desaparecen y persisten durante toda la interacción con el usuario. La imagen de foco se colocará sobre la imagen de fondo únicamente cuando el televidente se posicione sobre el elemento.

El valor de `zIndex` igual a 0 se ha reservado para que el usuario pueda colocar elementos en regiones que deban ir detrás del menú, como por ejemplo la imagen de fondo principal de la aplicación interactiva.

Este mecanismo de trabajo se detalla en la Figura 2.27. Se puede observar que cada uno de los elementos del menú está conformado por las tres imágenes descritas anteriormente, de las cuales la imagen de fondo y de texto persisten constantemente y la imagen de foco aparece (acción `start`) o se detiene (acción `stop`), permitiendo al televidente navegar entre los distintos elementos y visualizar el elemento que está seleccionado.

Este diseño presenta una excepción para las viñetas de navegación que permiten navegar entre las distintas vistas de un menú que ha sido dividido.

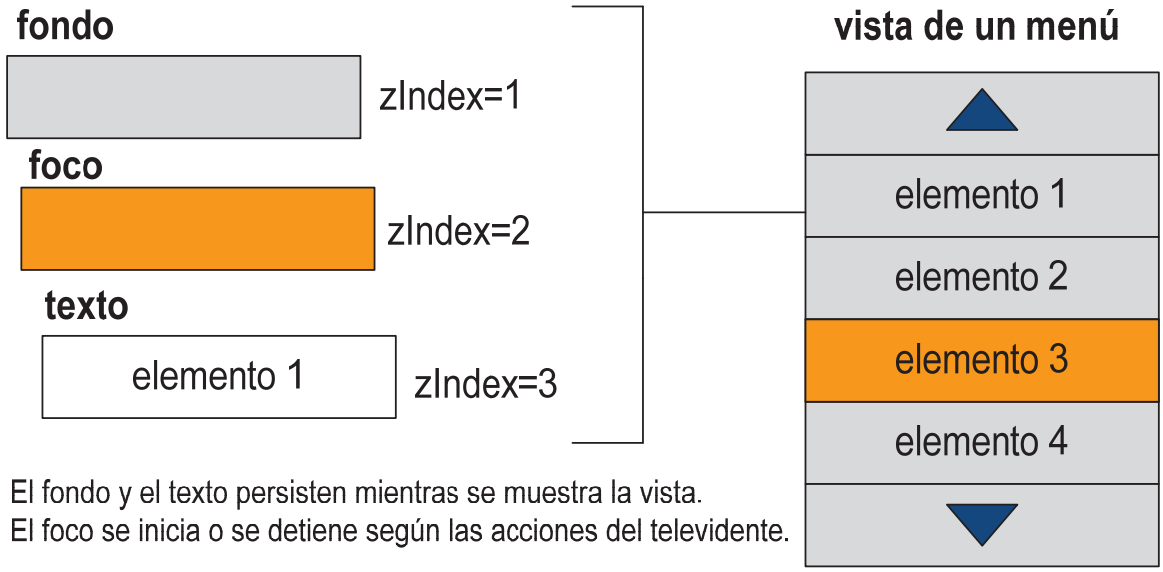


Figura 2.27 Conformación de un elemento de un menú

Los elementos de cada vista se estructuran de la misma manera que en el caso anterior, pero las viñetas de navegación están conformadas únicamente por la imagen de fondo y la de foco, es decir carecen de la imagen de texto. El símbolo gráfico que sugiere al televidente cómo desplazarse entre las distintas vistas se encuentra tanto en la imagen de fondo como en la del foco. La Figura 2.28 ilustra esta explicación.

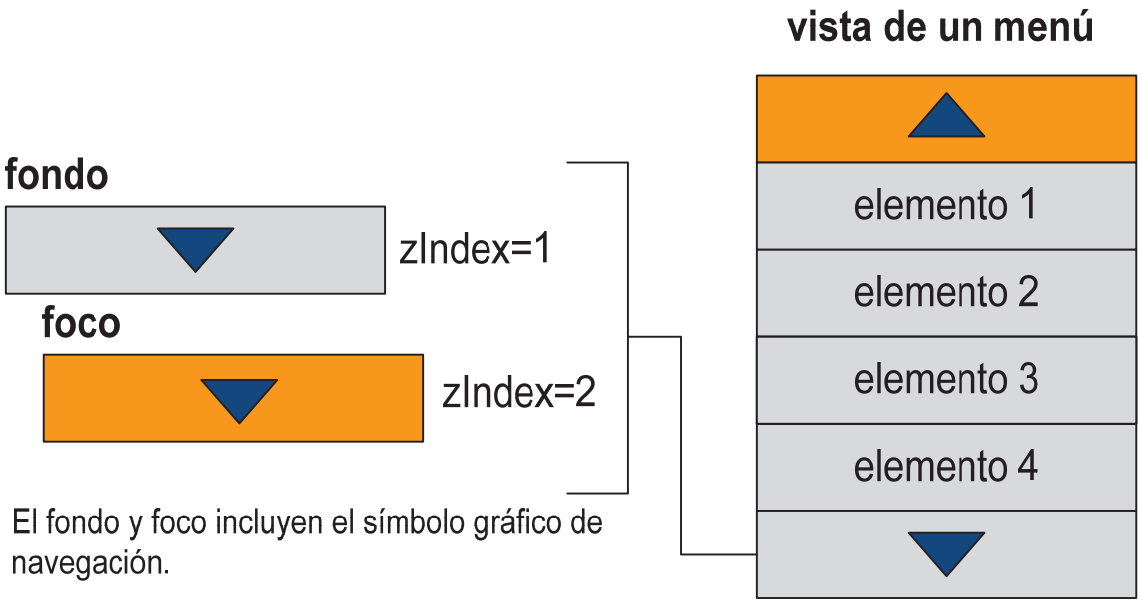


Figura 2.28 Conformación de una viñeta de navegación

Los atributos de la clase `Cuadro` son:

- **id:** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único que es asignado por la instancia de la clase `Boton` a la que pertenece la instancia de la clase `Cuadro`. Este identificador corresponde al identificador alfanumérico de la instancia de la clase `Boton` añadida la letra k para hacer referencia a un cuadro[39]. Luego, se añade un valor numérico que puede ser de 1 (si el cuadro representa la imagen de foco), 2 (si el cuadro representa la imagen de fondo) o 3 (si el cuadro representa la imagen de texto), obteniéndose así un identificador alfanumérico con el formato (m#v#f#c#k#).
- **zIndex:** Atributo de tipo `int`. Corresponde al valor de la propiedad `zIndex` de la región del cuadro.
- **puerta:** Atributo de tipo `bool`. Toma un valor de `true` si es necesario crear una puerta en el nodo de contexto para la imagen PNG asociada al cuadro, y un valor de `false` si no es necesario.
- **ruta:** Atributo de tipo `QString`. Corresponde al valor de la propiedad `src` del nodo de contenido correspondiente a la imagen PNG.
- **parametrosRegion:** Atributo de tipo `QMap<QString,QString>`. Es una colección con las propiedades de la región en donde se ubicará la imagen PNG. Estas propiedades son necesarias para construir la entidad y su respectivo código NCL.
- **parametrosDescriptor:** Atributo de tipo `QMap<QString,QString>`. Es una colección con las propiedades del descriptor que sirve para asociar la región del cuadro con el nodo de contenido asociado a la imagen PNG. Estas propiedades son necesarias para construir la entidad y su respectivo código NCL.
- **parametrosMedio:** Atributo de tipo `QMap<QString,QString>`. Es una colección con las propiedades del nodo de contenido asociado a la imagen PNG. Estas propiedades son necesarias para construir la entidad y su respectivo código NCL.

[39] Se emplea la letra k en lugar de la letra c, ya que esta última se utilizó para denotar una columna anteriormente.

- **`propiedadesCuadro:`** Atributo de tipo `QMap<QString,float>`. Almacena, en tiempo de ejecución, los pares `<nombre_propiedad,valor>` que se presentan en la Tabla 2.13.

Tabla 2.13 Propiedades del atributo `propiedadesCuadro` de la clase `Cuadro`

Dato	Descripción
`ancho`	Ancho de la imagen expresado como un porcentaje en relación al ancho de la pantalla del televisor.
`alto`	Alto de la imagen expresado como un porcentaje en relación al alto/altura de la pantalla del televisor.
`posX`	Posición horizontal de la imagen a mostrarse, expresada como un porcentaje establecido desde el borde izquierdo de la pantalla del televisor.
`posY`	Posición vertical de la imagen a mostrarse, expresada como un porcentaje establecido desde el borde superior de la pantalla del televisor.

- **`propiedadesBoton:`** Atributo de tipo `QMap<QString,float>`. Referencia a la colección de propiedades de la instancia de la clase `Boton` (atributo `propiedadesBoton`) a la que corresponde la instancia de la clase `Cuadro`.
- **`propiedadesVista:`** Atributo de tipo `QMap<QString,int>`. Referencia a la colección de propiedades de la instancia de la clase `Boton` (atributo `propiedadesVista`) a la que corresponde la instancia de la clase `Cuadro`.
- **`propiedadesMenu:`** Atributo de tipo `QMap<QString,float>`. Referencia a la colección de propiedades de la instancia de la clase `Boton` (atributo `propiedadesMenu`) a la que corresponde la instancia de la clase `Cuadro`.

2.5.2.6 Clase Enlace

La clase `Enlace` modela una entidad NCL de tipo `link` para un determinado botón del control remoto.

Los atributos de esta clase son:

- **id:** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único que es asignado por la instancia de la clase `Boton` a la que pertenece la instancia de la clase `Enlace`. Este identificador corresponde al identificador alfanumérico de la instancia de la clase `Boton` añadida la letra l (que hace referencia a `link`), seguida de un valor numérico que depende del botón del control remoto, como sugiere la Tabla 2.14. De esta manera se obtiene un identificador alfanumérico con el formato m#v#f#c#l#.

Tabla 2.14 Valor numérico según el botón del control remoto

Botón del control remoto	Valor numérico
Arriba	1
Abajo	2
Izquierda	3
Derecha	4
Rojo	5
Verde	6
Amarillo	7
Azul	8
ESC	9
OK	10

- **botonRemoto:** Atributo de tipo `int`. Establece el control del botón de acuerdo a los valores establecidos en la Tabla 2.14.
- **listaParametrosEnlace:** Atributo de tipo `QList<QMap<QString,QString>>*`. Almacena en tiempo de ejecución los pares `<rol,participante>` que indican las acciones a ejecutarse sobre determinados participantes cuando el televidente presione el botón del control remoto indicado en el atributo `botonRemoto`.

2.5.2.7 Clase Texto

La clase `Texto` modela el texto de un elemento de un menú y permite la generación de la imagen PNG con el texto.

Los atributos de esta clase son:

- **idBoton:** Atributo de tipo `QString`. Corresponde al identificador alfanumérico único para una instancia de la clase `Texto`. Este identificador corresponde al identificador alfanumérico de la instancia de la clase `Boton` al cual pertenece el texto, exceptuando la vista (se retira la letra v y su número siguiente). Así, se obtiene un identificador con el formato m#f#c#.
- **rutaTexto:** Atributo de tipo `QString`. Corresponde a la ruta a la carpeta de texto en donde el *plug-in* almacenará la imagen generada en formato PNG.
- **cambios:** Atributo de tipo `bool`. Toma el valor de `true` cuando se ha realizado un cambio en el texto y se requiere volver a generar la imagen. El valor de `false` indica que no se han realizado cambios.
- **propiedadesTexto:** Atributo de tipo `QMap<QString,QString>`. Almacena, en tiempo de ejecución, los pares `<nombre_propiedad,valor>` que se presentan en la Tabla 2.15. Con estas propiedades el *plug-in* genera el texto de un elemento del menú.

Tabla 2.15 Propiedades del atributo `propiedadesTexto` de la clase `Texto`

Propiedad	Descripción
fuente	Fuente que se emplea para generar las imágenes de texto.
color	Color del texto expresado en valor hexadecimal.
tamanio	Tamaño del texto en puntos.
anchoLienzo	Ancho del lienzo de la imagen en píxeles.
altoLienzo	Alto del lienzo de la imagen en píxeles.

- **propiedadesTextoDefecto:** Atributo de tipo `QMap<QString,QString>`. Referencia a la colección de propiedades de la instancia de la clase `Estructura` (atributo `propiedadesTextoDefecto`), que establece los valores por defecto con los cuales se genera el texto de un elemento del menú.

2.5.3 IMPLEMENTACIÓN DE LOS MÉTODOS DE LA INTERFAZ IPLUGINFACTORY

Para implementar los métodos de la interfaz `IPluginFactory` se debe crear una clase, por convención, llamada con el nombre del *plug-in* seguido de la palabra `Factory`. Por lo tanto, se ha creado la clase `MenuCreatorFactory`, la cual hereda de la interfaz `IPluginFactory` implementada en el composer-core. La Figura 2.29 muestra el diagrama de clase de `MenuCreatorFactory` implementada en el *plug-in* Menu Creator.

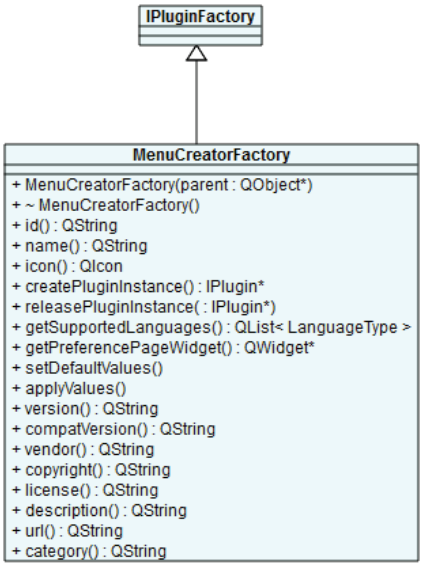


Figura 2.29 Clase `MenuCreatorFactory`

La clase `MenuCreatorFactory` realiza dos funciones: la de ofrecer información sobre el *plug-in* y la de crear y eliminar instancias del mismo. La información que se ofrece en la clase `MenuCreatorFactory` se presenta en la Tabla 2.16, detallando el método implementado para proveer un dato específico del *plug-in* Menu Creator.

Para la creación de una nueva instancia del *plug-in* se implementa el método `createPluginInstance()` de la interfaz, el cual retorna una instancia de la clase `MenuCreatorPlugin`[40], como se presenta en el Código 2.4.

[40] La clase `MenuCreatorPlugin` implementa los métodos de la interfaz `IPlugin`, como se describe en la siguiente sección.

Tabla 2.16 Información del *plug-in* Menu Creator especificada en la clase `MenuCreatorFactory`

Dato	Método implementado	Valor del dato
Identificador de *plug-in*	`QString id()`	epn.menucreator
Nombre del *plug-in*	`QString name()`	Menu Creator
Lenguajes soportados por el *plug-in*	`QList<LanguageType>getSupportedLanguages()`	NCL
Versión del *plug-in*	`QString version()`	0.1
Versión de compatibilidad	`QString compatVersion()`	0.1
Propietario	`QString vendor()`	Escuela Politécnica Nacional
Derechos	`QString copyright`	Escuela Politécnica Nacional/Quito-Ecuador
Licencia	`QString license()`	LGPL
Descipción	`QString description()`	Menu Creator permite la generación automática de código NCL de menús para Televisión Digital. Proyecto de titulación Desarrollado por: - David Cevallos Salas - Fernando Cevallos Salas Director: Ph.D. Iván Bernal Codirector: Ing. David Mejía, MSc. Escuela Politécnica Nacional Quito - Ecuador
Url para más información	`QString url()`	http://ginga.epn.edu.ec
Categoría	`QString category()`	NCL

```
IPlugin* MenuCreatorFactory::createPluginInstance()
{
    return new MenuCreatorPlugin();
}
```

Código 2.4 Método `createPluginInstance()`

Para eliminar una instancia del *plug-in* se implementa el método `releasePluginInstance()`, que destruye el objeto que se pasa como argumento, como se presenta en el Código 2.5.

```
void MenuCreatorFactory::releasePluginInstance(IPlugin *plugin)
{
   delete(plugin);
}
```

Código 2.5 Método `releasePluginInstance()`

2.5.4 IMPLEMENTACIÓN DE LOS MÉTODOS DE LA INTERFAZ IPLUGIN

Para implementar los métodos de la interfaz `IPlugin` se debe crear una clase, por convención, llamada con el nombre del *plug-in* seguido de la palabra `Plugin`. Esta clase debe heredar de la interfaz `IPlugin`, implementada en el composer-core. Por lo tanto, para el *plug-in* Menu Creator se ha implementado la clase `MenuCreatorPlugin`, cuyo diagrama de clase se presenta en la Figura 2.30.

La clase `MenuCreatorPlugin` tiene como atributos una referencia a la ventana principal del *plug-in* (atributo `window`), una referencia a la ventana de progreso (atributo `progreso`) que indica el porcentaje culminado de las tareas en la generación del código NCL de un menú; y referencias a las entidades básicas del proyecto Composer como la base de regiones (atributo `regionBase`), la base de descriptores (atributo `descriptorBase`) y el cuerpo del documento NCL (atributo `body`).

Los métodos de la interfaz `IPlugin` que se han implementado en la clase `MenuCreatorPlugin` se detallan en la Tabla 2.17.

Para el caso de Menu Creator, la clase `MenuCreatorPlugin` implementa los métodos necesarios para la gestión de las entidades NCL. Los métodos tienen como objetivo la gestión masiva del árbol jerárquico permitiendo añadir, editar y eliminar las entidades necesarias para crear menús.

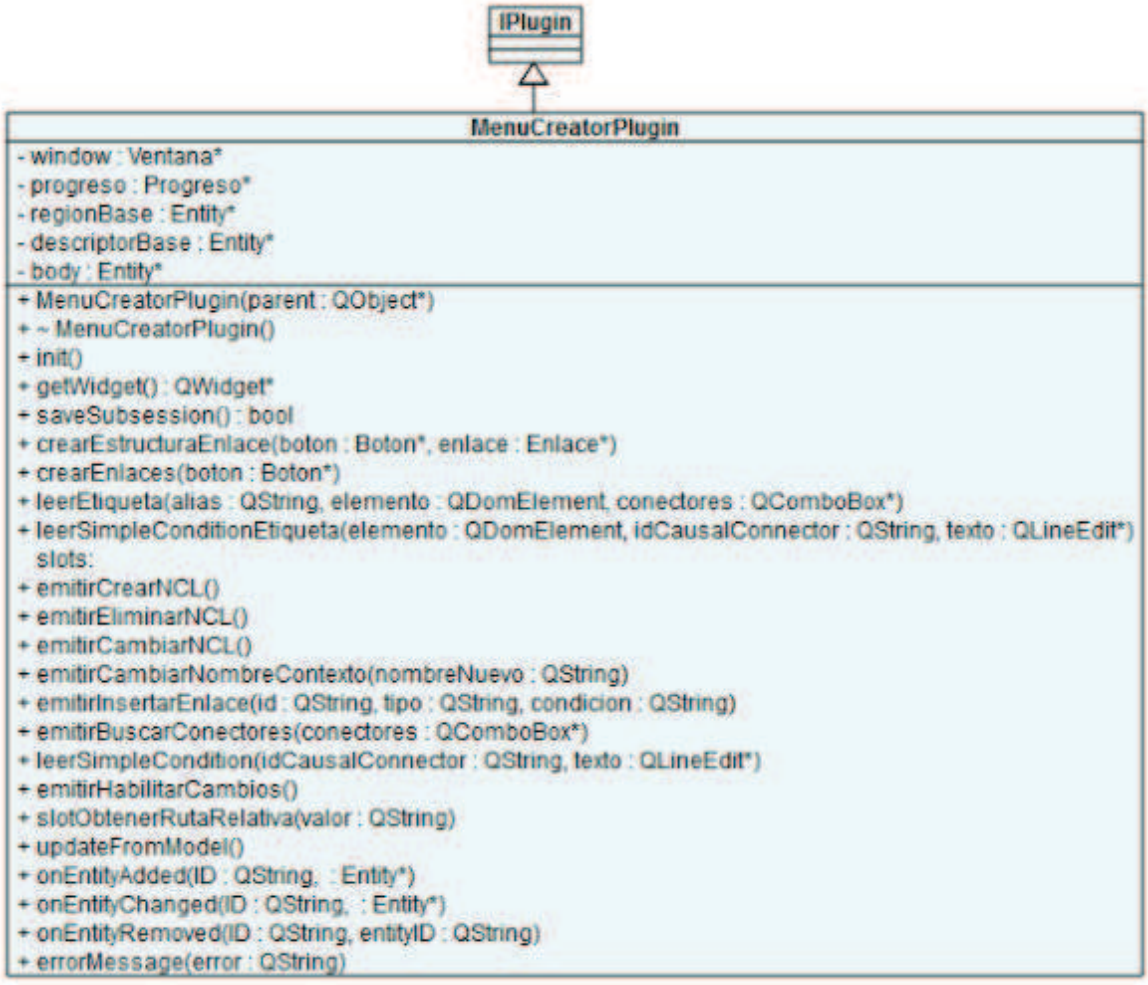


Figura 2.30 Clase `MenuCreatorPlugin`

Tabla 2.17 Métodos implementados de la interfaz `IPlugin` en la clase `MenuCreatorPlugin`

Método	Funcionalidad
`void init()`	Solicita al composer-core los datos almacenados en el archivo .cpr y genera nuevamente las clases `Estructura`, `Menu` y `Texto`. A partir de estas tres clases invoca a los métodos necesarios para volver a generar las clases `Vista`, `Boton`, `Cuadro` y `Enlace`.
`QWidget* getWidget()`	Retorna la ventana principal del *plug-in* Menu Creator.
`bool saveSubsession()`	Llama al método `serializar()` de la clase `Estructura`, transformando su resultado en un array de *bytes* que se comunica al composer-core para su posterior almacenamiento en el archivo .cpr.

Estos métodos son implementados como *slots* que se ejecutan cuando desde la interfaz gráfica se emite la señal (*signal*) correspondiente a la acción que desea realizar el usuario. Un *slot* que se ejecuta desde la clase `MenuCreatorPlugin` emite un conjunto de señales (*signals*) que, mediante el mecanismo *signals and slots*, gestionan el árbol jerárquico de entidades. La Figura 2.31 muestra la comunicación entre la interfaz gráfica de usuario del *plug-in* Menu Creator y el composer-core a través de la clase `MenuCreatorPlugin`. Mientras esta comunicación se produce, se presenta al usuario una barra de progreso como la de la Figura 2.32, que indica el porcentaje ejecutado de las tareas que se están llevando a cabo para generar el código NCL de un menú diseñado.

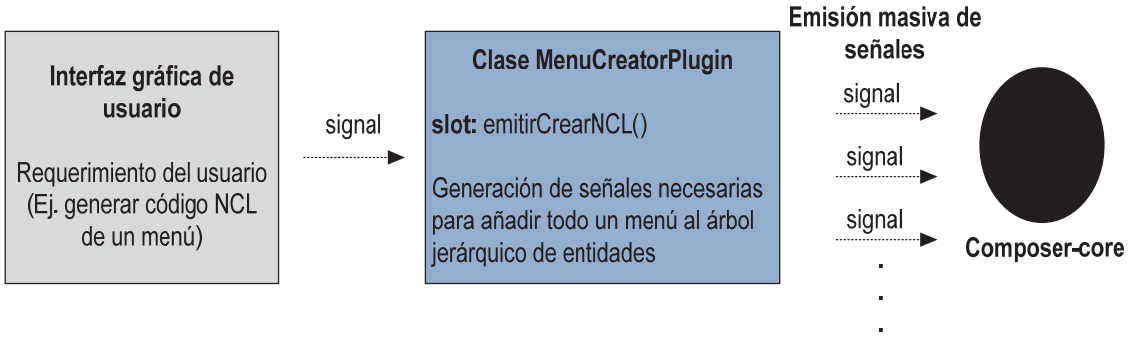


Figura 2.31 Comunicación entre la interfaz gráfica de usuario y el composer-core

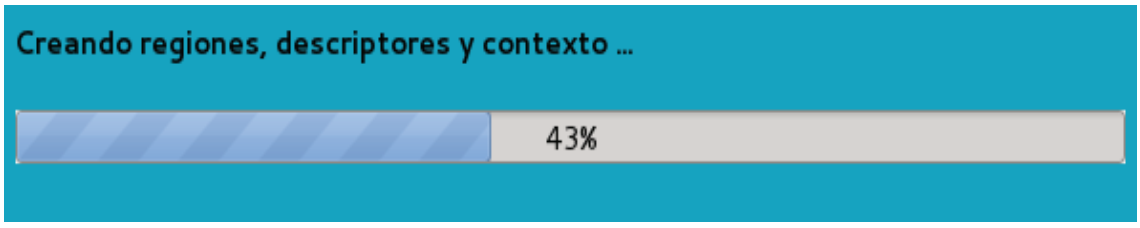


Figura 2.32 Barra de progreso

El tiempo que el composer-core tarde en añadir las entidades al árbol jerárquico para estructurar un menú dependerá, además del número de entidades requeridas, del número de *plug-ins* cargados en el IDE. Esto se debe a que cada vez que una entidad es gestionada (añadida, eliminada o modificada en el árbol), el composer-core comunica a todos los *plug-ins* mediante mensajes de control, las acciones realizadas sobre el árbol jerárquico de entidades. Los *plug-ins* deben ejecutar los *slots* correspondientes para actualizar sus estados y sus interfaces gráficas.

En la Tabla 2.18 se detallan los *slots* encargados de la gestión de las entidades NCL, implementados en la clase `MenuCreatorPlugin`.

Tabla 2.18 *Slots* implementados en la clase `MenuCreatorPlugin` para la gestión de las entidades NCL

Slot	Funcionalidad
`void emitirCrearNCL()`	Elimina (si existe) y crea el código NCL de un menú.
`void emitirEliminarNCL()`	Elimina el código NCL de un menú.
`void emitirCambiarNCL()`	Si se han efectuado cambios en la posición o dimensiones (ancho y alto/altura) del menú o sus elementos, este *slot* cambia las regiones dejando intactos los descriptores y el nodo de contexto del menú. Si los cambios son a nivel del nodo de contexto, por ejemplo en el número de filas o columnas del menú, o el número de filas o columnas que se muestran por cada vista, se hace uso del *slot* `emitirCrearNCL()`.
`void emitirCambiarNombreContexto (QString nombreNuevo)`	Permite cambiar el valor atributo `id` del nodo de contexto del menú por el valor del parámetro `nombreNuevo`.
`void emitirInsertarEnlace(QString id, QString tipo, QString condicion)`	Inserta un enlace NCL para un determinado elemento de un menú. El parámetro `id` corresponde al identificador del enlace, el parámetro `tipo` determina el tipo de conector causal empleado y el parámetro `condicion` establece la condición que se debe cumplir para ejecutar las acciones del conector causal.

Los demás métodos y *slots* de la clase sirven de apoyo para los *slots* encargados de la gestión de las entidades NCL. Estos métodos y *slots* se detallan en la Tabla 2.19.

Tabla 2.19 Métodos y *slots* de apoyo implementados en la clase `MenuCreatorPlugin`

Método	Funcionalidad
`void crearEstructuraEnlace(Boton* boton, Enlace* enlace)`	Junto al método `crearEnlaces()`, crea los enlaces de navegación para cada uno de los menús. Este método creará la estructura general del enlace, exceptuando los roles de cada participante a ser detenidos (`stop`) o iniciados (`start`) cuando se cumpla la condición del enlace.
`void crearEnlaces(Boton* boton)`	Incluye en la estructura general, creada mediante el método `crearEstructuraEnlace()`, los roles de cada componente a ser detenidos (`stop`) o iniciados (`start`) cuando se cumpla la condición del enlace.
`void leerEtiqueta(QString alias, QDomElement elemento,QComboBox* conectores)`	Busca recursivamente los conectores causales en un nodo XML.
`void leerSimpleConditionEtiqueta(QDomElement elemento, QString idCausalConnector, QLineEdit* texto)`	Busca la etiqueta `simpleCondition` que le corresponde a un determinado conector causal en un nodo XML.
Slots	**Funcionalidad**
`void emitirBuscarConectores (QComboBox* conectores)`	Busca conectores causales dentro del árbol jerárquico de entidades y archivos externos .ncl. Es útil para leer el archivo defaultConnBase.ncl. Hace uso del método `leerEtiqueta()`.
`void leerSimpleCondition(QString idCausalConnector, QLineEdit* texto)`	Busca la etiqueta `simpleCondition` dentro del árbol jerárquico de entidades y archivos externos .ncl. Es útil para leer el archivo defaultConnBase.ncl. Hace uso del método `leerSimpleConditionEtiqueta()`.
`void emitirHabilitarCambios()`	Establece que se han realizado cambios en el proyecto y de ser necesario habilita el botón `Guardar` en el composer-gui.
`void slotObtenerRutaRelativa(QString valor)`	Obtiene la ruta relativa tomando como referencia la ubicación del proyecto actual respecto a una determinada ubicación que se pasa como argumento.

Los *slots* `updateFromModel()`, `onEntityAdded()`, `onEntityChanged()`, `onEntityRemoved()` y `errorMessage()` se declaran mas no se implementan en Menu Creator, porque estos métodos se ejecutan únicamente cuando el *plug-in* recibe un mensaje de control de parte del composer-core informando de un cambio en el árbol jerárquico de entidades.

2.5.5 INTERFAZ GRÁFICA DE USUARIO

Cada uno de los *plug-ins* dentro de NCL Composer tiene una interfaz gráfica de usuario (GUI) que se abre como una subventana dentro de la ventana principal del IDE (composer-gui). Cuando NCL Composer ha reconocido un *plug-in,* este llama a través del composer-core al método `getWidget()` que retorna un objeto de tipo `QWidget` correspondiente a la interfaz gráfica del *plug-in* que será presentada al usuario.

La Figura 2.33 muestra la interfaz gráfica del *plug-in* Menu Creator.

Figura 2.33 Interfaz gráfica del *plug-in* Menu Creator

La parte superior izquierda de la ventana está destinada a listar los menús diseñados con el *plug-in* y a mostrar las diferentes opciones que el *plug-in* ofrece a manera de botones. La parte derecha se utiliza para el ingreso de los datos dependiendo del botón que haya seleccionado el usuario. La parte inferior muestra el menú o los menús que se han diseñado y su posición en el televisor. Además, la interfaz gráfica permite al usuario arrastrar cada menú para determinar su posición y navegar entre los elementos del mismo.

La lista de menús y opciones permiten el diseño de cada menú y la generación del código NCL respectivo. Las opciones disponibles en esta zona son las siguientes:

- **Ayuda:** Abre el archivo de ayuda (manual de usuario) en el que se proporcionan indicaciones sobre el uso del *plug-in*.
- **Nuevo:** Pone a disposición del usuario la zona *"Ingreso de Datos"* para la creación de un nuevo menú.
- **Editar:** Permite editar las opciones referentes a la estructura del menú.
- **Eliminar:** Elimina el menú seleccionado y su código NCL, si ha sido generado.
- **NCL:** Permite generar el código NCL del menú seleccionado. En caso de existir código NCL asociado previo, este se elimina y se sobrescribe con el nuevo código.
- **Eliminar NCL:** Elimina el código NCL del menú sin eliminar los parámetros de diseño establecidos por el usuario en el *plug-in*.
- **Texto/Enlaces:** Pone a disposición del usuario la zona *"Ingreso de Datos"* para ingresar el texto de cada elemento del menú, gestionar sus propiedades e insertar enlaces NCL.

Cuando el usuario crea un nuevo menú, en primer lugar se diseña de manera lógica mediante la estructura jerárquica de clases, antes de traducirse al conjunto de entidades a ser añadidas de manera adecuada al árbol jerárquico de entidades del composer-core. Después de ser diseñado, el menú se muestra en la parte inferior de la ventana (navegador interactivo de menús), de manera que el usuario pueda interactuar con el menú diseñado teniendo algunas facilidades como:

- Visualización de la ubicación del menú, sugiriendo cómo se mostrará en el televisor.
- Navegación entre las vistas del menú.
- Selección del elemento del menú que se desea editar dando un clic sobre el mismo.
- Arrastrar el menú para poder determinar su ubicación en la pantalla del televisor.
- Trabajar con múltiples menús, permitiendo ocultar o mostrar a cada menú para facilidad en el diseño.

La Figura 2.34 presenta, en el navegador interactivo de menús, un menú construido de cuatro filas y cuatro columnas, el cual se muestra en vistas conformadas por dos filas y dos columnas. El usuario puede desplazarse entre las vistas dando clic en las viñetas de navegación, y seleccionar un elemento de cada vista dando clic sobre el mismo.

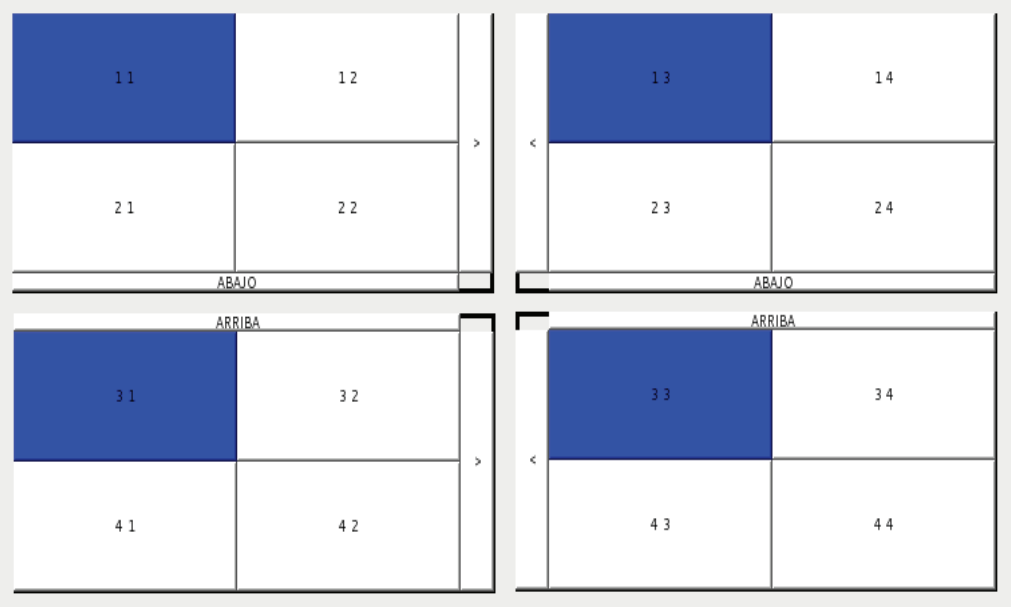


Figura 2.34 Navegación interactiva de menús

2.5.5.1 Construcción gráfica de menús

El *plug-in* muestra gráficamente las acciones que el usuario va realizando mientras diseña un menú. Para la construcción gráfica de los menús diseñados,

se ha empleado las funcionalidades de la biblioteca Qt usando un mecanismo de herencia para la clase `Menu`, `Boton` y `Texto`.

La clase `Menu` hereda de la clase `QFrame` y la clase `Boton` hereda de la clase `QPushButton`. Una instancia de la clase `QFrame` representa un contenedor de *Widgets*, mientras que una instancia de la clase `QPushButton` representa un botón de Qt. Sin embargo, aún con este mecanismo de herencia, las instancias de las clases `Menu` y `Boton` no pueden ser mostradas al usuario gráficamente porque necesitan ser invocadas y gestionadas desde una clase que provea propiedades gráficas al proyecto. Por lo tanto, la interfaz gráfica de usuario del *plug-in* es la encarga de realizar la invocación y gestión a las instancias de estas clases.

La Figura 2.35 ilustra este mecanismo, en el que la clase `Menu` y `Boton` heredan las propiedades de las clases `QFrame` y `QPushButton` respectivamente, para luego ser invocadas y gestionadas desde la interfaz gráfica de usuario del *plug-in* en tiempo de ejecución. De esta manera, las instancias de la clase `Menu` y `Boton` que se creen pueden ser usadas como *Widgets*.

Este mecanismo brinda una ventaja al *plug-in*, ya que se aprovecha la estructura jerárquica de clases en la que se basa Menu Creator para crear un sistema de generación de menús que pueda interactuar con el usuario, evitando la programación de código adicional para lograr este fin mediante la herencia de las clases propias de la biblioteca Qt.

De igual manera, para la clase `Texto` se ha considerado la herencia de la clase `QTextEdit`, para que pueda mostrarse un editor de texto multilínea que permita al usuario ingresar el texto correspondiente a cada elemento de un menú. En la Figura 2.36 se presenta el editor permitiendo ingresar el texto a ser presentado en cada uno de los elementos del menú.

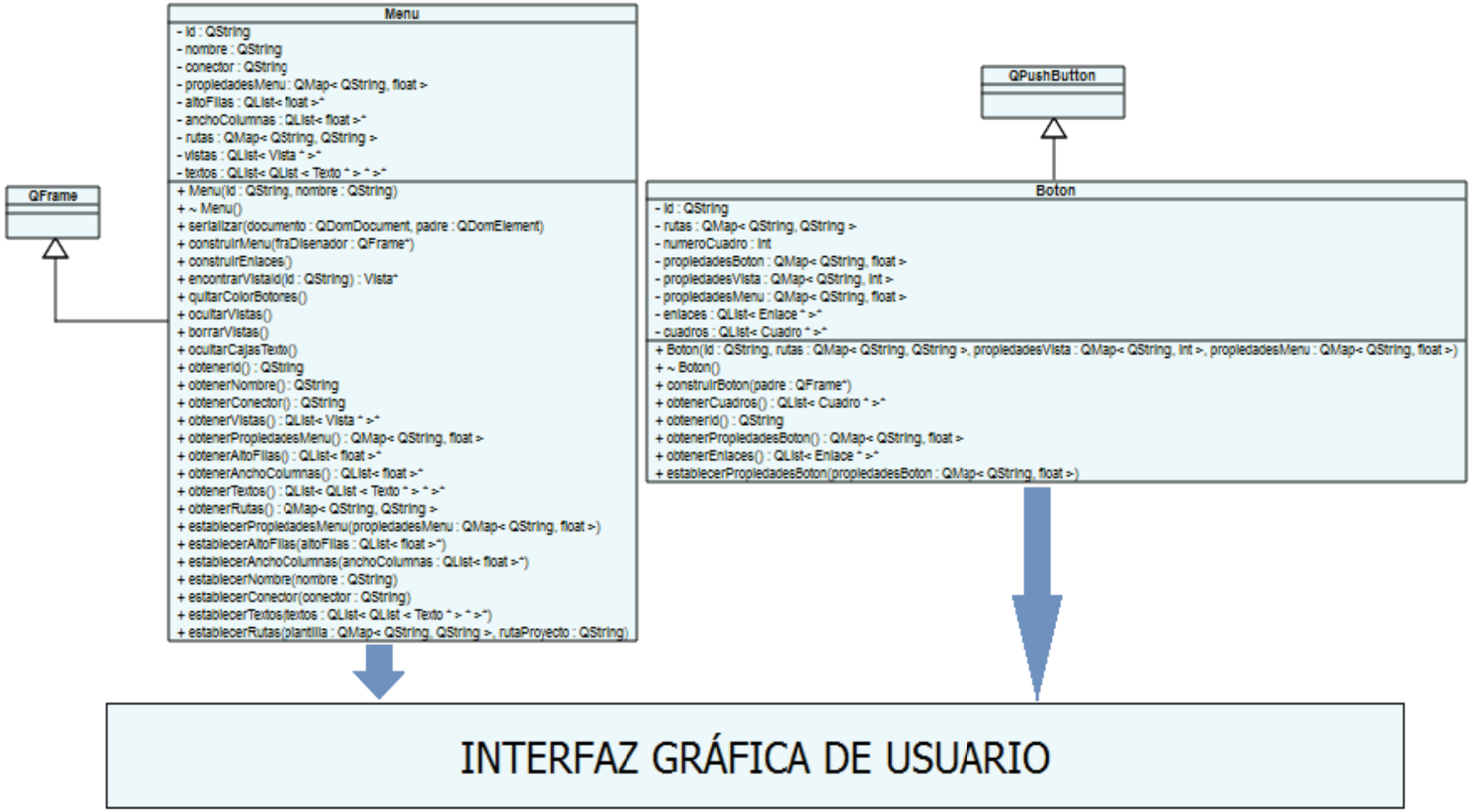


Figura 2.35 Relación de herencia entre las clases `Menu` y `Boton` con las clases `QFrame` y `QPushButton` de Qt respectivamente

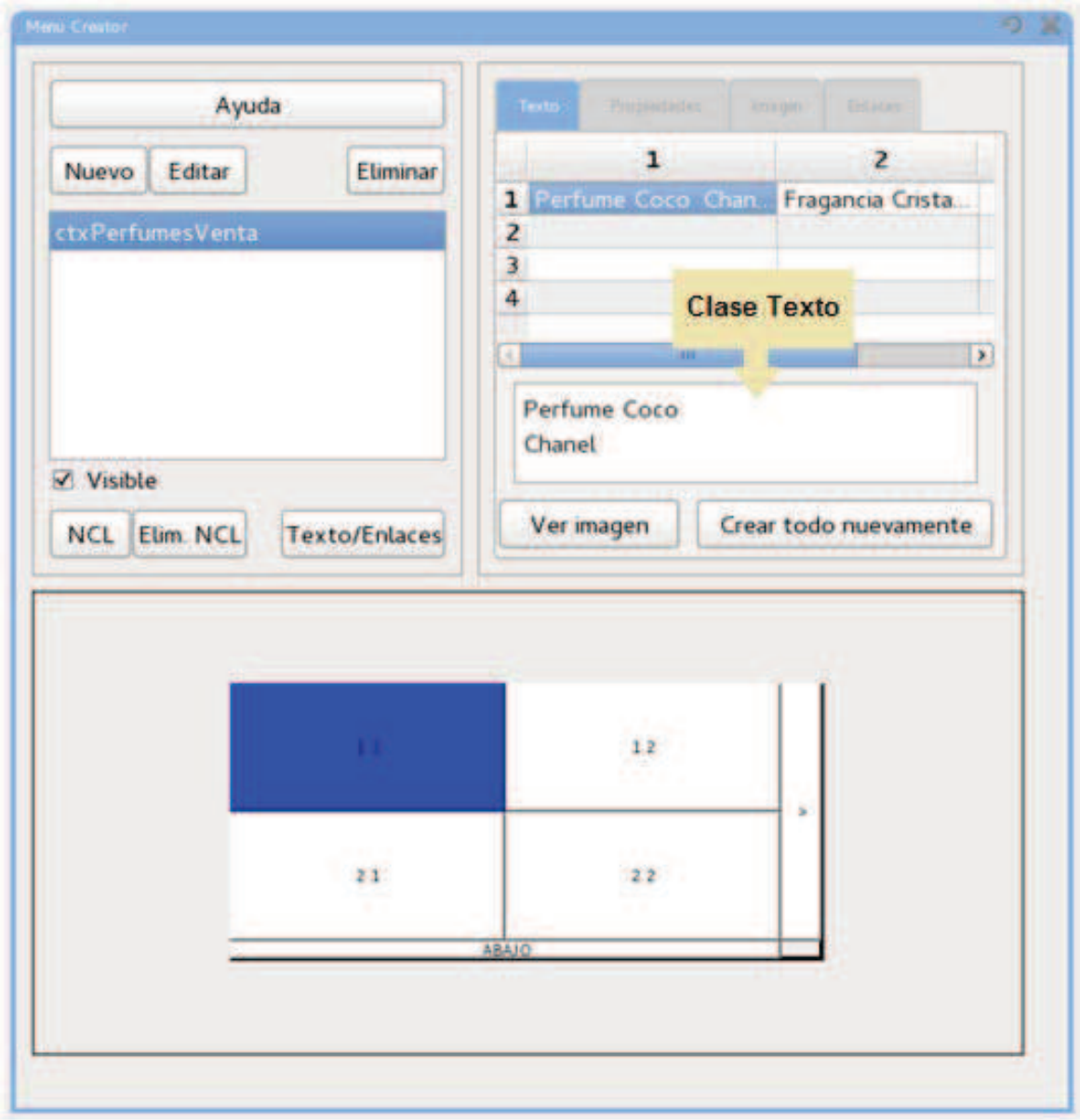


Figura 2.36 Editor de texto multilínea

De igual manera que en el caso anterior, aunque la clase `Texto` haya heredado de la clase `QTextEdit` no puede mostrarse gráficamente y requiere que la interfaz gráfica de usuario del *plug-in* la invoque y gestione para que pueda presentarse gráficamente. En la Figura 2.37 se muestra la clase `Texto` heredando de la clase `QTextEdit` de Qt para ser invocada y gestionada por la interfaz gráfica de usuario del *plug-in* en tiempo de ejecución.

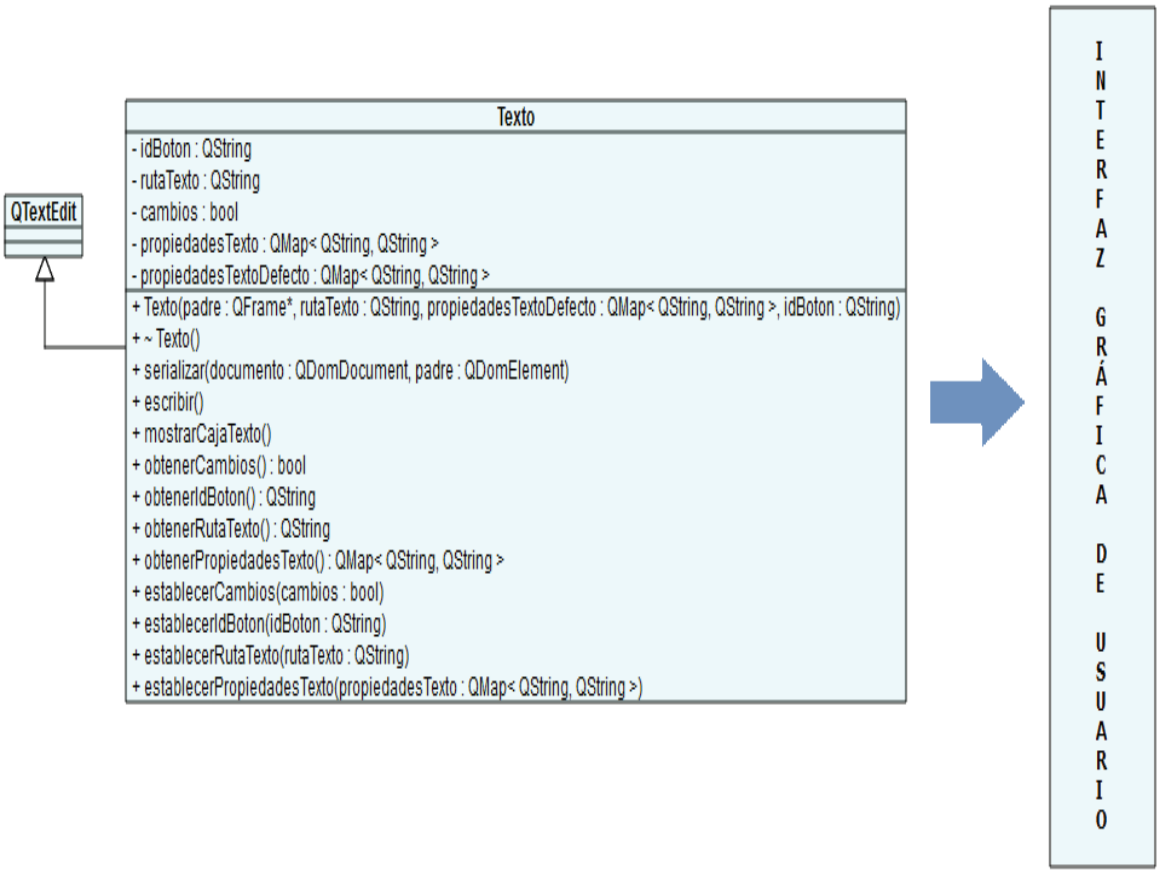


Figura 2.37 Relación de herencia entre la clase `Texto` con la clase `QTextEdit` de Qt

2.5.5.2 Módulos

La funcionalidad de Menu Creator puede ser resumida en tres módulos que proporcionan al usuario una potente herramienta para crear menús:

- Módulo de creación de estructura.
- Módulo de ingreso de texto.
- Módulo de inserción de enlaces NCL.

2.5.5.2.1 Módulo de creación de estructura

El módulo de creación de estructura permite definir los parámetros bajo los cuales se construirá un menú. Se compone de cuatro pestañas:

- **Parámetros:** Permite definir la ruta de la carpeta donde se encuentra la plantilla a ser utilizada (imágenes de fondo y foco del menú) y la ruta de la carpeta en la que se almacenarán las imágenes de texto generadas para cada elemento del menú. Además, permite seleccionar de un listado el conector causal a emplearse para posibilitar la navegación entre los elementos del menú, como se presenta en la Figura 2.38.

Figura 2.38 Pestaña *"Parámetros"* del módulo de creación de estructura

- **Propiedades:** Permite al usuario ingresar las propiedades de un menú tales como: el nombre (`id` del nodo del contexto), número de filas y de columnas, ancho y alto/altura del menú (expresados como un porcentaje respecto de la pantalla del televisor), número de filas y de columnas por vista, la posición horizontal del menú (expresado como un porcentaje respecto del ancho de la pantalla del televisor) y la posición vertical del menú (expresado como un porcentaje respecto del alto/altura de la pantalla del televisor). Esta pestaña se muestra en la Figura 2.39.

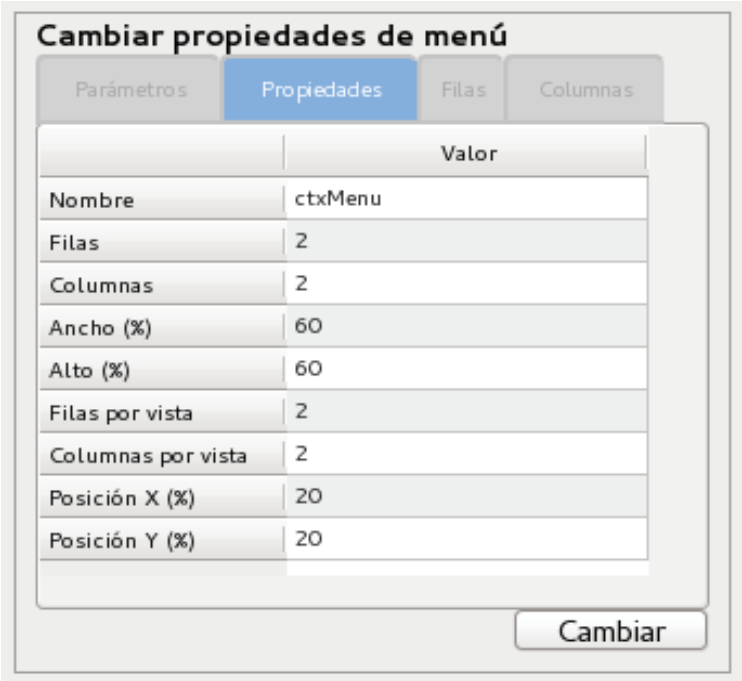


Figura 2.39 Pestaña *"Propiedades"* del módulo de creación de estructura

- **Filas:** Presenta y permite modificar, para cada una de las filas de una vista, su alto/altura expresado como un porcentaje respecto del alto de la vista. La vista de la que se desea visualizar el tamaño de sus filas puede ser seleccionada empleando el navegador interactivo de menús. Por defecto, cuando se crea una vista las filas tienen equitativamente el mismo alto/altura, como se muestra en la Figura 2.40.

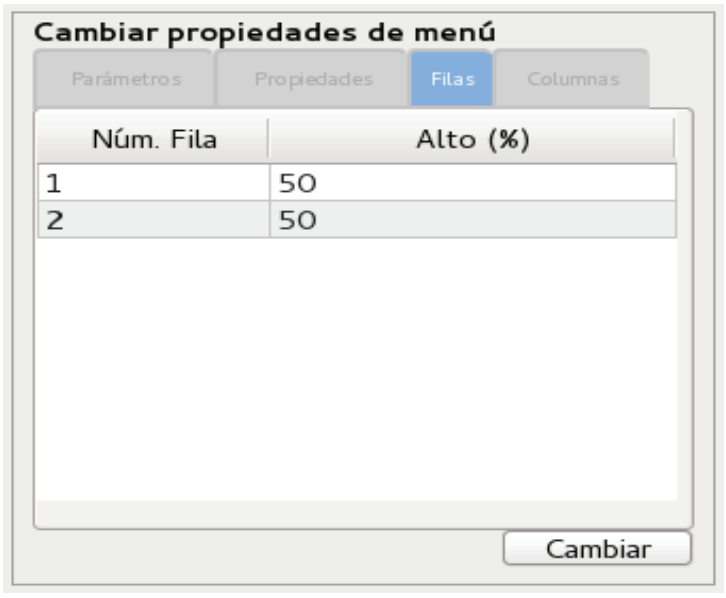


Figura 2.40 Pestaña *"Filas"* del módulo de creación de estructura

- **Columnas:** Realiza la misma acción que la pestaña *"Filas"*, pero para presentar y permitir modificar el ancho de las columnas de la vista actual. Esta pestaña se presenta en la Figura 2.41.

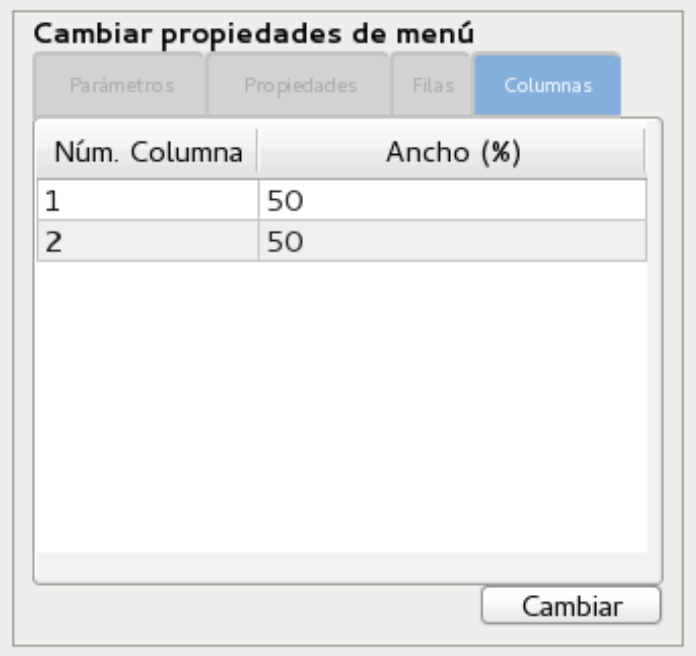


Figura 2.41 Pestaña *"Columnas"* del módulo de creación de estructura

2.5.5.2.2 Módulo de ingreso de texto

El *plug-in* cuenta con un módulo de ingreso de texto que permite introducir la información asociada a cada elemento de los menús diseñados y establecer las propiedades del texto (fuente, color, tamaño y dimensiones del lienzo). Además, genera una vista previa de la imagen a crearse con el texto.

Este módulo se compone de tres pestañas:

- **Texto:** Posibilita el ingreso del texto mediante un editor monolínea que permite ingresar la información correspondiente a cada elemento del menú en una sola línea. Además, brinda un editor de texto multilínea con el cual se puede ingresar el texto en varias líneas de un elemento seleccionado, como se presenta en la Figura 2.42.

Figura 2.42 Pestaña *"Texto"* del módulo de ingreso de texto

- **Propiedades:** Muestra las propiedades del texto como la fuente, el color, el tamaño de letra y las dimensiones del lienzo (ancho y alto/altura expresados en píxeles) que se emplean para generar la imagen, como se presenta en la Figura 2.43.

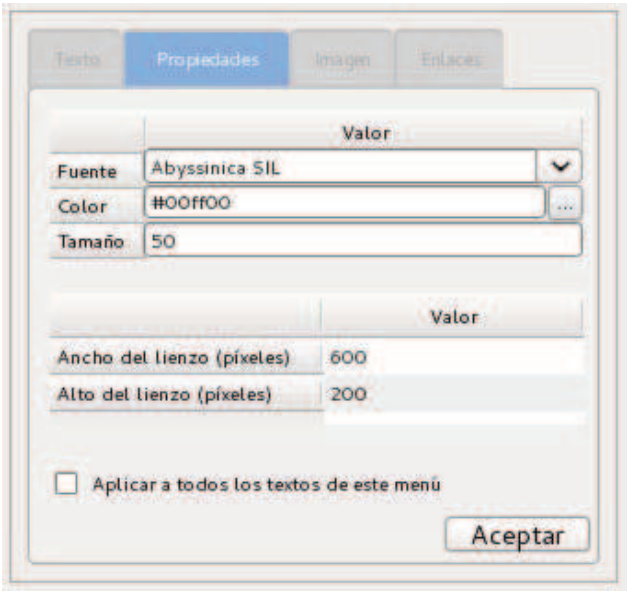


Figura 2.43 Pestaña *"Propiedades"* del módulo de ingreso de texto

- **Imagen:** Genera una vista previa de la imagen a crearse con el texto ingresado para un elemento de un menú con las propiedades seleccionadas, como se presenta en la Figura 2.44.

Figura 2.44 Vista previa del módulo de ingreso de texto

2.5.5.2.3 Módulo de inserción de enlaces NCL [7]

El módulo de inserción de enlaces NCL permite al usuario asociar elementos del menú entre sí o con otros elementos de la aplicación interactiva (por ejemplo nodos de contenido) a través de conectores causales [7].

Este módulo comprende una única pestaña que permite ingresar el `id` del enlace a crearse, escoger un conector causal y la condición para que se ejecuten las acciones de conector.

En la Figura 2.45 se presenta el módulo de inserción de enlaces NCL.

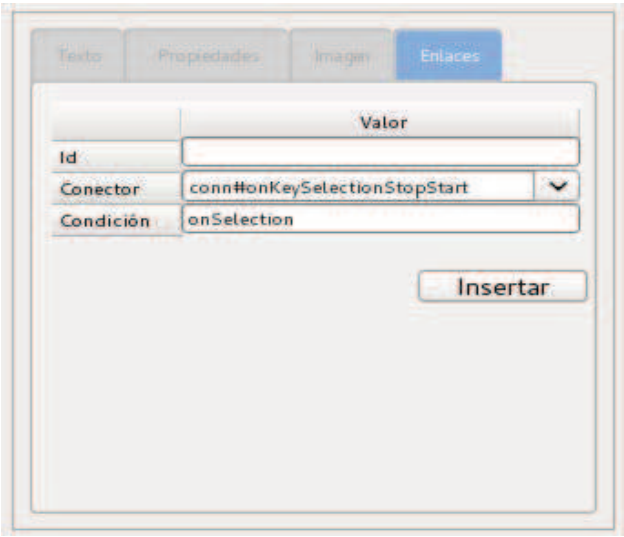


Figura 2.45 Módulo de inserción de enlaces NCL

El botón (elemento del menú), para el cual se va a insertar un enlace, debe ser previamente seleccionado desde el navegador interactivo de menús. En ocasiones, el usuario puede requerir la inserción de un enlace para una viñeta de navegación, para lo cual puede seleccionar la viñeta haciendo un clic derecho sobre ella y escogiendo la opción “Seleccionar”. La viñeta se marcará de color azul, como se puede apreciar en la Figura 2.46.

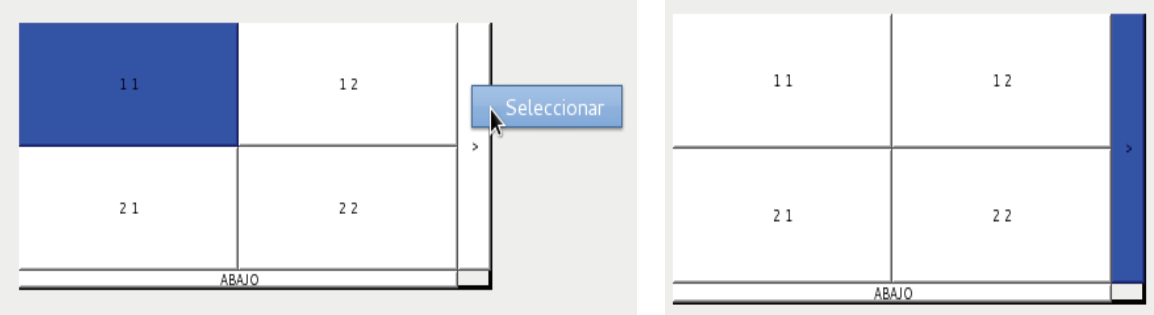


Figura 2.46 Selección de una viñeta en un menú que posee varias vistas

El participante asociado a la condición del enlace es siempre el foco del elemento. Cuando se crea un enlace en el proyecto Composer con el *plug-in* Menu Creator, únicamente se añade al árbol jerárquico de entidades la estructura general del enlace, permitiendo al usuario en base al uso de otros *plug-ins* de NCL Composer, ingresar más información como los roles de cada participante para iniciar (`start`), detener (`stop`), pausar (`pause`) y realizar otras acciones sobre ellos. La propiedad que permite al usuario encontrar el enlace para seguir editándolo en los otros *plug-ins* es el `id` del enlace que se especifica antes de insertarlo.

El Código 2.6 presenta el código de un enlace NCL insertado por el *plug-in* para el primer elemento de un menú. Nótese que el participante que debe cumplir la condición del conector tiene el identificador (`id`) del foco del elemento del menú.

```
<link id="enlaceAlPresionarEnterElementoMenu" xconnector="conn#onKeySelectionStopStart">
    <bind role="onSelection" component="m1f1c1k1m"/>
</link>
```

Código 2.6 Enlace NCL insertado por el *plug-in* Menu Creator

Desde este enlace se puede iniciar (`start`) o detener (`stop`) cualquier elemento de una aplicación interactiva, como por ejemplo un nodo de contexto o un nodo de contenido que se encuentre dentro del proyecto. El Código 2.7 presenta el código del enlace con los roles de cada participante añadidos; para ello, se ha hecho uso del *plug-in* NCL Textual View.

```
<link id="enlaceAlPresionarEnterElementoMenu" xconnector="conn#onKeySelectionStopStart">
      <bind role="onSelection" component="m1f1c1k1m">
            <bindParam name="keyCode" value="ENTER"/>
      </bind>
      <bind role="stop" component="media1"/>
      <bind role="start" component="media2"/>
</link>
```

Código 2.7 Enlace NCL con los roles de cada participante añadidos

2.5.6 MECANISMO DE ALMACENAMIENTO DE DATOS EN EL ARCHIVO .CPR

Cuando uno de los *plug-ins* desea guardar los valores de los atributos de los objetos empleados dentro del archivo .cpr del proyecto Composer, se requiere de la emisión de la señal `setPluginData` que tiene como argumento un array de *bytes* (`QByteArray data`[41]).

El composer-core es capaz de guardar y abrir los datos, pero el uso al que se destinen dichos datos depende de cada uno de los *plug-ins*, por lo que es responsabilidad del programador implementar los métodos necesarios para gestionar tanto el proceso de guardado (método `saveSubsession()`) así como el de apertura de la información (método `init()`).

Para que el array de *bytes* contenga la información de los atributos de los objetos empleados por el *plug-in*, estos objetos deben ser sometidos previamente a un proceso de serialización XML. El formato de serialización XML es independiente en cada uno de los *plug-ins*, por lo que el método de apertura debe ser capaz de leer la información XML y a partir de ella obtener la información de los atributos para crear nuevamente los objetos empleados por el *plug-in*.

[41] Ver Tabla 2.7.

La Figura 2.47 muestra estos procesos. El *plug-in* realiza una serialización XML de los atributos de los objetos para que el composer-core los guarde dentro del archivo .cpr. De igual manera, el archivo .cpr es leído por el composer-core quien recupera los datos XML para el *plug-in,* el cual es responsable de reconstruir los objetos nuevamente.

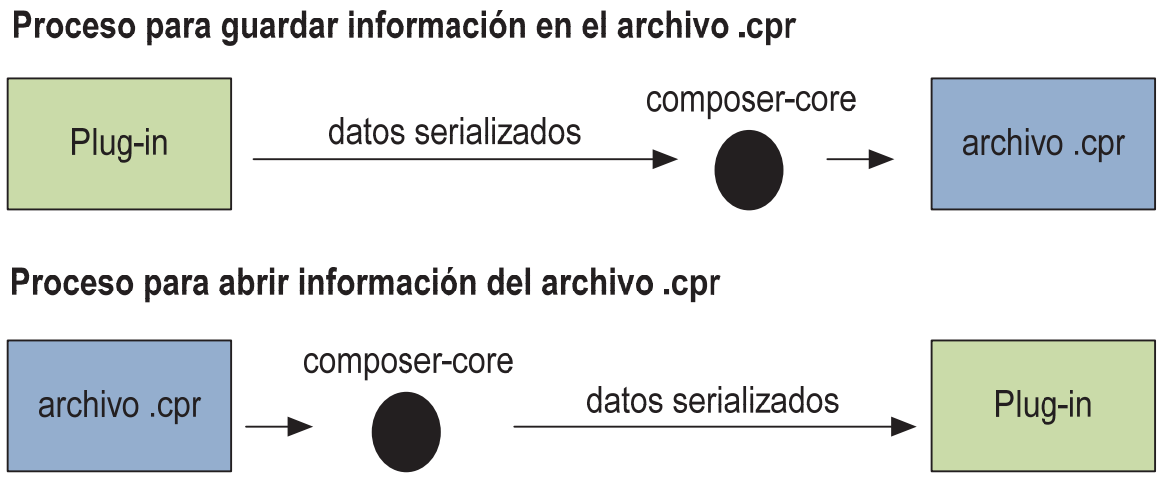


Figura 2.47 Proceso para guardar y abrir información del archivo .cpr

Para el caso del *plug-in* Menu Creator no es necesario serializar todos los objetos de la estructura jerárquica y se opta por serializar únicamente la información necesaria a partir de la cual se pueden reconstruir los menús diseñados por el usuario. Las instancias de las clases `Estructura`, `Menu` y `Texto` son las únicas clases necesarias para este propósito, por lo que el resto de instancias se reconstruyen en base a la invocación de los métodos de construcción de menús de las clases de la estructura jerárquica en tiempo de ejecución.

Es por ello que en las clases `Estructura`, `Menu` y `Texto` se ha implementado el método `serializar()`, que haciendo uso de los atributos de cada clase y de la capacidad de Qt para manejar XML, realiza el proceso de serialización de las instancias de cada una de estas clases.

La Figura 2.48 muestra un ejemplo en el que se ha creado, haciendo uso del *plug-in* Menu Creator, un menú de dos filas y dos columnas que se muestra en una sola vista.

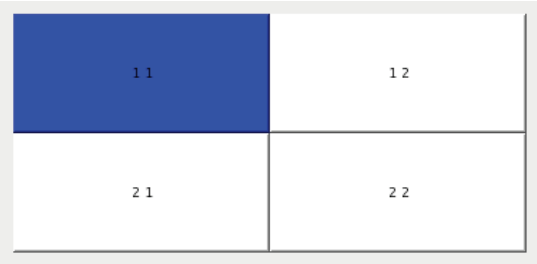


Figura 2.48 Menú de dos filas y dos columnas creado con el *plug-in* Menu Creator

En el Código 2.8 se presenta la serialización XML de las instancias de las clases `Estructura`, `Menu` y `Texto` necesarias para este menú. Para conseguirlo, se ha hecho uso de los métodos `serializar()` que contienen cada una de estas clases.

```
<Estructura numeroIdMenu="1">
   <Menu conector="conn#onKeySelectionStopStart" visible="1" nombre="ctxMenu" id="m1">
      <propiedadesMenu filas="2" alto="60" posX="20" vFilas="2" posY="20" ancho="60"
                       vColumnas="2" columnas="2"/>
      <altoFilas numero="2">
         <listaAlto valor="50"/>
         <listaAlto valor="50"/>
      </altoFilas>
      <anchoColumnas numero="2">
         <listaAncho valor="50"/>
         <listaAncho valor="50"/>
      </anchoColumnas>
      <rutas plantilla="Imagenes/Plantillas/Black_Chocolate/" texto="Imagenes/Texto/"/>
      <Texto contenido="Texto 1 1" idBoton="m1f1c1">
         <propiedadesTexto altoLienzo="200" fuente="Abyssinica SIL" anchoLienzo="600"
                           color="#00ff00" tamanio="50"/>
      </Texto>
      <Texto contenido="Texto 2 2" idBoton="m1f1c2">
         <propiedadesTexto altoLienzo="200" fuente="Abyssinica SIL" anchoLienzo="600"
                           color="#00ff00" tamanio="50"/>
      </Texto>
      <Texto contenido="Texto 2 1" idBoton="m1f2c1">
         <propiedadesTexto altoLienzo="200" fuente="Abyssinica SIL" anchoLienzo="600"
                           color="#00ff00" tamanio="50"/>
      </Texto>
      <Texto contenido="Texto 2 2" idBoton="m1f2c2">
         <propiedadesTexto altoLienzo="200" fuente="Abyssinica SIL" anchoLienzo="600"
                           color="#00ff00" tamanio="50"/>
      </Texto>
   </Menu>
</Estructura>
```

Código 2.8 Serialización XML de la información de un menú de dos filas y dos columnas

La etiqueta principal es `<Estructura>` y determina el valor del atributo `numeroIdMenu`, a partir del cual se construye el identificador alfanumérico único

de cada uno de los menús. A continuación se anidan cada uno de los menús con la etiqueta `<Menu>` y determina: el tipo de conector causal empleado para la navegación entre las distintas vistas; el valor del atributo `visible`, heredado de la clase `QFrame`, que tiene un valor de 1 si el menú es visible o de 0 si no es visible para el usuario; el valor del atributo `nombre` para el `id` del nodo de contexto que representa al menú; y, el valor del identificador alfanumérico único del menú.

Dentro de la etiqueta `<Menu>` se serializan, como nuevas etiquetas, los atributos `propiedadesMenu`, `altoFilas`, `anchoColumnas` y `rutas` de la clase `Menu`[42].

Finalmente, también dentro de la etiqueta `<Menu>` se anida el texto correspondiente a cada elemento del menú mediante la etiqueta `<Texto>`. Dentro de cada una de estas etiquetas se especifica el contenido del texto del elemento y el valor del atributo `idBoton` de la instancia de la clase `Texto`[43].

Además, dentro de cada etiqueta `<Texto>` se serializa, como una nueva etiqueta, la colección `QMap` contenida en el atributo `propiedadesTexto` de la instancia de la clase `Texto`.

2.6 CARACTERÍSTICAS Y FUNCIONALIDADES

El *plug-in* de generación automática de menús para el IDE NCL Composer facilita la creación de aplicaciones interactivas basadas en Ginga-NCL, permitiendo que un usuario genere contenido interactivo sin tener conocimientos avanzados del lenguaje NCL. Los menús que el *plug-in* es capaz de crear son de un alto nivel de interactividad pues genera los menús con elementos media y varias características y funcionalidades que dan lugar a aplicaciones llamativas para el televidente, a diferencia de un menú típico creado manualmente con Ginga-NCL en el que el televidente, usualmente, hace uso únicamente de los botones ROJO, VERDE, AMARILLO y AZUL del control remoto para seleccionar las opciones que el menú ofrece.

[42] Ver sección 2.5.2.2.
[43] Ver sección 2.5.2.7.

Entre estas características y funcionalidades se tiene:

- Uso de estilos gráficos.
- Opciones de diseño de menús.
- División de un menú en vistas y uso de viñetas de navegación.
- Menús con filas y columnas de distinto tamaño.
- Ingreso de texto.
- Navegación entre elementos y vistas de un menú.
- Inserción de enlaces NCL.

Para detallar estas características y funcionalidades, se ha diseñado un menú, como ejemplo, de cuatro filas y una columna, como se aprecia en la Figura 2.49. Este menú se presenta en dos vistas conformadas por dos filas y una columna cada una y presenta información sobre la fauna de cuatro islas del Archipiélago de Galápagos. Su diseño se irá modificando para describir las características y funcionalidades del *plug-in* Menu Creator.

Figura 2.49 Menú de las Islas Galápagos

2.6.1 USO DE ESTILOS GRÁFICOS

El *plug-in* permite al usuario diseñar menús con varios estilos gráficos. Para ello, se hace uso de plantillas que el *plug-in* ofrece al diseñador o que pueden ser creadas de forma personalizada. Para crear un nuevo menú, el usuario debe especificar una carpeta de plantillas en donde se almacenarán las imágenes PNG, como sugiere la Figura 2.50; y en base a estas imágenes, el *plug-in* creará el menú diseñado. Cada menú creado por el usuario puede usar un estilo gráfico diferente.

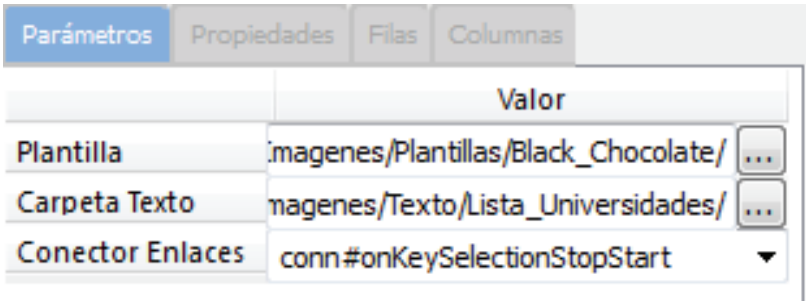


Figura 2.50 Parámetros necesarios para creación de menús

Para el *plug-in* Menu Creator se han creado cuatro plantillas de estilos gráficos: *White_Snow*, *Green_Nature*, *Purple_World* y *Black_Chocolate*.

En las Figuras 2.51, 2.52, 2.53 y 2.54 se presentan imágenes de la aplicación interactiva con un menú sobre las Islas Galápagos, el cual fue diseñado con las distintas plantillas del *plug-in* Menu Creator.

Figura 2.51 Menú obtenido con la plantilla *White_Snow*

Figura 2.52 Menú obtenido con la plantilla *Green_Nature*

Figura 2.53 Menú obtenido con la plantilla *Purple_World*

Figura 2.54 Menú obtenido con la plantilla *Black_Chocolate*

El usuario puede diseñar su propia plantilla creando diez imágenes en formato PNG que deben tener un nombre específico como se detalla en la Tabla 2.20.

Tabla 2.20 Nombres y funciones de las imágenes que conforman una plantilla

Nombre de la imagen	Función
noseleccionado	Imagen de fondo de un elemento de menú.
noseleccionadoab	Imagen de fondo de la viñeta inferior.
noseleccionadoarr	Imagen de fondo de la viñeta superior.
noseleccionadoder	Imagen de fondo de la viñeta derecha.
noseleccionadoiz	Imagen de fondo de la viñeta izquierda.
seleccionado	Imagen de foco de un elemento de menú.
selccionadoab	Imagen de foco de la viñeta inferior.
seleccionadoarr	Imagen de foco de la viñeta superior.
seleccionadoder	Imagen de foco de la viñeta derecha.
seleccionadoiz	Imagen de foco de la viñeta izquierda.

La Figura 2.55 muestra las imágenes que fueron creadas para la plantilla *Black_Chocolate*.

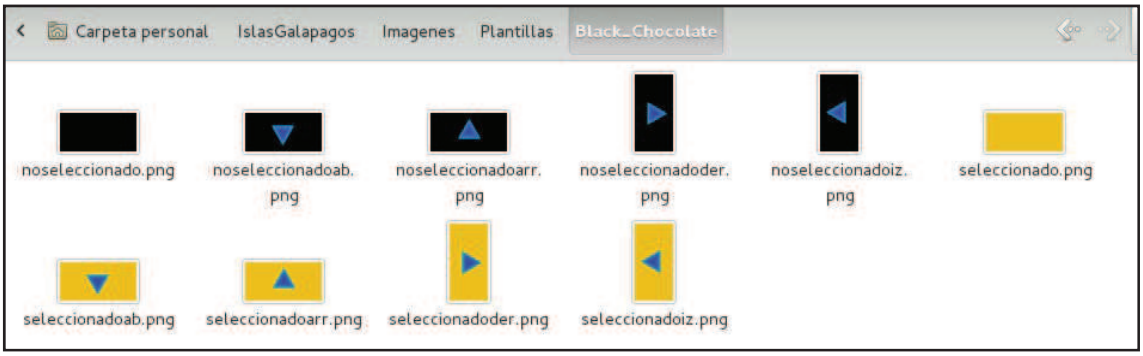


Figura 2.55 Imágenes de la plantilla *Black_Chocolate*

2.6.2 OPCIONES DE DISEÑO DE MENÚS

Para diseñar un menú, el usuario debe especificar un nombre que lo identifique, el número de filas y columnas, las dimensiones del menú (ancho y alto), como un porcentaje de la pantalla completa, el número de filas y columnas de una vista del menú y los porcentajes de las coordenadas horizontal y vertical de ubicación en la pantalla. Además, cada uno de los menús diseñados se puede mover con el mouse en el navegador interactivo de menús para especificar su ubicación en la pantalla del televisor.

En la Figura 2.56 se presenta el menú de las Islas Galápagos obtenido al ingresar los datos del menú en el *plug-in*, pero se ha especificado que se presenten todos los elementos en una sola vista.

Figura 2.56 Menú de cuatro elementos presentado en una sola vista

El *plug-in* permite al usuario cambiar las dimensiones y la posición del menú de manera sencilla, manipulándolo y arrastrándolo mediante el navegador interactivo de menús. En la Figura 2.57 se muestra el menú creado aumentado de tamaño y en una posición distinta en la pantalla. La Figura 2.58 presenta el resultado obtenido.

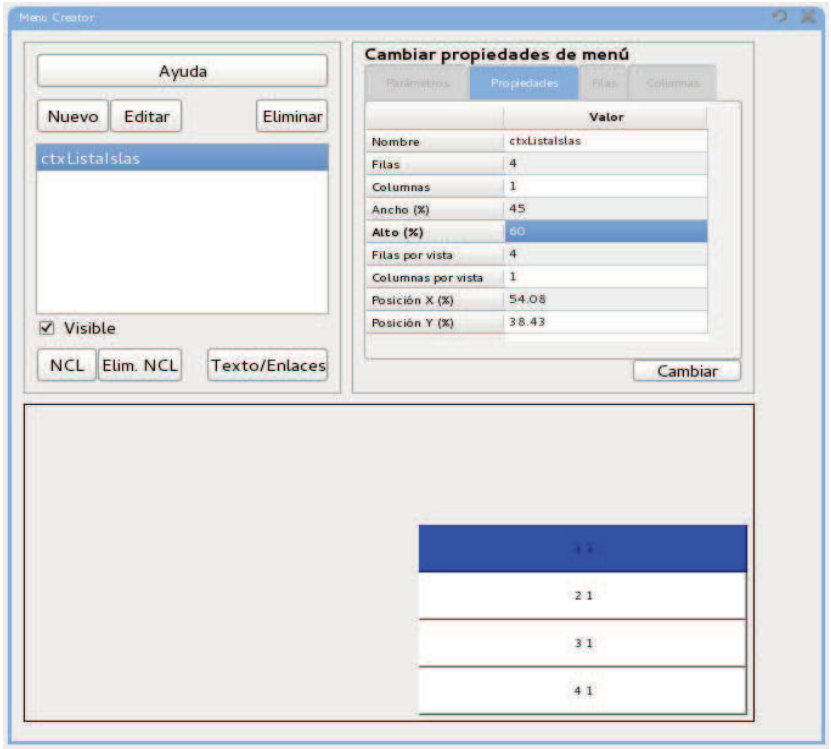


Figura 2.57 Menú aumentado de tamaño y en una posición distinta

Figura 2.58 Resultado obtenido al cambiar el tamaño y ubicación del menú

2.6.3 DIVISIÓN DE UN MENÚ EN VISTAS Y USO DE VIÑETAS DE NAVEGACIÓN

El *plug-in* permite dividir un menú creado en varias vistas, generando de forma automática las viñetas de navegación necesarias para moverse entre las distintas vistas.

La Figura 2.59 muestra una variante en el diseño del menú de las Islas Galápagos, en el que se ha dispuesto de ocho islas distribuidas en un menú de cuatro filas y dos columnas que se muestran en vistas conformadas por dos filas y una columna cada una, y que además cuenta con las viñetas de navegación generadas automáticamente y necesarias para moverse entre las distintas vistas.

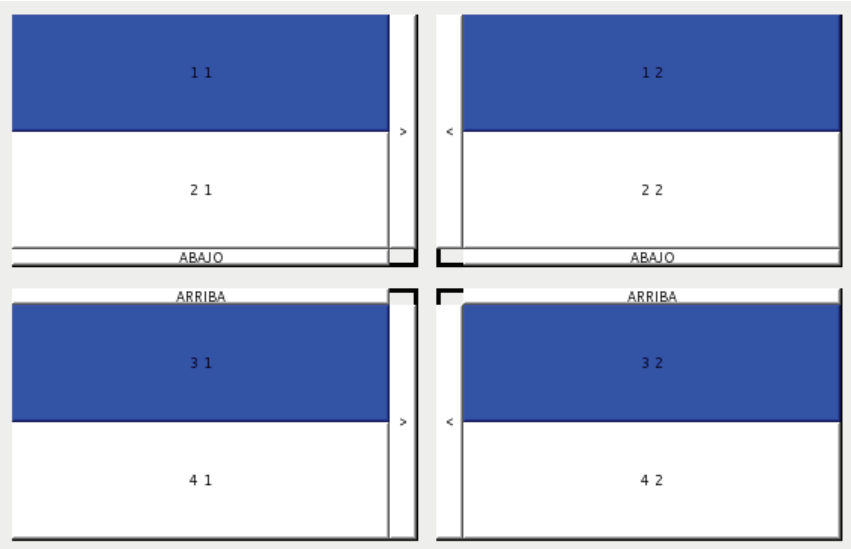


Figura 2.59 Menú de cuatro filas y dos columnas dividido en vistas de dos filas y una columna

La Figura 2.60 presenta las cuatro vistas obtenidas como resultado de la división del menú que consta de cuatro filas y dos columnas.

Figura 2.60 Vistas obtenidas al dividir el menú que consta de cuatro filas y dos columnas

El navegador interactivo de menús del *plug-in* le permite al usuario navegar entre las vistas generadas, ya sea dando clic sobre los botones del menú o desplazándose por el editor monolínea. Esto posibilita ver la disposición con la que el menú será generado, como se muestra en la Figura 2.61.

(a) (b)

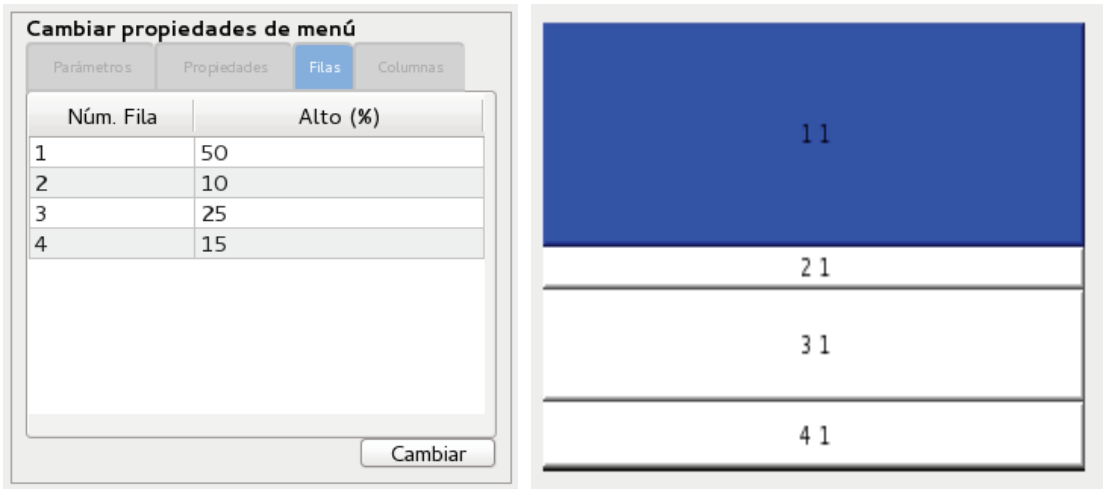
Figura 2.61 Navegación entre las distintas vistas de un menú: (a) Segunda vista (b) Tercera vista

2.6.4 MENÚS CON FILAS Y COLUMNAS DE DISTINTO TAMAÑO

Por defecto, el *plug-in* Menu Creator genera los menús diseñados con filas y columnas del mismo tamaño equitativamente distribuidas en el espacio disponible. Sin embargo, también permite crear menús conformados por filas y columnas de distintas dimensiones. Para ello, el usuario puede especificar los porcentajes de ocupación de cada fila y columna.

Así por ejemplo, si se desea diseñar un menú de cuatro filas y una columna que se presente en una única vista con filas de distinto tamaño, el usuario deberá ingresar el alto de cada fila especificado como un porcentaje del alto total. La Figura 2.62 detalla los porcentajes ingresados para cada una de las cuatro filas y

el menú diseñado que se muestra en el navegador interactivo de menús al usuario. La Figura 2.63 presenta el menú obtenido en una aplicación interactiva.



(a) (b)

Figura 2.62 Creación de un menú con filas de tamaño variable: (a) Procentajes de alto de las filas (b) Menú diseñado

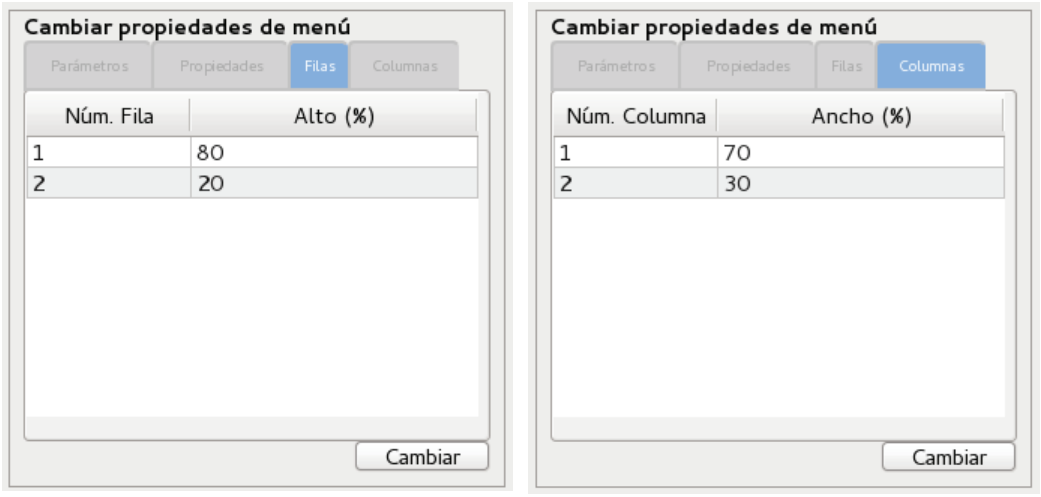
Figura 2.63 Menú con filas de distinto tamaño

El usuario puede establecer los porcentajes de las dimensiones de las filas y de las columnas según requiera.

En el caso de que el menú haya sido dividido en varias vistas, para cada una de las vistas, el usuario tiene la posibilidad de establecer las dimensiones de las filas y de las columnas que la conforman en porcentaje relativo al tamaño de la vista

que se encuentra editando. Si las filas o columnas editadas intervienen en otras vistas, estas se actualizarán automáticamente dando consistencia al menú.

Para un menú de cuatro filas y dos columnas que se presenta en vistas de dos filas y dos columnas se tendrían dos vistas, cuyas dimensiones de filas y columnas se pueden cambiar. La Figura 2.64 presenta el ingreso de los porcentajes de las dimensiones de las filas y columnas para la primera vista. En la Figura 2.65 se puede observar la primera vista diseñada que se obtiene.



(a) (b)

Figura 2.64 Dimensiones de filas y columnas de la primera vista: (a) Porcentajes de alto de cada fila (b) Porcentajes de ancho de cada columna

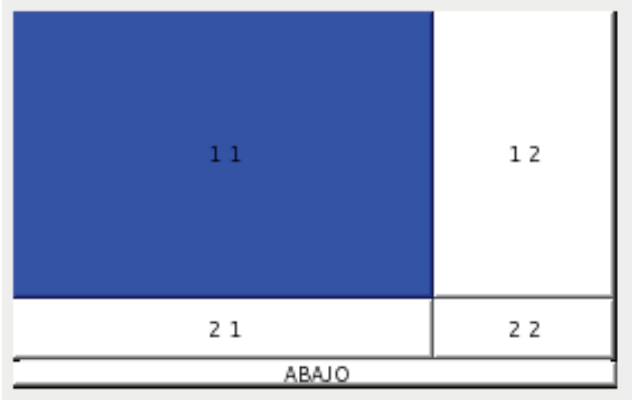
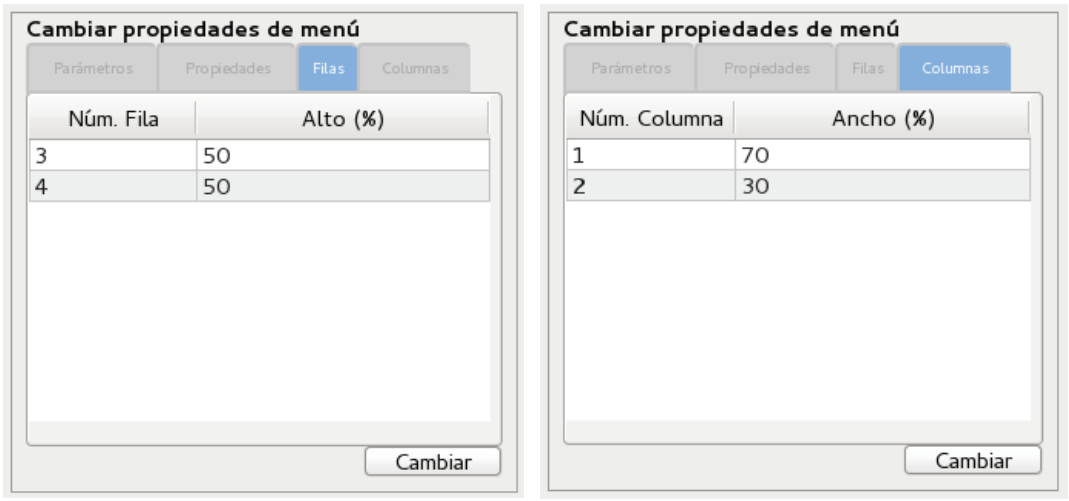


Figura 2.65 Primera vista diseñada

De igual manera se puede realizar para la segunda vista. La Figura 2.66 presenta los porcentajes de las dimensiones de las filas y de las columnas de la segunda vista. En la Figura 2.67 se presenta la vista diseñada que se obtiene. Los valores

del ancho de las columnas son iguales a los de la primera vista, ya que ambas columnas intervienen en las dos vistas.



(a) (b)

Figura 2.66 Dimensiones de filas y columnas de la segunda vista: (a) Porcentajes de alto de cada fila (b) Porcentajes de ancho de cada columna

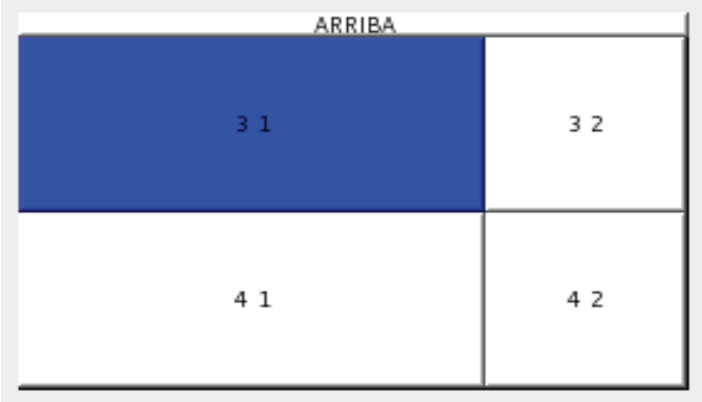


Figura 2.67 Segunda vista diseñada

2.6.5 INGRESO DE TEXTO

El módulo de ingreso de texto del *plug-in* permite introducir la información asociada a cada elemento de los menús diseñados. Cuando se crea un nuevo menú, se debe especificar una carpeta en donde se almacenarán las imágenes en formato PNG que el *plug-in* generará para cada elemento del menú con el texto ingresado por el usuario.

El módulo de ingreso de texto ofrece al usuario varias propiedades como distintas fuentes y tamaño de letra, una paleta de colores y además permite la variación de las dimensiones del lienzo (tamaño de la imagen PNG con el texto), como se presenta en la Figura 2.68. Este módulo también ofrece al usuario una vista previa del texto generado con las propiedades seleccionadas.

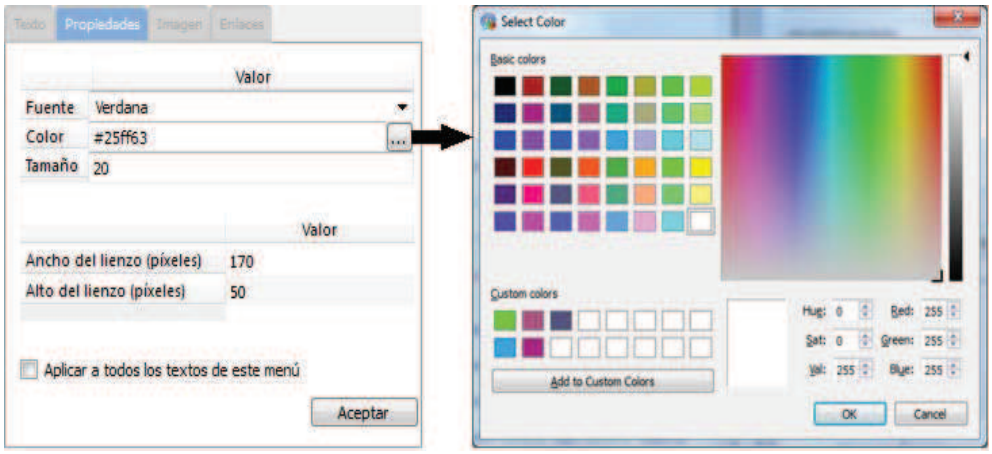


Figura 2.68 Propiedades de texto

El texto de cada elemento del menú se emplea para generar una imagen PNG y se guarda en la carpeta seleccionada por el usuario empleando como nombre de archivo un identificador alfanumérico compuesto de la letra m (menú), seguido por el número de menú (m#), luego la letra f (fila), seguido por el número de fila del elemento de menú al que corresponde (f#), luego la letra c (columna), seguido por el número de columna del elemento de menú al que corresponde (c#). De esta manera, cada imagen se guarda con un nombre en formato m#f#c#. En la Figura 2.69 se muestran las imágenes que contienen texto generadas para el menú de cuatro filas y una columna de las Islas Galápagos.

Figura 2.69 Imágenes que contienen texto

El usuario puede determinar el tamaño, color y fuente de letra empleada para generar el texto de las imágenes de todos los elementos del menú. La Figura 2.70 presenta un ejemplo.

Figura 2.70 Menús generados con iguales propiedades de texto para cada elemento

Adicionalmente, el usuario puede cambiar el color, tamaño y fuente de letra para cada uno de los elementos de un menú, es decir, determinar las propiedades del texto de manera individual para cada elemento. La Figura 2.71 presenta un ejemplo.

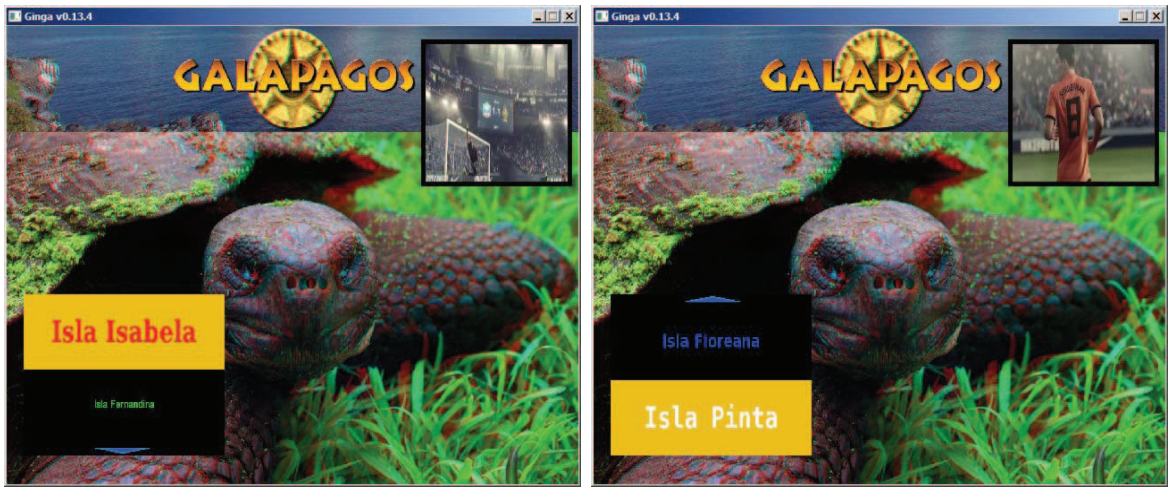


Figura 2.71 Menú generado con distintas propiedades de texto para cada elemento

El *plug-in* cuenta con un editor multilínea, el cual permite ingresar el texto de los elementos de un menú en varias líneas en lugar de una sola. La Figura 2.72

muestra el editor multilínea, en el que se ha ingresado el texto del primer elemento del menú de las Islas Galápagos en dos líneas en lugar de una sola.

Figura 2.72 Editor multilínea

En la Figura 2.73 se presenta la primera vista del menú de las Islas Galápagos con el texto de su primer elemento escrito en dos líneas. El *plug-in* centra el texto de manera adecuada automáticamente.

Figura 2.73 Menú obtenido empleando el editor multilínea del *plug-in*

2.6.6 NAVEGACIÓN ENTRE ELEMENTOS Y VISTAS DE UN MENÚ

Al diseñar un menú, el *plug-in* genera de forma automática la lógica necesaria para que el televidente pueda navegar entre los distintos elementos y entre las distintas vistas, en caso de que el menú haya sido dividido. El foco se irá desplazando a través de los elementos del menú, según las indicaciones del televidente.

En la Figura 2.74 se presenta un ejemplo de esta funcionalidad. El foco se mueve entre los distintos elementos del menú de acuerdo a las indicaciones del televidente a través del control remoto.

Cuando se presenta al televidente la primera vista del menú, el foco del primer elemento se activa. El televidente puede navegar entre los distintos elementos mediante los botones de los cursores hacia arriba, abajo, izquierda y derecha. Al presionar el botón del cursor hacia abajo, el foco del tercer elemento se activa; y al presionar el botón del cursor hacia la derecha se activa el foco del cuarto elemento.

Cuando el televidente presione el botón con el cursor hacia abajo, el foco de la viñeta de navegación se activa para sugerir al televidente que la seleccione con el botón OK. Cuando el televidente presiona el botón OK, la primera vista desaparecerá por completo para presentarse la segunda, con el foco del primer elemento activado.

En esta vista, el televidente puede desplazarse entre los distintos elementos de igual manera que en la primera vista. Así por ejemplo, si presiona el botón con el cursor hacia abajo, el foco del tercer elemento se activará.

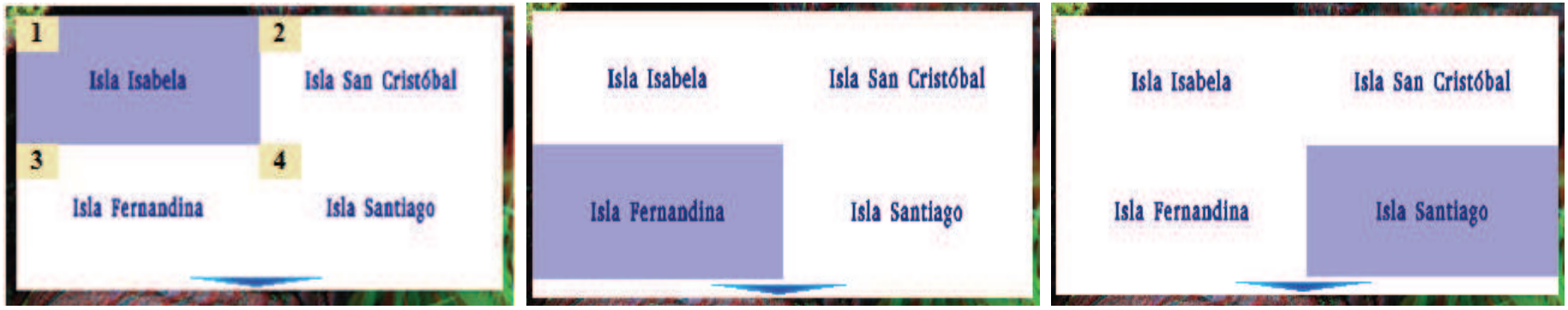


Figura 2.74 Navegación entre los elementos de un menú y vistas. Posición del foco: (a) Primer elemento de la primera vista (b) Tercer elemento de la primera vista (c) Cuarto elemento de la primera vista (d) Viñeta de navegación de la primera vista (e) Primer elemento de la segunda vista (f) Tercer elemento de la segunda vista

2.6.7 INSERCIÓN DE ENLACES NCL

El módulo de inserción de enlaces NCL permite al usuario asociar elementos del menú entre sí o con otros elementos de la aplicación interactiva a través de conectores causales. Esto posibilita que el usuario pueda determinar qué elemento de la aplicación iniciar, detener, pausar, entre otras acciones, ante una indicación del televidente con el control remoto.

En la Figura 2.75 se presenta la aplicación interactiva obtenida al insertar un enlace que emplea un conector causal para el primer elemento del menú de las Islas Galápagos, correspondiente a la Isla Isabela. Cuando el televidente se ubica sobre el primer elemento del menú y pulsa el botón OK, este conector inicia un nuevo nodo de contenido, a la izquierda de la pantalla, que informa sobre la fauna en la Isla Isabela.

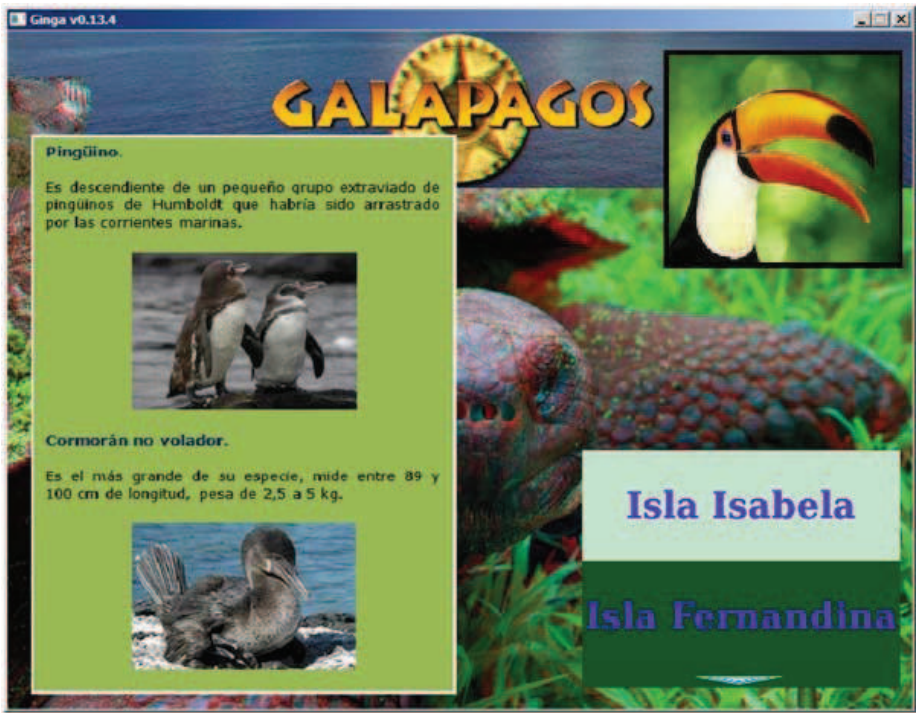


Figura 2.75 Aplicación interactiva obtenida mediante inserción de enlaces NCL

2.7 CÓDIGO NCL GENERADO POR EL PLUG-IN

Esta sección presenta el código NCL del menú de las Islas Galápagos creado con el *plug-in* Menu Creator, que se obtuvo en la Figura 2.75.

2.7.1 IDENTIFICADORES (ID) DE ENTIDADES

Para el diseño de un menú se necesitan tres imágenes por cada elemento del menú.

Para cada una de las imágenes se requiere crear tres entidades NCL (una región, un nodo de contenido y un descriptor). Por lo tanto, por cada elemento de un menú se hace uso de nueve entidades NCL (tres regiones, tres descriptores y tres nodos de contenido). También se requieren crear puertas que permitan presentar la primera vista del menú cuando se inicia el nodo de contexto. Además, se requieren crear las entidades de enlace NCL (`links`) correspondientes a los enlaces que posibilitan la navegación a través de los elementos del menú.

Uno de los principales problemas con la generación automática de entidades NCL, es asignar el valor de la propiedad `id` de la etiqueta XML de cada entidad, el cual debe ser único. Se debe establecer un mecanismo tal que garantice que no se duplicará el atributo `id` de ninguna entidad para evitar errores. Para ello, el *plug-in* Menu Creator hace uso de códigos alfanuméricos que estructuran de manera adecuada el atributo `id` de cada una de las entidades necesarias para crear uno o más menús.

En el Código 2.9 se presenta la estructura del código alfanumérico empleado para asignar el atributo `id` a cada entidad necesaria para las imágenes de fondo, de foco y de texto.

```
m#f#c#k[1,2,3][r,d,m,p]
```

Código 2.9 Estructura del código alfanumérico para el atributo `id` de las entidades necesarias para los elementos de un menú

Esta estructura consiste en una secuencia que emplea una letra seguida de un número.

Las letras empleadas son:

- **m:** Identifica a un menú. El número a continuación indica el número de menú. El primer menú creado por el usuario toma el valor de 1, el segundo 2 y así sucesivamente.
- **f:** Identifica una fila del menú. El número a continuación indica el número de fila del elemento del menú para el cual se crea la entidad. Así, si el elemento se ubica en la primera fila tomará el valor de 1, en la segunda 2 y así sucesivamente.
- **c:** Identifica el número de columna del menú. El número a continuación indica el número de columna del elemento del menú para el cual se crea la entidad. Así, si el elemento se ubica en la primera columna tomará el valor de 1, en la segunda 2 y así sucesivamente.
- **k:** Identifica si la entidad es creada para la imagen de fondo, de foco o de texto. Así, el valor de 1 indica que la entidad es para la imagen de foco, el valor de 2 indica que es para la imagen de fondo y el valor de 3 para la imagen de texto.

La estructura culmina con una letra que identifica el tipo de entidad de la que se trata:

- **r:** Indica que la entidad es una región.
- **d:** Indica que la entidad es un descriptor.
- **m:** Indica que la entidad es un nodo de contenido (medios).
- **p:** Indica que la entidad es una puerta.

De esta manera, todas las entidades creadas para presentar en la pantalla las tres imágenes que conforman un elemento de un menú tiene un `id` único que, además de representar el tipo de entidad, indica el número de menú, número de fila, número de columna e imagen (fondo, foco o texto) para la que fue creada.

En la Tabla 2.21 se resume el uso del código alfanumérico para estructurar el atributo `id` que se asigna a cada una de las entidades empleadas para mostrar en pantallas las tres imágenes que conforman un elemento de un menú.

Tabla 2.21 Códigos alfanuméricos para la propiedad `id` de las entidades necesarias para el foco, fondo y texto

Elemento NCL	Código alfanumérico único
Foco	
Puerta	m#f#c#k1p
Región	m#f#c#k1r
Descriptor	m#f#c#k1d
Nodo de contenido	m#f#c#k1m
Fondo	
Puerta	m#f#c#k2p
Región	m#f#c#k2r
Descriptor	m#f#c#k2d
Nodo de contenido	m#f#c#k2m
Texto	
Puerta	m#f#c#k3p
Región	m#f#c#k3r
Descriptor	m#f#c#k3d
Nodo de contenido	m#f#c#k3m

Las viñetas de navegación emplean un código alfanumérico un tanto distinto como lo sugiere el Código 2.10. En primer lugar, el código indica el número de vista mediante la letra v después del número de menú. Además, cuando se crean las entidades para presentar en la pantalla las dos imágenes necesarias para mostrar la viñeta de navegación, el código de numeración de la fila se mantiene constante en cero y se cambia el valor del número de la columna dependiendo de la orientación de la viñeta de navegación. La Tabla 2.22 muestra el código numérico de la columna empleado en la generación del `id` de las entidades necesarias para las imágenes de fondo y foco de las viñetas de navegación.

```
m#v#f0c#k[1,2][r,d,m,p]
```

Código 2.10 Estructura del código alfanumérico para el atributo `id` de las entidades necesarias para las viñetas de navegación

Todos los menús (m), vistas (v), columnas (c) y cuadros (k) se numerarán empezando desde uno y su valor se incrementa paulatinamente conforme el *plug-in* cree las entidades. El valor de la fila se mantiene siempre en cero.

Tabla 2.22 Código numérico de la columna según la orientación de la viñeta de navegación

Orientación de la viñeta dentro del menú	Código numérico (valor de c)
Arriba	1
Abajo	2
Izquierda	3
Derecha	4

Para los enlaces que se generan automáticamente y que posibilitan la navegación a través del menú, se hace uso de un código alfanumérico único con una leve modificación. El Código 2.11 muestra la estructura de dicho código en el que se ha remplazado la parte final, después de la especificación de la columna, por una letra l que hace referencia a enlace (link), seguida del número único de enlace que depende del botón del control remoto asociado con el enlace. La correspondencia entre dicho número y el botón del control remoto es la misma que se describió en la Tabla 2.14.

```
m#v#f#c#l#
```

Código 2.11 Estructura del código alfanumérico para el atributo `id` de una entidad enlace

2.7.2 ENTIDADES NCL

A continuación se presenta la generación de las entidades NCL del menú de las Islas Galápagos y su respectivo código.

2.7.2.1 Regiones

Las regiones se anidan dentro de la base de regiones del encabezado del documento NCL. El *plug-in* Menu Creator busca, antes de generar el código, la base de regiones del documento NCL.

El Código 2.12 presenta el código NCL de las regiones creadas automáticamente por el *plug-in* Menu Creator, necesarias para el menú de las Islas Galápagos.

```
<region3ase id="rgbase1">
    <region id="m1v1f0c2k1r" left="62.23%" top="91.98%" width="35.20%" height="2.65%" zIndex="2"/>
    <region id="m1v1f0c2k2r" left="62.23%" top="91.98%" width="35.20%" height="2.65%" zIndex="1"/>
    <region id="m1f1c1k1r" left="62.23%" top="59.43%" width="35.20%" height="16.48%" zIndex="2"/>
    <region id="m1f1c1k2r" left="62.23%" top="59.43%" width="35.20%" height="16.48%" zIndex="1"/>
    <region id="m1f1c1k3r" left="62.23%" top="59.43%" width="35.20%" height="16.48%" zIndex="3"/>
    <region id="m1f2c1k1r" left="62.23%" top="75.71%" width="35.20%" height="16.48%" zIndex="2"/>
    <region id="m1f2c1k2r" left="62.23%" top="75.71%" width="35.20%" height="16.48%" zIndex="1"/>
    <region id="m1f2c1k3r" left="62.23%" top="75.71%" width="35.20%" height="16.48%" zIndex="3"/>
    <region id="m1v2f0c1k1r" left="62.23%" top="59.43%" width="35.20%" height="2.65%" zIndex="2"/>
    <region id="m1v2f0c1k2r" left="62.23%" top="59.43%" width="35.20%" height="2.65%" zIndex="1"/>
    <region id="m1f3c1k1r" left="62.23%" top="61.88%" width="35.20%" height="16.48%" zIndex="2"/>
    <region id="m1f3c1k2r" left="62.23%" top="61.88%" width="35.20%" height="16.48%" zIndex="1"/>
    <region id="m1f3c1k3r" left="62.23%" top="61.88%" width="35.20%" height="16.48%" zIndex="3"/>
    <region id="m1f4c1k1r" left="62.23%" top="78.15%" width="35.20%" height="16.48%" zIndex="2"/>
    <region id="m1f4c1k2r" left="62.23%" top="78.15%" width="35.20%" height="16.48%" zIndex="1"/>
    <region id="m1f4c1k3r" left="62.23%" top="78.15%" width="35.20%" height="16.48%" zIndex="3"/>
</regionBase>
```

Código 2.12 Ejemplo de regiones creadas por el *plug-in* Menu Creator

2.7.2.2 Descriptores

Los descriptores se crean dentro de la base de descriptores del encabezado del documento NCL. El *plug-in* Menu Creator busca, antes de generar el código, la base de descriptores del documento NCL.

El Código 2.13 presenta los descriptores creados para el menú de las Islas Galápagos.

```
<descriptorBase id="descriptorBase1">
    <descriptor id="m1v1f0c2k1d" region="m1v1f0c2k1r"/>
    <descriptor id="m1v1f0c2k2d" region="m1v1f0c2k2r"/>
    <descriptor id="m1f1c1k1d" region="m1f1c1k1r"/>
    <descriptor id="m1f1c1k2d" region="m1f1c1k2r"/>
    <descriptor id="m1f1c1k3d" region="m1f1c1k3r"/>
    <descriptor id="m1f2c1k1d" region="m1f2c1k1r"/>
    <descriptor id="m1f2c1k2d" region="m1f2c1k2r"/>
    <descriptor id="m1f2c1k3d" region="m1f2c1k3r"/>
    <descriptor id="m1v2f0c1k1d" region="m1v2f0c1k1r"/>
    <descriptor id="m1v2f0c1k2d" region="m1v2f0c1k2r"/>
    <descriptor id="m1f3c1k1d" region="m1f3c1k1r"/>
    <descriptor id="m1f3c1k2d" region="m1f3c1k2r"/>
    <descriptor id="m1f3c1k3d" region="m1f3c1k3r"/>
    <descriptor id="m1f4c1k1d" region="m1f4c1k1r"/>
    <descriptor id="m1f4c1k2d" region="m1f4c1k2r"/>
    <descriptor id="m1f4c1k3d" region="m1f4c1k3r"/>
</descriptorBase>
```

Código 2.13 Ejemplo de descriptores creados por el *plug-in* Menu Creator

2.7.2.3 Nodo de contexto [6]

Cada menú que se crea con el *plug-in* genera automáticamente un nodo de contexto que se anida dentro del cuerpo del documento NCL (`body`). Un nodo de contexto permite una mejor organización en la estructura lógica del documento NCL [6] y además permite el reuso de los menús generados.

Dentro del nodo de contexto generado se crean los nodos de contenido, las puertas y los enlaces necesarios para la navegación entre los elementos del menú. Cuando se inicia el nodo de contexto (es decir, se realiza la acción `start` sobre el nodo), las distintas puertas se encargan de que se muestre la primera vista. En el Código 2.14 se presenta parte del código NCL del nodo de contexto obtenido para el menú de las Islas Galápagos.

```
<body id="myBodyID">
    <context id="ctxListaIslas">
        <media id="m1v1f0c2k1m" src="Imagenes/Plantillas/Green_Nature/seleccionadoab.png" descriptor="m1v1f0c2k1d"/>
        <media id="m1v1f0c2k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionadoab.png" descriptor="m1v1f0c2k2d"/>
        <port id="m1v1f0c2k2p" component="m1v1f0c2k2m"/>

        .....Parte del código NCL se ha omitido.....

    </context>
</body>
```

Código 2.14 Código NCL del nodo de contexto correspondiente al menú de las Islas Galápagos

2.7.2.4 Nodos de contenido y puertas

Los nodos de contenido y puertas se crean dentro del nodo de contexto. Para cada elemento del menú se crean tres nodos de contenido correspondientes al fondo, foco y texto; exceptuando a las viñetas de navegación para las que se crean únicamente dos nodos de contenido correspondientes al fondo y al foco.

Las puertas son creadas únicamente para la primera vista del menú de la siguiente manera:

- Una puerta para cada nodo de contenido correspondiente al fondo de los elementos y las viñetas de navegación.

- Una puerta para cada nodo de contenido correspondiente al texto de los elementos de la vista.
- Una puerta para el nodo de contenido correspondiente al foco del primer elemento de la vista.

Esto hace que cuando se inicie el nodo de contexto, se presente al televidente la primera vista del menú con el foco del primer elemento activado.

En el Código 2.15 se pueden observar los nodos de contenido y puertas creadas dentro del nodo de contexto. Para cada uno de los nodos de contenido que poseen una puerta, esta se genera bajo la definición del nodo.

```
<context id="ctxListalslas">
    <media id="m1v1f0c2k1m" src="Imagenes/Plantillas/Green_Nature/seleccionadoab.png" descriptor="m1v1f0c2k1d"/>
    <media id="m1v1f0c2k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionadoab.png" descriptor="m1v1f0c2k2d"/>
    <port id="m1v1f0c2k2p" component="m1v1f0c2k2m"/>
    <media id="m1f1c1k1m" src="Imagenes/Plantillas/Green_Nature/seleccionado.png" descriptor="m1f1c1k1d"/>
    <port id="m1f1c1k1p" component="m1f1c1k1m"/>
    <media id="m1f1c1k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionado.png" descriptor="m1f1c1k2d"/>
    <port id="m1f1c1k2p" component="m1f1c1k2m"/>
    <media id="m1f1c1k3m" src="Imagenes/Texto/Listalslas/m1f1c1.png" descriptor="m1f1c1k3d"/>
    <port id="m1f1c1k3p" component="m1f1c1k3m"/>
    <media id="m1f2c1k1m" src="Imagenes/Plantillas/Green_Nature/seleccionado.png" descriptor="m1f2c1k1d"/>
    <media id="m1f2c1k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionado.png" descriptor="m1f2c1k2d"/>
    <port id="m1f2c1k2p" component="m1f2c1k2m"/>
    <media id="m1f2c1k3m" src="Imagenes/Texto/Listalslas/m1f2c1.png" descriptor="m1f2c1k3d"/>
    <port id="m1f2c1k3p" component="m1f2c1k3m"/>
    <media id="m1v2f0c1k1m" src="Imagenes/Plantillas/Green_Nature/seleccionadoarr.png" descriptor="m1v2f0c1k1d"/>
    <media id="m1v2f0c1k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionadoarr.png" descriptor="m1v2f0c1k2d"/>
    <media id="m1f3c1k1m" src="Imagenes/Plantillas/Green_Nature/seleccionado.png" descriptor="m1f3c1k1d"/>
    <media id="m1f3c1k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionado.png" descriptor="m1f3c1k2d"/>
    <media id="m1f3c1k3m" src="Imagenes/Texto/Listalslas/m1f3c1.png" descriptor="m1f3c1k3d"/>
    <media id="m1f4c1k1m" src="Imagenes/Plantillas/Green_Nature/seleccionado.png" descriptor="m1f4c1k1d"/>
    <media id="m1f4c1k2m" src="Imagenes/Plantillas/Green_Nature/noseleccionado.png" descriptor="m1f4c1k2d"/>
    <media id="m1f4c1k3m" src="Imagenes/Texto/Listalslas/m1f4c1.png" descriptor="m1f4c1k3d"/>

    .....Parte del código NCL se ha omitido.....

</context>
```

Código 2.15 Ejemplo del código NCL de nodos de contenido y puertas

2.7.2.5 Enlaces NCL

Esencialmente, el *plug-in* crea automáticamente dos tipos de enlaces que emplean, por defecto, conectores causales de tipo `onKeySelectionStopStart` para navegar entre los elementos del menú y entre las distintas vistas.

El primer tipo de enlace permite al televidente navegar entre los elementos que conforman una vista del menú. Este enlace detiene la imagen de foco del elemento actual (ejecuta la acción `stop`) e inicia la imagen de foco (ejecuta la acción `start`) del otro elemento correspondiente al botón del control remoto presionado por el televidente (arriba, abajo, derecha o izquierda). El Código 2.16 presenta el código NCL generado automáticamente por el *plug-in* para este tipo de enlace.

```
<link id="m1v1f1c1l2" xconnector="conn#onKeySelectionStopStart">
    <bind role="onSelection" component="m1f1c1k1m">
        <bindParam name="keyCode" value="CURSOR_DOWN"/>
    </bind>
    <bind role="stop" component="m1f1c1k1m"/>
    <bind role="start" component="m1f2c1k1m"/>
</link>
```

Código 2.16 Código NCL generado por el *plug-in* para navegación entre elementos de una misma vista del menú de las Islas Galápagos

El segundo tipo de enlace permite la navegación entre las vistas y ocurre cuando se presiona el botón OK sobre una viñeta de navegación del menú. El enlace detiene todos los elementos que conforman la vista actual (ejecutando la acción `stop`) e inicia los elementos de la nueva vista (ejecutando la acción `start`). El Código 2.17 muestra el código NCL para este tipo de enlace.

```
<link id="m1v1f0c2l10" xconnector="conn#onKeySelectionStopStart">
    <bind role="onSelection" component="m1v1f0c2k1m">
        <bindParam name="keyCode" value="ENTER"/>
    </bind>
    <bind role="stop" component="m1v1f0c2k2m"/>
    <bind role="stop" component="m1f1c1k2m"/>
    <bind role="stop" component="m1f1c1k3m"/>
    <bind role="stop" component="m1f2c1k2m"/>
    <bind role="stop" component="m1f2c1k3m"/>
    <bind role="stop" component="m1v1f0c2k1m"/>
    <bind role="start" component="m1v2f0c1k2m"/>
    <bind role="start" component="m1f3c1k2m"/>
    <bind role="start" component="m1f3c1k3m"/>
    <bind role="start" component="m1f3c1k1m"/>
    <bind role="start" component="m1f4c1k2m"/>
    <bind role="start" component="m1f4c1k3m"/>
</link>
```

Código 2.17 Código NCL generado por el *plug-in* para navegación entre vistas del menú de las Islas Galápagos

REFERENCIAS

[1] Blog Spot. Funciones inline en C++. [Online]. Disponible en: http://codigomaldito.blogspot.com/2005/12/funciones-inline.html (Consultado el 22 de noviembre de 2013).

[2] E. Granizo Montalvo, *Lenguaje C Teoría y Ejercicios*, Segunda edición ed., ESPE, Ed. Quito, Ecuador:Editorial ESPE, 1999.

[3] J. Blanchette y M. Summerfield, *C++ GUI Programming with Qt 4*, Primera edición ed., Trolltech, Ed. Stoughton, USA:Prentice Hall, 2006.

[4] J. Thelin, *Foundations of Qt Development*, Primera edición ed., Trolltech, Ed. Berkeley CA, USA:Apress, 2007.

[5] Kioskeda net. La función inline en C++. [Online]. Disponible en: http://es.kioskea.net/faq/2823-la-funcion-inline-en-c (Consultado el 22 de noviembre de 2013).

[6] L. F. Gomes Soares y S. D. Junqueira Barbosa, *Programando em NCL 3.0*, Segunda edición ed., PUC-Rio, Ed. Río de Janeiro, Brasil, 2012.

[7] L. Gomes y R. Ferreira, "Nested Context Model 3.0 Part 1 - NCM Core", Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Monografía en Ciencia de Computación ISSN 0103-9741.

[8] Nokia Developer. Qt Reference Documentation Signals&Slots. [Online]. Disponible en: http://harmattan-dev.nokia.com/docs/platform-api-reference/xml/daily-docs/libqt4/signalsandslots.html (Consultado el 12 de diciembre de 2013).

[9] Nokia Developer. Qt SDK. [Online]. Disponible en: http://developer.nokia.com/Develop/Qt/Tools/ (Consultado el 20 de noviembre de 2013).

[10] P. Deitel y H. Deitel, *C/C++ Cómo programar y Java*, Cuarta edición ed., RR Donnelley, Ed. Crawfordsville, USA: Prentice Hall, 2004.

[11] Qt Project. QtSettings Class. [Online]. Disponible en: http://qt-project.org/doc/qt-5.0/qtcore/qsettings.html (Consultado el 20 de noviembre de 2013).

[12] Qt Project. Signals&Slots. [Online]. Disponible en: http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html (Consultado el 13 de diciembre de 2013).

[13] TeleMídia (PUC-Rio). AttributeReferences Class. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1util_1_1AttributeReferences.html (Consultado el 26 de noviembre de 2013).

[14] TeleMídia (PUC-Rio). Como compilar o NCL Composer a partir do código-fonte. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/pt-br/doc/tutorial/how_to_build_ncl_composer_from_source_code (Consultado el 20 de noviembre de 2013).

[15] TeleMídia (PUC-Rio). ComposerSetting.cpp. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/ComposerSettings_8cpp_source.html (Consultado el 26 de noviembre de 2013).

[16] TeleMídia (PUC-Rio). ComposerSettings utilities. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1util_1_1Utilities.html (Consultado el 26 de noviembre de 2013).

[17] TeleMídia (PUC-Rio). Entity Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1model_1_1Entity.html (Consultado el 23 de noviembre de 2013).

[18] TeleMídia (PUC-Rio). How to: build NCL Composer from source code. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/doc/tutorial/how_to_build_ncl_composer_from_source_code (Consultado el 14 de septiembre de 2013).

[19] TeleMídia (PUC-Rio). IDocumentParser Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IDocumentParser.html (Consultado el 21 de noviembre de 2013).

[20] TeleMídia (PUC-Rio). ILanguageProfile Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1ILanguageProfile.html (Consultado el 21 de noviembre de 2013).

[21] TeleMídia (PUC-Rio). IPluginFactory Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IPluginFactory.html (Consultado el 21 de noviembre de 2013).

[22] TeleMídia (PUC-Rio). IPlugin Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IPlugin.html (Consultado el 21 de noviembre de 2013).

[23] TeleMídia (PUC-Rio). LanguageControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1LanguageControl.html (Consultado el 25 de noviembre de 2013).

[24] TeleMídia (PUC-Rio). MessageControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1MessageControl.html (Consultado el 18 de noviembre de 2013).

[25] TeleMídia (PUC-Rio). MessageControl Member List. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1MessageControl-members.html (Consultado el 18 de noviembre de 2013).

[26] TeleMídia (PUC-Rio). NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/_media/screenshot/nclcomposer-0.1.5-01.png (Consultado el 15 de noviembre de 2013).

[27] TeleMídia (PUC-Rio). PluginControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1PluginControl.html (Consultado el 18 de noviembre de 2013).

[28] TeleMídia (PUC-Rio). PluginControl.cpp. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/PluginControl_8cpp_source.htmlhttp://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1PluginControl.html (Consultado el 18 de noviembre de 2013).

[29] TeleMídia (PUC-Rio). Project Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1model_1_1Project.html (Consultado el 23 de noviembre de 2013).

[30] TeleMídia (PUC-Rio). ProjectControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1ProjectControl.html (Consultado el 25 de noviembre de 2013).

[31] TeleMídia (PUC-Rio). ProjectReader Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1ProjectReader.html (Consultado el 25 de noviembre de 2013).

[32] Wikipedia. Archivos de cabecera. [Online]. Disponible en: http://es.wikipedia.org/wiki/Archivo_de_cabecera (Consultado el 13 de diciembre de 2013).

[33] Wikipedia. Biblioteca de enlace dinámico. [Online]. Disponible en: http://es.wikipedia.org/wiki/Biblioteca_de_enlace_din%C3%A1mico (Consultado el 13 de diciembre de 2013).

[34] Wikipedia. Qt (Biblioteca). [Online]. Disponible en: http://es.wikipedia.org/wiki/Qt_%28biblioteca%29 (Consultado el 13 de diciembre de 2013).

[35] Wikipedia. MinGW. [Online]. Disponible en: Wikipedia. MinGW. [Online]. Disponible en: http://es.wikipedia.org/wiki/MinGW (Consultado el 13 de diciembre de 2013).

CAPÍTULO 3

DISEÑO E IMPLEMENTACIÓN DEL SUBSISTEMA DE ADQUISICIÓN Y PROCESAMIENTO DE DATOS

En este capítulo se realiza el diseño e implementación del Subsistema de Adquisición y Procesamiento de Datos (SAPDa), el cual está conformado por:

- Un servicio web *Windows Communication Foundation* (WCF) híbrido que, principalmente, implementa un *bot* de búsqueda para obtener información de sitios web, gestiona las bases de datos para el almacenamiento y procesamiento de información y envía los datos solicitados al STB del televidente a través del canal de retorno.
- Una aplicación de escritorio para consumir el servicio web (cliente) permitiendo al usuario realizar las acciones de búsqueda, almacenamiento y procesamiento de información.
- Una aplicación de escritorio (Mixer) encargada de realizar la generación de datos NCL asociando la información almacenada en las bases de datos con los menús generados en base al *plug-in* y/o con nodos de contenido (referentes a imágenes PNG o *scripts* Lua) declarados dentro del documento NCL, para la generación del contenido interactivo final a mostrarse al televidente.
- Una base de datos por cada aplicación interactiva creada, la cual es generada automáticamente por el servicio web.

3.1 SERVICIO WEB WCF [5], [21], [25]

El servicio web WCF es el principal componente del SAPDa e implementa métodos que pueden ser invocados por aplicaciones que consuman el servicio.

El servicio web además de ser utilizado por la aplicación cliente para permitir al usuario realizar las acciones de búsqueda, almacenamiento y procesamiento de

información, también es consumido por la aplicación Mixer para obtener información de las bases de datos y por el STB del televidente, a través de un *script* Lua, para la obtención de datos a través del canal de retorno.

Dado que las aplicaciones cliente y Mixer son aplicaciones de escritorio, pueden comunicarse con el servicio web empleando SOAP y hacer uso de los métodos ofrecidos por el servicio a través del *proxy*, lo cual facilita la programación de las aplicaciones [21]. Sin embargo, para la comunicación con el STB del televidente a través del canal de retorno es conveniente emplear la arquitectura REST, para así evitar la envoltura del mensaje SOAP y poder retornar datos en texto plano en lugar de XML [25], lo cual reduce significativamente la complejidad del *script* Lua que se ejecuta en el STB y que debe solicitar datos y presentarlos adecuadamente en la pantalla del televisor.

Por lo tanto, el servicio web WCF es un servicio híbrido que emplea SOAP para ser consumido por las aplicaciones cliente y Mixer, y a la vez emplea la arquitectura REST para ser consumido por el STB del televidente, como se ilustra en la Figura 3.1.

Figura 3.1 Consumo del servicio web WCF empleando SOAP y REST

El servicio SOAP y el servicio RESTful son los encargados de emplear las bases de datos para realizar acciones según las órdenes que sus clientes les indiquen. Para poder realizar esto, el servicio WCF debe convertirse en un cliente del servicio de bases de datos brindado por SQL Server [5], como sugiere la Figura 3.2.

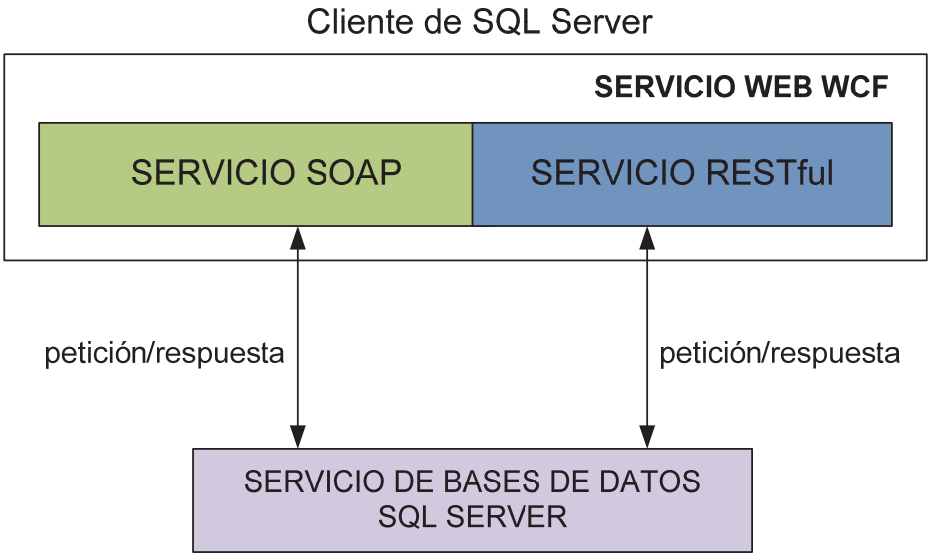


Figura 3.2 Servicio web WCF como cliente del servicio de bases de datos SQL Server

De esta manera, considerando el servicio web WCF y las bases de datos, el SAPDa brinda tres servicios: el servicio SOAP, el servicio RESTful y el servicio de bases de datos, los cuales deben ser alojados en servidores.

El alojamiento de los servicios en los servidores se puede distribuir de cualquier manera, empleando un único servidor para alojar a los tres servicios, o haciendo uso de dos o tres servidores en el que cada uno aloje a uno o más servicios. Sin embargo, cualquiera que sea la distribución de los servicios en los servidores, no es aconsejable ofrecer el servicio RESTful en un servidor distinto del de bases de datos, ya que al ser el servicio RESTful consumido por el STB del televidente, el consultar las bases de datos en un servidor remoto para la obtención de la información a enviarse a través del canal de retorno, generaría un retardo adicional que obligaría al televidente a esperar más tiempo para poder mirar el contenido solicitado en el televisor, a diferencia del tiempo que tendría que esperar teniendo en un único servidor el servicio RESTful y de bases de datos.

Por lo tanto, para el alojamiento del servicio web WCF y de bases de datos, se consideran dos opciones[44]:

[44] Estas son las opciones recomendadas pero no significa que sean las únicas. Así una tercera opción es utilizar dos servidores en el que un servidor aloje los servicios SOAP y de bases de datos y el otro servidor aloje el servicio RESTful; y una cuarta opción sería emplear tres servidores para que cada uno aloje a uno de los servicios.

- Emplear un único servidor que aloje a los tres servicios, lo cual es la solución óptima por evitar un retardo adicional al manipular las bases de datos en un servidor remoto, como se presenta en la Figura 3.3.
- Emplear dos servidores en el que uno de ellos aloje al servicio SOAP y otro al servicio RESTful y de bases de datos, como se presenta en la Figura 3.4.

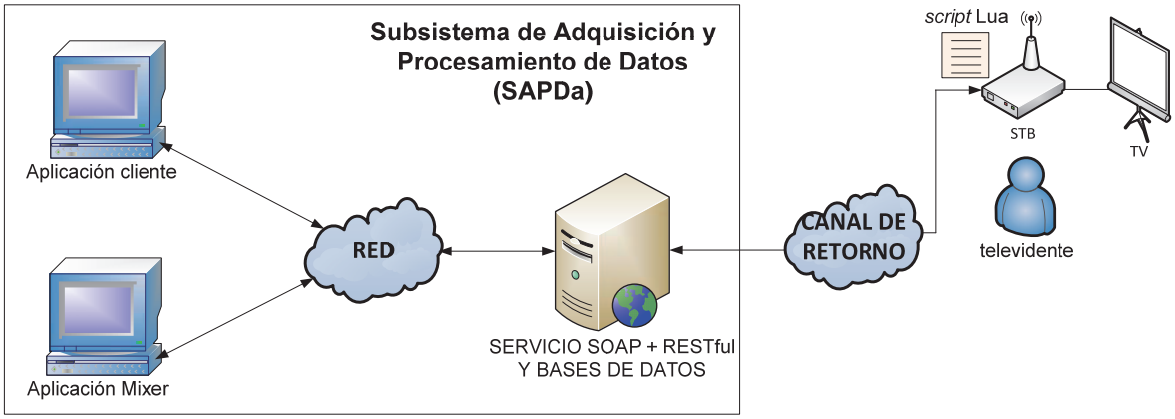


Figura 3.3 Alojamiento de servicios empleando un único servidor

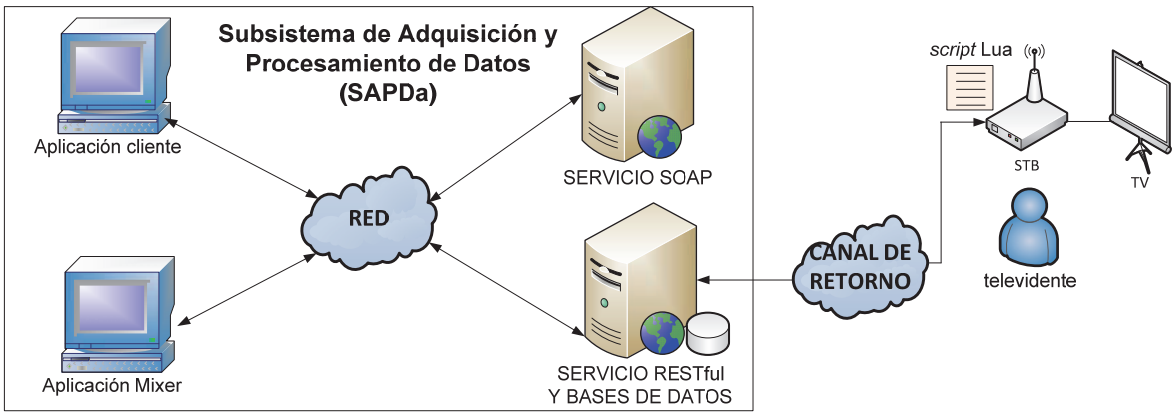


Figura 3.4 Alojamiento de servicios empleando dos servidores

3.1.1 SERVICIO SOAP

El servicio SOAP implementa y ofrece métodos encargados de realizar las siguientes tareas:

- Configuración de la dirección IP empleada para la comunicación con el servicio RESTful y de bases de datos.

- Búsqueda de información en sitios web.
- Configuración de horarios de búsqueda periódica de información.
- Almacenamiento de información de las búsquedas en sitios web realizadas por los usuarios en un archivo binario mediante serialización de objetos.
- Gestión de bases de datos para almacenamiento y procesamiento de información.
- Gestión de usuarios.

La aplicación cliente hace uso de todas las tareas ofrecidas por el servicio SOAP mientras que la aplicación Mixer únicamente hace uso de los métodos para obtener la dirección IP del servidor que aloja al servicio RESTful (servidor de retorno) y del servidor de bases de datos, así como también de algunos de los métodos de gestión de bases de datos para obtener información.

3.1.1.1 Configuración de la dirección IP empleada para la comunicación con el servicio RESTful y el servicio de bases de datos

Para que el servicio RESTful sea consumido por el STB del televidente, el *script* Lua requiere construir un URL en el que conste la dirección IP[45] del servidor de retorno. De igual manera, para que el servicio SOAP pueda manipular la información de las bases de datos, requiere conocer la dirección IP del servidor de bases de datos para establecer una conexión con SQL Server. Aunque el servicio RESTful se aloje en el mismo servidor de bases de datos, la dirección IP de ambos bien podría no ser la misma. Así por ejemplo, si el STB consume el servicio RESTful por Internet la dirección IP requerida para el *script* Lua debería ser una dirección IPv4 pública que permita la comunicación con el servidor, mientras que la empleada por el servicio SOAP para el uso de las bases de datos podría ser la dirección IP privada de ese mismo servidor en una Intranet.

El servicio SOAP implementa dos métodos de configuración de la dirección IP empleada para la comunicación con el servicio RESTful, los cuales permiten obtener y establecer, respectivamente, la dirección IP que se empleará en la generación de los *scripts* Lua. La aplicación Mixer utiliza únicamente el método

[45] En este caso la dirección IP, tanto para el servidor de retorno como para el de bases de datos, corresponde a una dirección IPv4.

para la obtención de la dirección IP con el fin de construir el URL necesario para solicitar datos al servicio RESTful a través del canal de retorno. La aplicación cliente utiliza ambos métodos para mostrar al usuario la dirección actual establecida, así como para permitirle especificar una nueva.

Por otro lado, el servicio SOAP también implementa dos métodos de configuración de la dirección IP empleada para la comunicación con el servicio de bases de datos. Ambos métodos son empleados por la aplicación cliente para presentar al usuario la dirección actual que el servicio SOAP está empleando para la conexión con SQL Server, así como para permitir que el usuario pueda especificar el valor de dicha dirección.

3.1.1.2 Búsqueda de información

Para cada celda de una tabla de las distintas bases de datos, el usuario puede realizar una consulta (búsqueda) en distintos sitios web de confianza que indique, con el objetivo de obtener la información que se almacenará en dicha celda.

Para ello, el usuario especifica de manera general un listado de los sitios web de confianza que desea emplear para realizar las distintas consultas. De este listado, el usuario puede seleccionar aquellos sitios web en los que desea buscar para obtener la información correspondiente a una determinada celda. Además, el usuario deberá ingresar por cada celda un listado de palabras clave con las cuales se realiza la búsqueda en los distintos sitios web.

Por cada sitio web seleccionado para consultar la información de una determinada celda, el servicio SOAP realiza una búsqueda en Google utilizando el listado de palabras clave ingresado por el usuario. La búsqueda que se realiza es la misma que realizaría el usuario en el buscador ingresando cada palabra clave separada por un espacio en blanco y colocando, al final del listado, el URL del sitio web.

Por cada búsqueda realizada se obtiene, a partir de los diez primeros resultados retornados por Google, un listado con los URL correspondientes a las páginas web contenidas en el sitio, las cuales podrían tener información útil para el usuario.

Finalmente, se convierte la información de cada página web encontrada en texto plano y se la separada en párrafos. Cada párrafo de una página web es puntuado mediante un algoritmo de puntuaciones, de manera que se presente al usuario como resultado de la consulta únicamente el párrafo con el puntaje más alto.

3.1.1.2.1 Búsqueda en Google y obtención de los URL de páginas web [1], [4], [9], [11], [19], [26], [27]

Con el fin de realizar una búsqueda en Google, se emplea la clase `WebClient`, disponible a través del espacio de nombres `System.Net` [19]. Un objeto de esta clase permite descargar la información de un recurso web a través del método `DownloadData()` cuyo argumento es el URL del recurso [19]. En este caso, este URL corresponde a una petición de búsqueda a Google, la cual se estructura como sugiere el Código 3.1. La palabra `búsqueda` que se indica en la estructura consiste en una cadena de texto conformada por el listado de palabras clave con las cuales se realiza la búsqueda y el URL del sitio web, separados por el símbolo `+`. Al emplear este URL, Google retornará los diez primeros resultados de la búsqueda.

```
http://www.google.com.ec/search?q=búsqueda
```

Código 3.1 Estructura del URL para búsqueda en Google

El método `DownloadData()` obtiene un *array* de *bytes*, correspondiente al resultado de búsqueda, el cual puede ser convertido en una cadena de texto empleando la clase abstracta[46] `Encoding` disponible a través del espacio de nombres `System.Text` [9].

El resultado final obtenido es una cadena de texto con el resultado de la búsqueda retornado por Google en formato HTML. Los URL de las páginas web se pueden obtener haciendo uso de la biblioteca `HTMLAgilityPack`.

[46] Una clase abstracta (*abstract class*) no puede ser instanciada, por lo que sus métodos deben ser invocados empleando directamente el nombre de la clase.

`HTMLAgilityPack` es una biblioteca de clases que permite analizar el código HTML de páginas web de manera sencilla para poder obtener la información que se desee; este proceso es también denominado *screen scraping*. La biblioteca fue desarrollada para C# y puede obtenerse de [1] para su uso. La Tabla 3.1 presenta algunas de las clases de la biblioteca, las cuales son útiles porque permiten manipular el código de páginas web sin tener que preocuparse de si el código HTML está bien formado o no, ya que encontrarse con páginas web cuyo código HTML contenga errores es bastante común en Internet.

Tabla 3.1 Algunas clases de la biblioteca `HTMLAgilityPack`

Clase	Descripción
`HtmlDocument`	Modela un documento con código HTML.
`HtmlNode`	Representa un nodo HTML, el cual puede ser hijo de otro nodo y a la vez contener nodos hijos.
`HtmlAtribute`	Representa una propiedad de la etiqueta HTML de un nodo.
`HtmlWebException`	Permite manejar excepciones producidas por código HTML mal formado.

Son muchas las funcionalidades que brinda esta biblioteca para trabajar con código HTML. Sin embargo, una de las principales características radica en que `HTMLAgilityPack` permite obtener mediante consultas `XPATH`[47] una colección de nodos HTML correspondientes a determinadas etiquetas HTML que cumplan con la condición de la consulta, así como obtener de manera sencilla el valor de determinadas propiedades de las etiquetas HTML de dichos nodos. Se pasa una ruta `XPATH` como una cadena de texto al método `SelectNodes()` de un objeto de la clase `HtmlNode`. Particularmente, el nodo padre[48] de un documento HTML se puede obtener mediante la propiedad `DocumentNode` del objeto de la clase `HtmlDocument` correspondiente al código HTML de la página web.

Dado que en una página web retornada por Google, los enlaces hacia otras páginas web están contenidos en la propiedad `href` de las etiquetas con el *tag name* `a`, mediante una consulta `XPATH`, como la del Código 3.2, se podrá obtener

[47] `XPATH` (ruta XML) evalúa una expresión de tipo `//tag_name[@propiedad]` para obtener un subconjunto del conjunto de nodos HTML que cumplan con la condición de la expresión [26].
[48] El nodo padre de un documento HTML es el nodo raíz en el cual se anidan todos los demás nodos que conforman el documento.

facilmente una colección con los nodos HTML cuyo *tag name* sea `a` y contengan la propiedad `href`. De esta manera, se logra obtener una colección con todos los URL de las páginas web correspondientes a la búsqueda en Google.

```
//a[@href]
```

Código 3.2 Ruta `XPATH` para obtener nodos HTML con la etiqueta `a` y propiedad `href`

Por defecto, Google emplea un mecanismo de codificación de URL, por lo que los URL obtenidos en la búsqueda no pueden ser empleados directamente para acceder a las páginas web [4]. Por ejemplo, si en el buscador se realiza una búsqueda con las palabras `misión http://www.epn.edu.ec` como se muestra en la Figura 3.5, el primer resultado encontrado corresponde al sitio web con el URL: `http://www.epn.edu.ec/index.php?option=com_content&view=article&id=1131&Itemid=270`, correspondiente a la página web que se muestra en la Figura 3.6.

La Figura 3.7 presenta parte del código HTML del resultado de la búsqueda retornado por Google. El URL correspondiente a la página web anteriormente descrita se encuentra resaltado en azul. Nótese que este URL se encierra entre comillas como valor de la propiedad `href` de la etiqueta `a` (que denota un enlace). Como se puede apreciar, Google no retorna el URL de la página web sino un URL codificado de la siguiente forma:

```
/url?q=http://www.epn.edu.ec/index.php%3Foption%3Dcom_content
%26view%3Darticle%26id%3D1131%26Itemid%3D270&amp;sa=U&amp;ei=
vLPNUu2AFZC7kQeMsIH4Bg&amp;ved=0CCEQFjAA&amp;usg=AFQjCNEDs64q
7u3SD54bY3w1_LHqlsB9pQ
```

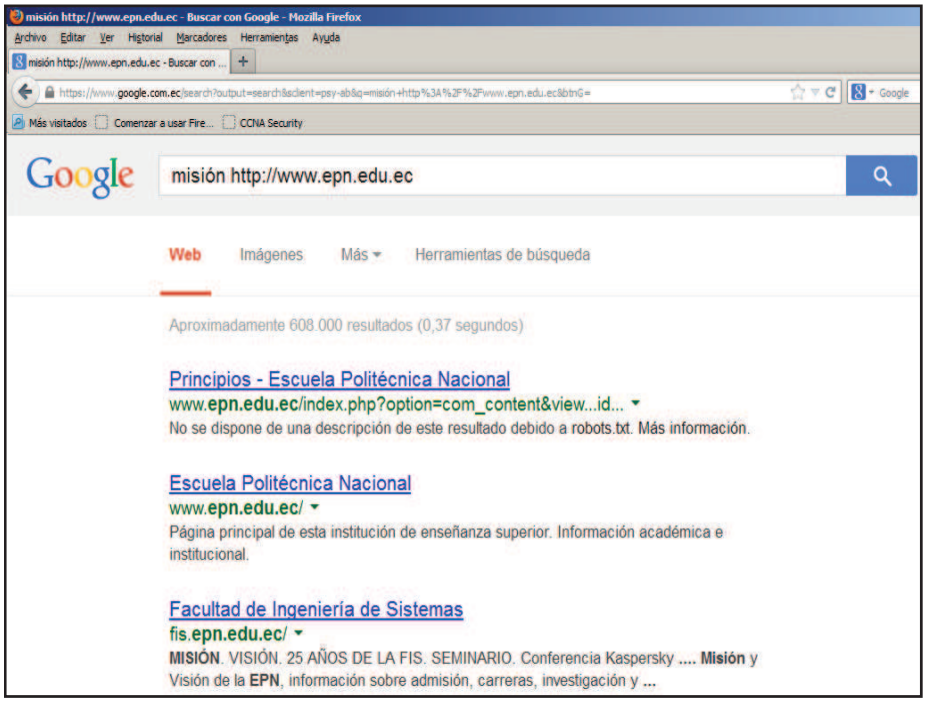


Figura 3.5 Búsqueda en Google con las palabras `misión http://www.epn.edu.ec`

Figura 3.6 Página web correspondiente al URL `http://www.epn.edu.ec/index.php?option=com_content&view=article&id=1131&Itemid=270`

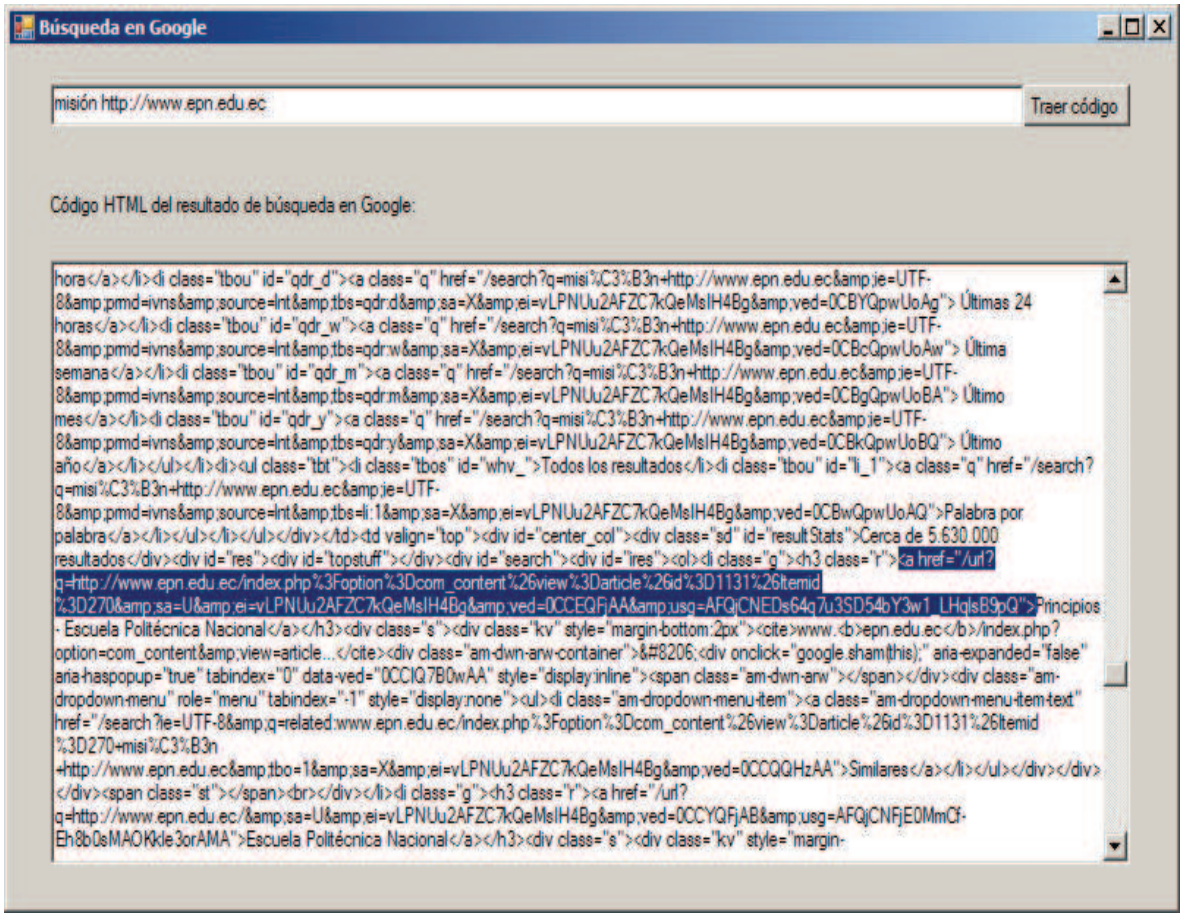


Figura 3.7 Código HTML con el URL codificado por Google de la página web `http://www.epn.edu.ec/index.php?option=com_content&view=article&id=1131&Itemid=270`

Un análisis del mecanismo de codificación de URL que Google realiza excede el alcance de este proyecto. Sin embargo, es importante mencionar que de manera general, el URL codificado está conformado por varios campos separados por el símbolo `&` y que el URL sin codificar se puede obtener a través de la manipulación del campo inicial, es decir, de la cadena de texto antes de la aparición del primer símbolo `&`. El proceso de decodificación del URL retornado por Google se detalla en la Figura 3.8. En este proceso se ha añadido un paso de selección, en el que únicamente se toman los URL correspondientes a páginas web con contenido HTML para su posterior análisis.

Si la página web está contenida dentro del sitio web, el valor de la propiedad `href` empezará con la secuencia `/url?q=` seguido del URL del sitio web. Si no es así, la página web corresponde a otro sitio web por lo que se descarta.

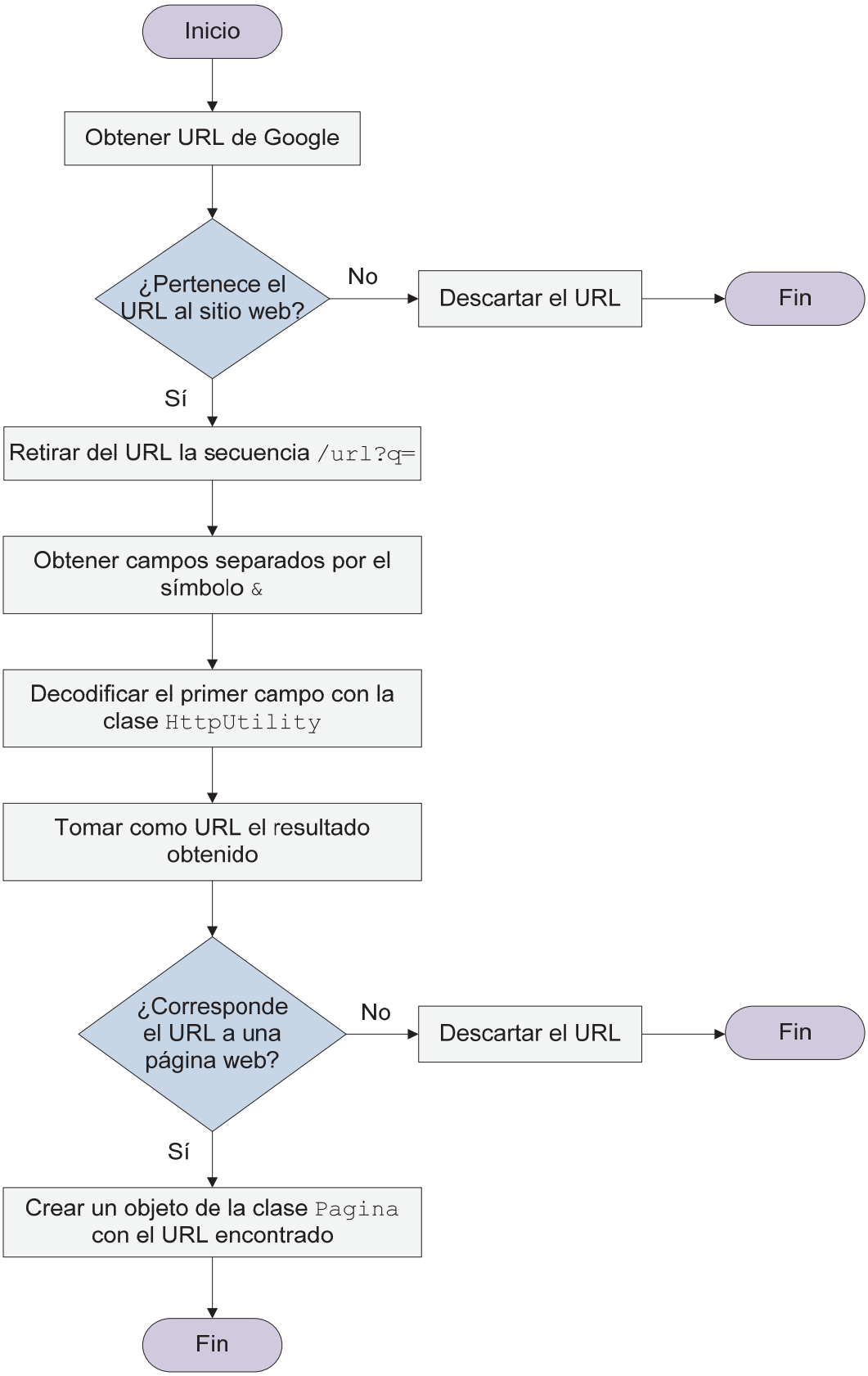


Figura 3.8 Proceso de decodificación y selección de un URL obtenido mediante una búsqueda en Google

Ciertos caracteres especiales dentro del URL retornado por Google están codificados. Por ejemplo, los espacios en blanco son reemplazados con un signo `+`, y algunos caracteres especiales son reemplazados por un signo `%` seguido por su equivalente hexadecimal en código ASCII [27]. Visual Studio ofrece la clase sellada[49] `HttpUtility` disponible en el espacio de nombres `System.Web` [11]. El método `UrlDecode()` de esta clase recibe como argumento una cadena de texto que posee caracteres especiales codificados, y retorna como resultado la cadena de texto decodificada [11].

El URL original (sin la codificación de Google) puede ser entonces recuperado si se elimina de la cadena de texto del URL codificado por Google la secuencia inicial `/url?q=` y el texto restante después del primer `&` (incluyendo al símbolo `&`); y se pasa la cadena resultante al método `UrlDecode()` de la clase `HttpUtility`.

Finalmente, como paso adicional, se analiza la extensión del recurso al cual corresponde el URL obtenido, con el fin de asegurar de que el URL corresponda a una página web con contenido HTML y no a recursos como, por ejemplo, documentos o aplicaciones en línea. Por lo tanto, si el URL culmina en extensiones tales como .indd, .pdf, .ppt, .doc, .xls, .pptx, .docx, .xlsx, .exe, entre otras, dicho URL es descartado. Si el URL es válido, entonces se crea un objeto de la clase `Pagina`[50] con el URL encontrado.

3.1.1.2.2 Obtención de la información de una página web en texto plano

Una vez que se han obtenido los URL correspondientes a las distintas páginas web en las que puede existir información de interés para el usuario, el código HTML de cada página web puede ser obtenido mediante un objeto de la clase `WebClient` y el empleo de la clase `Encoding`, de forma similar al mecanismo de obtención del resultado de búsqueda en Google.

---

[49] Una clase sellada (*sealed class*) no permite que otras clases hereden de ella resguardando todos sus atributos, métodos y propiedades.

[50] La clase `Pagina` es una de las clases de la estructura jerárquica de clases empleada por el servicio SOAP para guardar la información de las búsquedas en sitios web realizadas por los usuarios mediante serialización de objetos, y se describe en las secciones 3.1.1.4 y 3.1.1.8.6.

La información de la página web está contenida en nodos HTML por lo que, para facilitar su posterior análisis mediante el algoritmo de puntuaciones, es necesario obtener dicha información en texto plano eliminando las etiquetas HTML y tomando únicamente la información de interés, es decir, tomando como una cadena de texto únicamente la información correspondiente a la que el usuario puede leer al abrir la página web.

Esta tarea se puede realizar mediante la biblioteca `HTMLAgilityPack` y la implementación de un método recursivo[51] que recorra cada uno de los nodos HTML, para ir añadiendo la información encontrada en cada nodo a una cadena de texto inicialmente vacía. La Figura 3.9 describe el método recursivo que implementa el servicio SOAP para este propósito.

Antes de invocar al método por primera vez, se obtiene el nodo padre del documento HTML mediante la propiedad `DocumentNode` del objeto de la clase `HtmlDocument` correspondiente al código HTML de la página web. Al invocarse el método recursivo se pasa el nodo padre como argumento del método. El nodo padre, al igual que el resto de nodos del documento HTML, es manipulado mediante la creación de un objeto de la clase `HtmlNode`. La Tabla 3.2 detalla los atributos de la clase `HtmlNode` empleados en el método recursivo para manipular los nodos.

Tabla 3.2 Algunos atributos de la clase `HtmlNode` de la biblioteca `HTMLAgilityPack`

Atributo	Descripción
`ChildNodes`	Colección de objetos `HtmlNode` correspondientes a los nodos hijos del nodo.
`OriginalName`	Nombre original de la etiqueta HTML del nodo que permite reconocer el contenido de la etiqueta. Por ejemplo: • **`script:`** La etiqueta contiene código de *scripting*. • **`#comment:`** La etiqueta contiene un comentario. • **`#text:`** La etiqueta contiene información.
`InnerText`	Texto contenido dentro de la etiqueta HTML.

[51] Un método recursivo es aquel que se invoca a sí mismo.

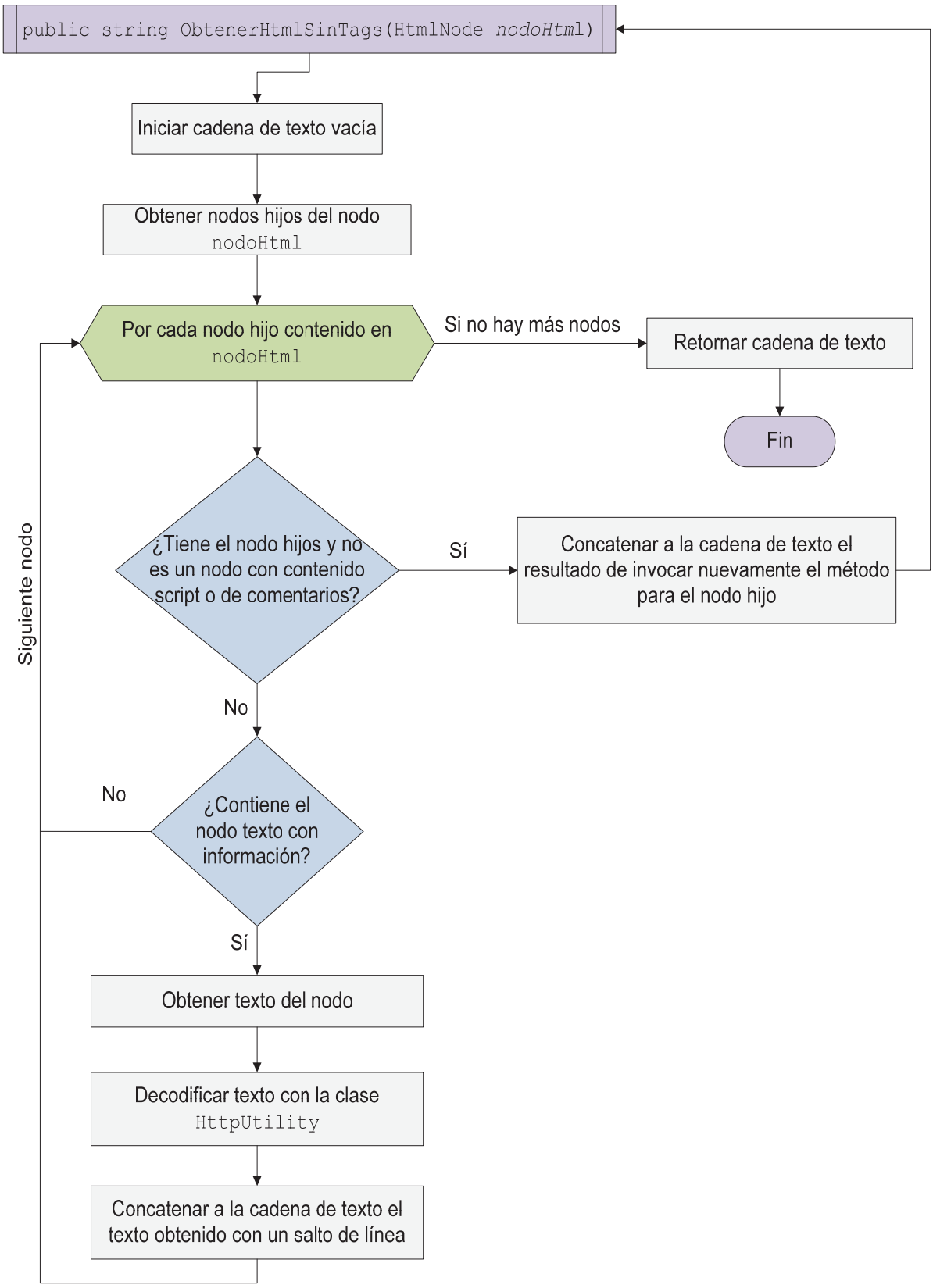


Figura 3.9 Método recursivo para obtención de la información de una página web en texto plano

El método recursivo empieza inicializando una cadena de texto vacía a la cual se irá concatenando paulatinamente la información almacenada en cada nodo. Luego, mediante el atributo `ChildNodes` del objeto de la clase `HtmlNode`, correspondiente al nodo que se pasa como argumento del método, se obtiene una colección con los nodos hijos del nodo HTML. El atributo `OriginalName` permite conocer el nombre original de la etiqueta HTML del nodo. Si el nodo no corresponde a un nodo con una etiqueta *script* o de comentario y posee nodos hijos, el método se vuelve a invocar nuevamente con el nodo como argumento.

Si el nodo no posee más nodos hijos y corresponde a una etiqueta que contiene información, se añade a la cadena de texto el texto contenido en el nodo HTML, el cual se puede obtener mediante el atributo `InnerText` del objeto de la clase `HtmlNode` correspondiente al nodo.

Al culminar el método recursivo, el resultado será la obtención de la información contenida en la página web como una cadena de texto.

3.1.1.2.3 Algoritmo de puntuaciones

El algoritmo de puntuaciones empleado para puntuar a cada línea que conforma la información de una página web se detalla en el diagrama de flujo de la Figura 3.10. Este algoritmo es efectuado para cada página web correspondiente a los URL obtenidos con la búsqueda en Google.

En primer lugar, se debe obtener la información contenida en la página web como texto plano mediante el proceso descrito en la sección anterior, para luego proceder a puntuar cada línea.

El algoritmo emplea el listado de palabras clave ingresado por el usuario para puntuar a cada línea. Por cada palabra clave se analiza cada línea para determinar si esta contiene la palabra, empleando tres criterios:

- Si la línea contiene en alguna parte de su texto la palabra clave, se suman dos puntos.
- Si la línea contiene exactamente la palabra clave más otro texto adicional, se suman tres puntos.

- Si la línea contiene exactamente la palabra clave sin otro texto adicional, se suman cuatro puntos.

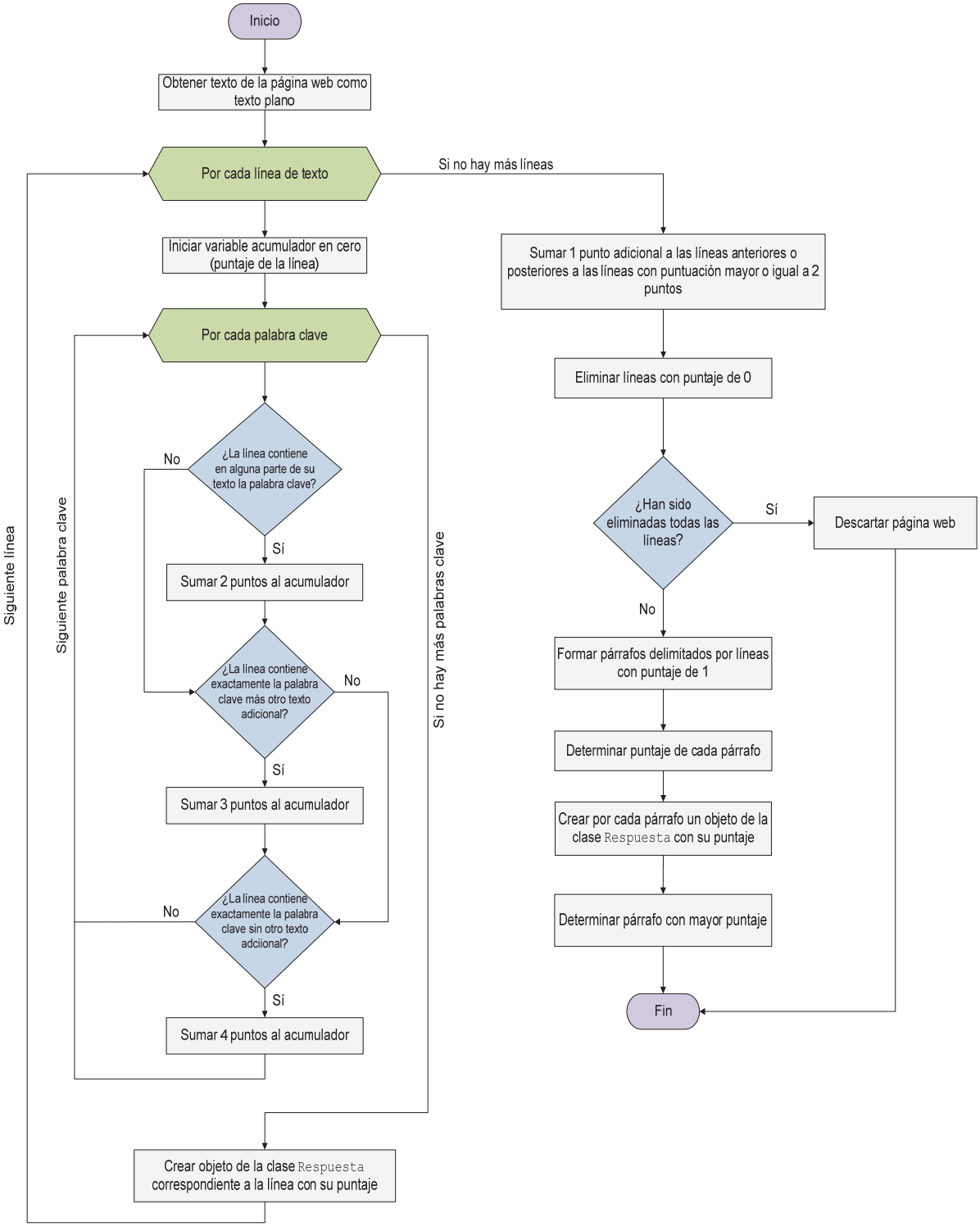


Figura 3.10 Algoritmo de puntuaciones

La puntuación total de cada línea corresponde a la suma de los puntuajes obtenidos por su análisis con cada palabra clave. Para obtener resultados más precisos, el algoritmo añade un punto adicional a cada línea que sea inmediatamente anterior o posterior a una línea con un puntaje mayor o igual a dos, pues es probable que el dato exacto que busca el usuario se encuentre entre estas líneas. Si la línea ha recibido puntaje (es decir, tiene un puntaje mayor o igual uno) se crea un objeto de la clase `Respuesta`[52] que representa la línea. Finalmente, aquellas líneas que no han recibido puntaje son eliminadas.

Luego, el texto resultante por las distintas líneas es separado en párrafos[53]. Las líneas con puntaje igual a uno determinan el inicio y final de cada párrafo. La puntuación total del parráfo se obtiene entonces con la suma de los puntajes de cada una de las líneas que lo conforman. Por cada párrafo se crea un objeto de la clase `Respuesta` que representa al párrafo.

El párrafo con el puntaje más alto es el que se presenta al usuario como resultado de búsqueda en la página web, pues existe una mayor probabilidad de que en dicho párrafo se encuentre la información que el usuario está buscando.

Puede darse el caso de que ninguna línea haya recibido puntaje de acuerdo a los criterios del algoritmo, de manera que todas las líneas hayan sido eliminadas. En este caso, el algoritmo considera que en la página web no existe información de interés para el usuario y no retornará resultados para dicha página web.

3.1.1.2.4 Servicio de Windows para búsqueda periódica de información [12], [18]

El servicio SOAP permite que el usuario pueda ingresar el listado de palabras clave así como los distintos sitios web de confianza, con lo cual puede realizar la búsqueda de la información inmediatamente. Sin embargo, también permite programar búsquedas periódicas de información en donde el usuario establece los días de la semana, horas y minutos en que el servicio SOAP debe realizar la

[52] La clase `Respuesta` es una de las clases de la estructura jerárquica de clases empleada por el servicio SOAP para guardar la información de las búsquedas en sitios web realizadas por los usuarios mediante serialización de objetos, y se describe en las secciones 3.1.1.4 y 3.1.1.8.7.

[53] La división en parráfos que realiza el algoritmo no corresponde a párrafos gramaticales, sino a un conjunto de líneas que recopilan la mayor cantidad de coincidencias con las palabras clave, y en las que es más probable se encuentre el dato exacto que el usuario está buscando.

búsqueda de información en los sitios web indicados por el usuario. Esta funcionalidad permite obtener información actualizada.

Para lograr esto, el servicio SOAP implementa un método que crea un hilo de ejecución en segundo plano, de manera que este hilo sea el encargado de comparar, empleando un bucle infinito, la hora actual del sistema con la de los horarios indicados por el usuario, y en caso de que alguno de los horarios indicado por el usuario coincida con la hora del sistema invoca al método de búsqueda en sitios web.

De igual manera, el servicio SOAP implementa un método encargado de eliminar este hilo de ejecución para detener la búsqueda de información.

Dado que el cliente del servicio SOAP no es capaz de invocar a estos métodos que crean y eliminan el hilo de ejecución, pues el servicio de búsqueda periódica debe poder ser iniciado o detenido todo un siempre y no tan solo mientras el cliente emplea la aplicación de escritorio, se ha creado un proyecto junto al servicio SOAP correspondiente a un servicio de Windows, el cual ha sido nombrado Buscador.

El servicio de Windows es implementado en base a una clase con dos métodos, uno que se ejecuta cada vez que el servicio es iniciado (método `OnStart()`), y otro que se ejecuta cada vez que el servicio es detenido (método `OnStop()`) [12].

En esta clase se incorpora el *proxy* que permite emplear el servicio SOAP, de manera que en cada uno de estos métodos se invoque a los métodos del servicio SOAP que permiten crear el hilo de ejecución y eliminarlo, respectivamente.

Finalmente, para que el servicio de Windows creado pueda ser iniciado o detenido al igual que el resto de servicios de Windows, este debe ser instalado [18]. Para la instalación es necesario compilar el proyecto y, desde el `prompt` del *"Símbolo del sistema para desarrolladores de VS2012"*, cambiar el directorio actual al directorio del proyecto y ejecutar el comando que se presenta en el Código 3.3. En este comando, `nombre_servicio.exe` corresponde al archivo ejecutable de extensión .exe que se obtiene al compilar el proyecto.

```
installutil.exe nombre_servicio.exe
```

Código 3.3 Comando para instalar servicio de Windows [18]

La Figura 3.11 presenta el servicio creado, correctamente instalado y listo para ser usado al igual que el resto de servicios de Windows.

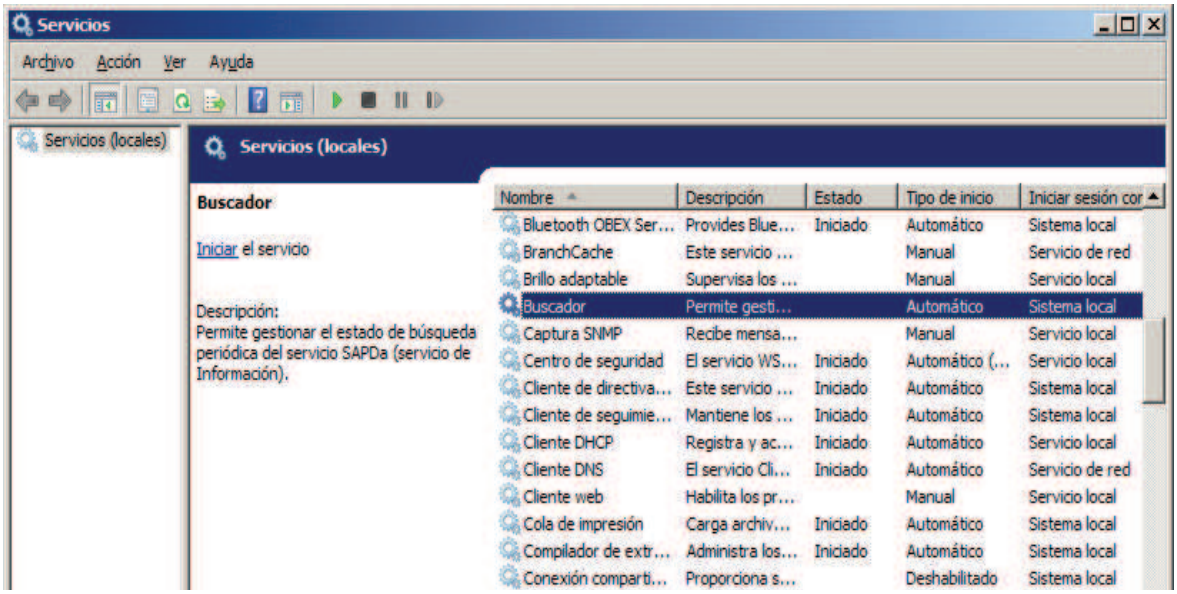


Figura 3.11 Servicio de búsqueda periódica de información instalado

Cuando se inicia el servicio Buscador, como se muestra en la Figura 3.12, el servicio SOAP crea el hilo de ejecución en segundo plano para realizar la búsqueda periódica de información de acuerdo a los horarios determinados por el usuario. Cuando se detiene el servicio, dicho hilo de ejecución se elimina y el servicio de búsqueda periódica de información ya no puede ser utilizado hasta que el servicio sea iniciado nuevamente.

3.1.1.3 Configuración de horarios de búsqueda periódica de información

El servicio SOAP implementa dos métodos de configuración orientados a obtener y establecer los horarios de búsqueda indicados por el usuario. Dichos métodos son empleados únicamente por la aplicación cliente.

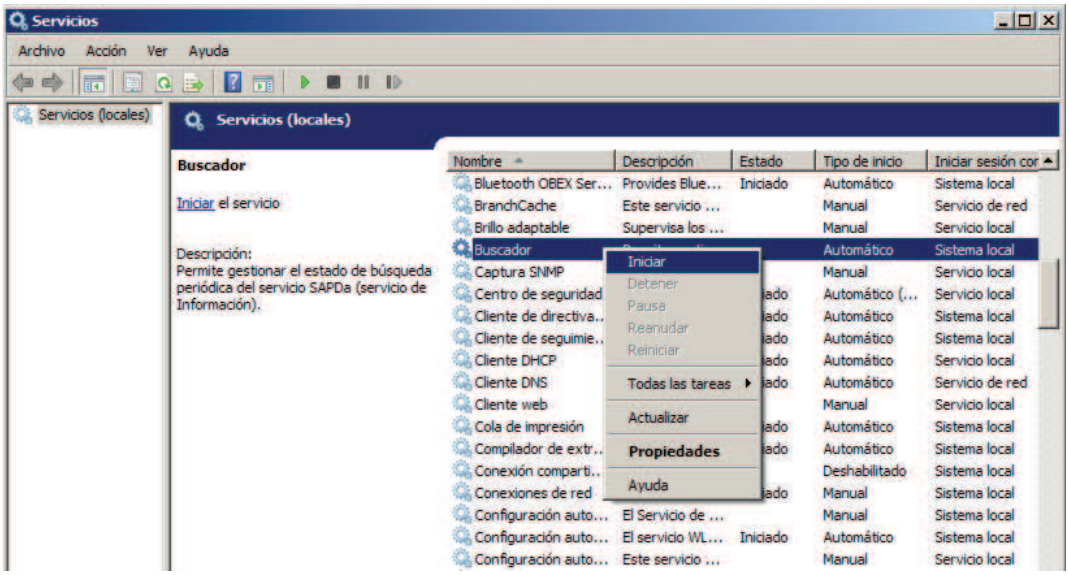


Figura 3.12 Servicio de búsqueda periódica de información siendo iniciado

La información encontrada por estas búsquedas es almacenada en un archivo binario en el servidor que aloja al servicio SOAP, de tal manera que el usuario al abrir la aplicación cliente pueda observar la información recolectada por el servicio web como sugerencias.

3.1.1.4 Almacenamiento de información de las búsquedas en sitios web realizadas por los usuarios en un archivo binario mediante serialización de objetos [3], [8], [10], [14]

Serializar un objeto significa convertir al mismo en un *stream* de *bytes*, de manera que dicho *stream* se pueda almacenar en un repositorio, como por ejemplo un archivo binario. Deserializar un objeto involucra realizar la operación contraria, esto es, leer el *stream* desde un archivo binario o el repositorio empleado y restablecer el objeto para que pueda ser utilizado, como sugiere la Figura 3.13.

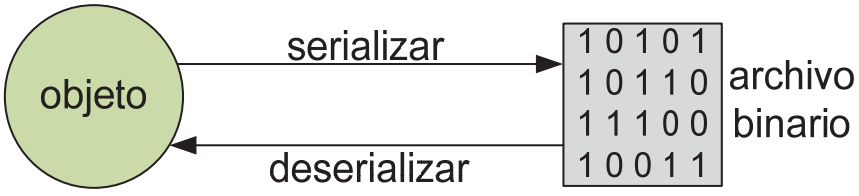


Figura 3.13 Serialización y deserialización de un objeto en un archivo binario

El servicio SOAP emplea la serialización de objetos con el fin de almacenar en un archivo binario la información correspondiente a los parámetros de búsqueda de

información de los usuarios, tales como los sitios web de confianza en donde buscar, los horarios de búsqueda periódica de información establecidos y los listados de palabras clave, así como los resultados de la ejecución de dichas búsquedas.

Para evitar el trabajo de serializar varios objetos, el servicio SOAP hace uso de una estructura jerárquica de clases que se presenta en la Figura 3.14, establecidas básicamente mediante asociaciones binarias *one way*[54] y relaciones binarias de composición[55], en el que una clase de nivel superior tiene una o varias referencias a una clase de nivel inferior [3].

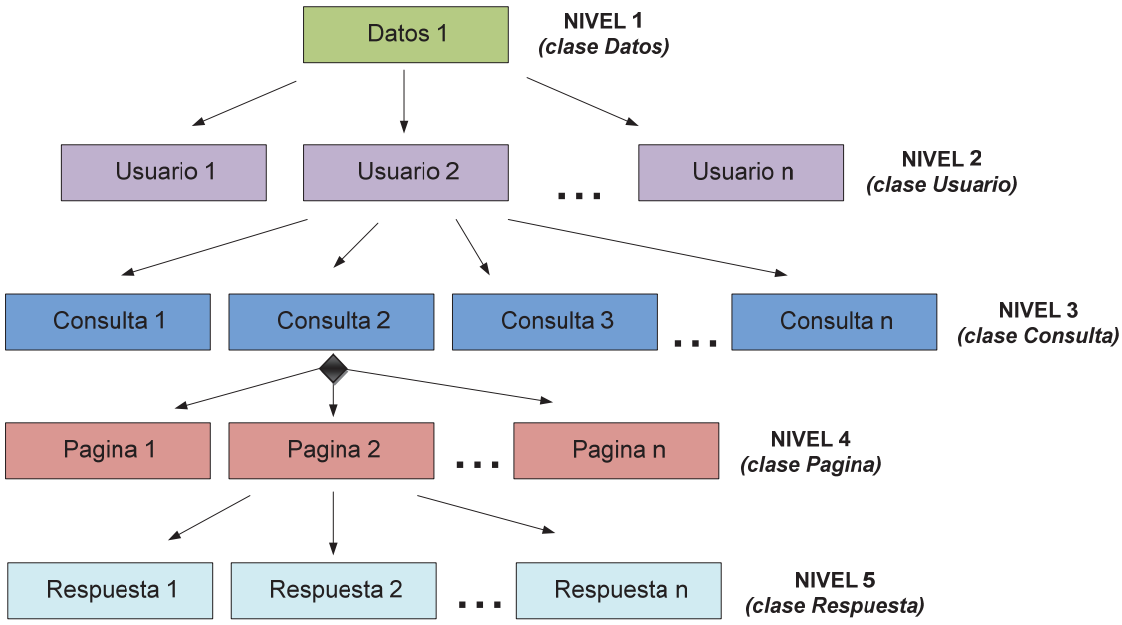


Figura 3.14 Estructura jerárquica de clases empleada por el servicio SOAP para guardar las búsquedas en sitios web realizadas por los usuarios

La clase de nivel superior consiste en una colección de la información de todos los usuarios, por lo que ha sido nombrada `Datos`[56]. Un objeto de esta clase tiene como atributos parámetros generales tales como la dirección IP para la comunicación con el servicio RESTful, la dirección IP para la comunicación con el servicio de bases de datos, la cadena de conexión con la cual el servicio SOAP se

[54] Relación débil entre dos clases, en la que la clase de nivel superior tiene una o varias referencias hacia la clase de nivel inferior [3].
[55] Relación fuerte entre dos clases, en la cual para que una instancia de la clase de nivel superior exista necesariamente debe existir por lo menos una instancia de la clase de nivel inferior [3].
[56] Aún cuando una clase debe ser nombrada en singular, se ha hecho una excepción a la regla pues el nombre en plural muestra la razón de la necesidad de esta clase.

conectará con SQL Server[57] y los horarios de búsqueda periódica de información. Un objeto de esta clase tiene referencias a objetos de tipo `Usuario`, lo cual permite almacenar toda la información del trabajo de los usuarios mediante la serialización de tan solo un objeto de la clase `Datos`.

Un objeto de la clase `Usuario` tiene como atributos el `ID` de usuario, la contraseña, el listado general de todos los URL de los sitios web de confianza del usuario que pueden ser empleados para las distintas búsquedas, entre otros.

Cada objeto de la clase `Usuario` tiene referencias a objetos de la clase `Consulta`. Un objeto de clase `Consulta` representa una acción de búsqueda por parte del usuario para obtener información específica correspondiente a una celda de una tabla. Por lo tanto, esta clase tiene como atributos el listado de palabras clave y el listado de URL de los sitios web de confianza indicados por el usuario para realizar esta búsqueda concreta, además del nombre de la base de datos, el nombre de la tabla y los números de fila y columna para el cual corresponde el dato que se busca.

Cada objeto de la clase `Consulta` tiene referencias a objetos de la clase `Pagina`, la cual representa una página web obtenida como resultado de la búsqueda en los sitios web de confianza. Un objeto de la clase `Pagina` tiene como atributo el URL de la página web que representa. Dado que un objeto de la clase `Consulta` no debería existir si no tiene al menos un objeto de la clase `Pagina`, existe una relación fuerte entre ambas clases representada por una relación de composición [3].

A partir del *screen scraping* de cada página realizado mediante la biblioteca `HTMLAgilityPack` se obtienen los párrafos resultantes, por lo que un objeto de la clase `Pagina` tiene referencias a objetos de la clase `Respuesta`, cada uno de los cuales representa una línea puntuada o un párrafo de la página web con información útil para el usuario.

[57] Ver sección 3.1.1.5.1.

Un objeto de la clase `Respuesta` tiene como atributos la línea o colección de líneas que conforman el parráfo y la puntuación de la línea o párrafo obtenido a través del algoritmo de puntuaciones.

Para que un objeto de una clase sea serializable se debe aplicar el atributo `[Serializable]` en la definición de la clase [14]. Si se intenta serializar un objeto de una clase en la que no se ha especificado que sus instancias son serializables, se producirá una excepción del tipo `SerializationException` [14]. De igual manera, si un objeto de una clase serializable tiene referencias a objetos de otras clases, estas también deberán poder ser serializables. Por lo tanto, para serializar un objeto de la clase `Datos`, todas las clases especificadas en la Figura 3.14 deberán llevar el atributo `[Serializable]`.

El proceso de serialización de un objeto en un archivo binario se realiza en Visual Studio por medio de un objeto de la clase `BinaryFormatter` [8]. Esta clase se encuentra en el espacio de nombres `System.Runtime.Serialization.Formatters.Binary` [8]. El Código 3.4 presenta el código en C# para la serialización de un objeto de la clase `Datos` en un *stream* de *bytes*. La línea 1 crea un objeto de la clase `Datos` llamado `datos` y la línea 2 crea un objeto de la clase `BinaryFormatter` llamado `formateador`. El objeto de la clase `Datos` será empleado para ir almacenando la información del trabajo de los usuarios, valiéndose de objetos de las clases de niveles inferiores.

```
1:    Datos datos = new Datos();
2:    BinaryFormatter formateador = new BinaryFormatter();
3:    Stream flujo = File.OpenWrite("path_al_archivo_binario");
4:    formateador.Serialize(flujo,datos);
```

Código 3.4 Código en C# para serializar un objeto en un archivo binario

En la línea 3 se hace uso del método `OpenWrite()` de la clase estática[58] `File` que tiene como argumento una cadena de texto con el *path* del archivo binario en

[58] Una clase estática (*static class*) no puede ser instanciada, por lo que sus métodos deben ser invocados empleando directamente el nombre de la clase.

donde se almacenará el objeto serializado [10]. Si dicho archivo no existe en el *path* especificado, la clase `File` creará el archivo; y si existe el archivo se reescribirá. Esta operación retorna un objeto de la clase `Stream`[59]. Para emplear la clase `File` se debe incorporar el espacio de nombres `System.IO` [10]. Finalmente, en la línea 4 se hace uso del objeto de la clase `BinaryFormatter` para invocar al método `Serialize()` cuyos argumentos son el objeto de la clase `Stream` anteriormente obtenido y el objeto a serializarse.

El Código 3.5 presenta el código en C# para el proceso de deserialización. La línea 1 crea el objeto de la clase `BinaryFormatter` llamado `formateador` al igual que en el caso anterior. En la línea 2 se emplea el método estático `OpenRead()` de la clase `File` para poder leer el contenido del archivo binario cuyo *path* se indica como una cadena de texto. Esta operación retorna un objeto de la clase `Stream`. Finalmente, en la línea 3 se deserializa el *stream* del archivo especificado mediante el método `Deserialize()` del objeto de la clase `BinaryFormatter`, cuyo argumento es el objeto de la clase `Stream`. Esta operación retorna el objeto de la clase `Datos`, el cual puede ser manipulado con una referencia de dicha clase.

```
1:    BinaryFormatter formateador = new BinaryFormatter();
2:    Stream flujo = File.OpenRead("path_al_archivo_binario");
3:    Datos datos = formateador.Deserialize(flujo);
```

Código 3.5 Código en C# para deserializar un objeto almacenado en un archivo binario

3.1.1.5 Gestión de bases de datos para almacenamiento y procesamiento de información [7], [15], [17], [28]

SQL Server es un sistema gestor de bases de datos (SGBD) que permite la creación y manipulación de bases de datos relacionales, por lo que trabaja fundamentalmente con tablas [28]. Una base de datos en SQL Server está conformada por una o más tablas que almacenan información y puede ser

[59] En realidad el objeto retornado es un objeto de la clase `FileStream`. Sin embargo, puede emplearse con una referencia de la clase `Stream` de la cual hereda [10], lo cual facilita su posterior uso con el objeto de la clase `BinaryFormatter` [8].

definida, manipulada y controlada a través del lenguaje SQL (*Structured Query Language*).

SQL es un lenguaje declarativo en el que solo se debe expresar lo que se desea hacer sobre las bases de datos mediantes sentencias y el motor de SQL Server[60] realiza la acción indicada y retorna el resultado obtenido. Existen dos versiones del lenguaje SQL: Transact-SQL (T-SQL) y Ansi-SQL, siendo T-SQL el lenguaje nativo de SQL Server.

Para que el servicio SOAP pueda gestionar las bases de datos con el fin de almacenar y procesar información, este debe convertirse en un cliente de SQL Server para indicar con sentencias las acciones que desea realizar sobre las bases de datos, como lo indica la Figura 3.15.

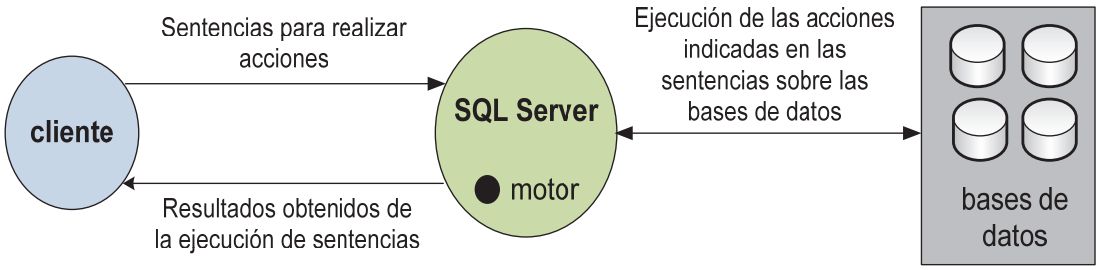


Figura 3.15 Gestión de bases de datos con SQL Server

Incluyendo el espacio de nombres `System.Data.SqlClient`, se tiene a disposición una colección completa de clases para convertir a cualquier aplicación en un cliente de SQL Server y así poder establecer una comunicacón cliente-servidor con el fin de manipular las bases de datos [17]. Algunas de las clases que ofrece este espacio de nombres `System.Data.SqlClient` se presentan en la Tabla 3.3.

El procedimiento de comunicación entre el cliente y SQL Server se presenta en la Figura 3.16.

[60] El motor es el servicio principal de SQL Server que permite almacenar, procesar y proteger los datos [7].

Tabla 3.3 Algunas clases del espacio de nombres `System.Data.SqlClient` [17]

Clase	Descripción
`SqlConnection`	Representa una conexión abierta a una base de datos de SQL Server.
`SqlCommand`	Representa una sentencia T-SQL o un procedimiento[61] que se ejecuta en una base de datos de SQL Server.
`SqlDataReader`	Proporciona una forma de leer una tabla resultante de la ejecución de sentencias T-SQL mediante la lectura progresiva de sus filas.
`SqlException`	Excepción que se produce cuando SQL Server devuelve una advertencia o un error.

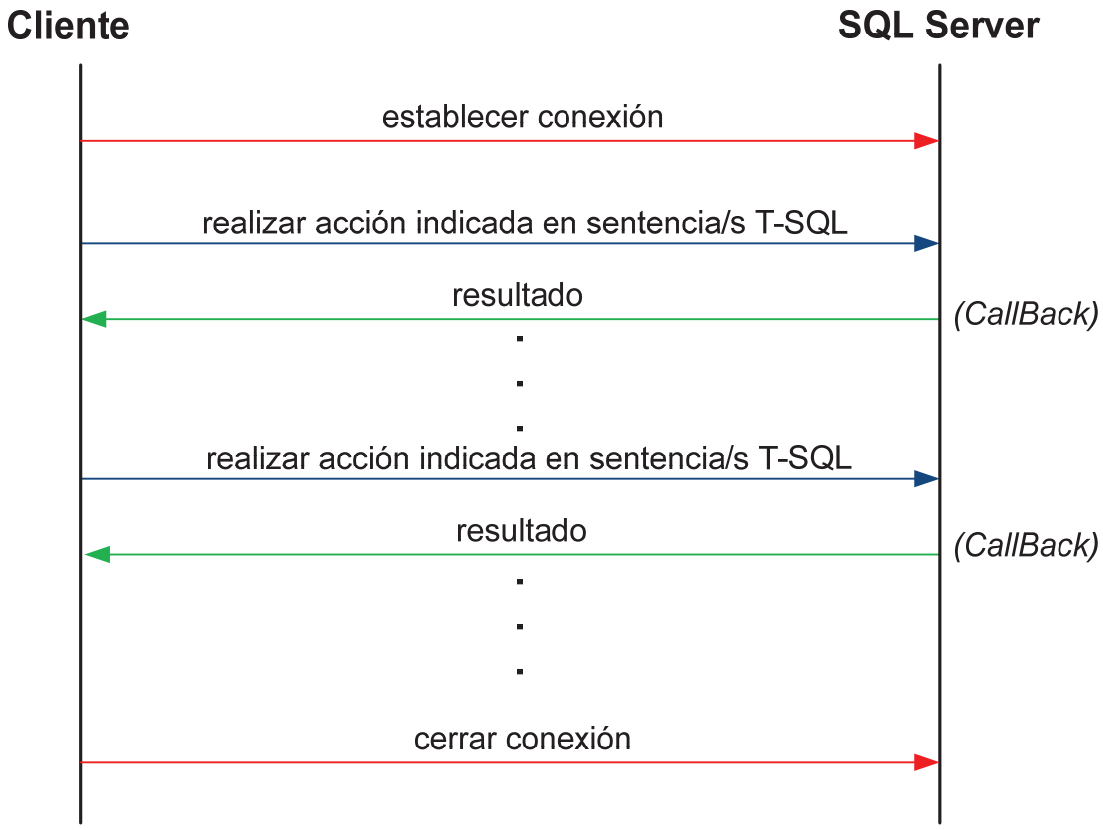


Figura 3.16 Procedimiento de comunicación entre cliente y SQL Server

En primer lugar, el cliente debe establecer una conexión con SQL Server. Para establecer esta conexión se emplea la clase `SqlConnection`. El constructor de esta clase tiene como parámetro una cadena de texto referente a la conexión que el cliente desea establecer[62]. Un objeto de esta clase permite enviar a SQL Server

[61] Conjunto de sentencias T-SQL.
[62] Ver sección 3.1.1.5.1.

una petición de conexión cuando se invoca a su método `Open()`. En caso de que SQL Server rechace la conexión o se produzca algún otro problema que le impida establecer la conexión, se generará una excepción `SqlException`.

Una vez establecida la conexión con SQL Server, el cliente debe indicar mediante sentencias T-SQL las acciones que desea que SQL Server realice sobre las bases de datos. Para ello, se debe crear un objeto de la clase `SqlCommand` [15]. El constructor de esta clase tiene como parámetros una cadena de texto que expresa una sentencia T-SQL o un grupo de sentencias T-SQL (procedimiento), y el objeto de la clase `SqlConnection` creado anteriormente [15].

La clase `SqlCommand` implementa una serie de métodos que permiten solicitar que SQL Server ejecute la sentencia y retorne el resultado obtenido (*CallBack*). El tipo de respuesta enviado por SQL Server depende del método empleado para solicitar la ejecución de la acción. Algunos de estos métodos se presentan en la Tabla 3.4. El cliente deberá implementar el código necesario para el tratamiento de la información en la respuesta enviada por SQL Server.

Tabla 3.4 Algunos métodos de la clase `SqlCommand` [15]

Método	Descripción
`ExecuteReader()`	Ejecuta la/s sentencia/s T-SQL y retorna un objeto de la clase `SqlDataReader` que permite obtener la información mediante la lectura progresiva de la/s fila/s de la tabla resultante por la ejecución de la/s sentencia/s.
`ExecuteNonQuery()`	Ejecuta la/s sentencia/s T-SQL y retorna el número de filas afectadas. Si la acción no produce filas afectadas retorna el valor de -1. Es útil para ejecutar sentencias para las cuales SQL Server no retorna información en su respuesta.
`ExecuteScalar()`	Ejecuta la/s sentencia/s T-SQL y retorna la primera columna de la primera fila del conjunto de resultados devuelto por la ejecución de la/s sentencia/s. Las demás columnas o filas no se tienen en cuenta.
`ExecuteXmlReader()`	Ejecuta la/s sentencia/s T-SQL y retorna un objeto de la clase `XmlReader` que contiene el resultado de la acción en formato XML.

Una vez que el cliente ha terminado su trabajo con las bases de datos, la conexión debe ser cerrada invocando al método `Close()` del objeto de la clase `SqlConnection`.

3.1.1.5.1 Cadena de texto para la conexión con SQL Server [5]

La cadena de texto del constructor de la clase `SqlConnection` determina los parámetros necesarios para establecer una conexión con SQL Server. Esta cadena de conexión tiene el siguiente formato [5]:

`DataSource=``dirección_servidor`**`;UserID=`**`id_usuario`**`;Password=`**`password`

Donde:

- **`dirección_servidor:`** Representa la dirección IP o el nombre de dominio[63] que permite ubicar al servidor de bases de datos en la red. Este parámetro es obligatorio.
- **`id_usuario:`** Es una cadena de texto correspondiente al identificador de usuario. Este parámetro es obligatorio. Cada sentencia T-SQL ejecutada por SQL Server se lleva a cabo especificando el identificador (`ID`) de un usuario concreto.
- **`password:`** Es una cadena de texto con la contraseña necesaria para que el usuario pueda hacer uso de las bases de datos. Este parámetro es necesario únicamente si el usuario ha condicionado la seguridad de su cuenta con una contraseña.

Si los parámetros de la cadena de conexión son correctos, al invocar al método `Open()` del objeto de la clase `SqlConnection` se establecerá la conexión entre el cliente y SQL Server, de manera que el cliente pueda manipular las bases de datos disponibles para el usuario indicado en la cadena de conexión, así como realizar las acciones que le estén permitidas a dicho usuario acorde a los permisos establecidos en SQL Server.

---

[63] En caso de emplear un nombre de dominio para ubicar al servidor de bases de datos, también será necesario hacer uso de un servidor DNS (*Domain Name System*).

El servicio web WCF emplea únicamente un usuario SQL *master* cuyo `ID` es *adminNCL*, correspondiente al administrador del sistema, el cual tiene acceso total a todas las bases de datos con todos los permisos. El usuario administrador no se puede eliminar.

3.1.1.5.2 Creación de una base de datos [5], [15]

El Código 3.6 presenta la sentencia T-SQL necesaria para crear una nueva base de datos. SQL Server no retorna información al ejecutar esta sentencia [15].

```
CREATE DATABASE nombre_BD
```

Código 3.6 Sentencia T-SQL para crear una base de datos [5]

Una base de datos creada corresponde a un tema de una aplicación interactiva que se está desarrollando; por lo tanto, se tendrá una base de datos por cada aplicación interactiva, generada automáticamente por el servicio SOAP.

3.1.1.5.3 Obtener información de las bases de datos disponibles para un usuario [5],[15]

El Código 3.7 indica la sentencia T-SQL para obtener información de las bases de datos disponibles para el usuario que solicitó ejecutar esta sentencia. Esta información está almacenada en la tabla `SYS.DATABASES` de SQL Server [15], la cual es retornada al ejecutarse la sentencia.

```
SELECT *
FROM SYS.DATABASES
```

Código 3.7 Sentencia T-SQL para obtener información de las bases de datos disponibles en SQL Server para un usuario [5]

3.1.1.5.4 Modificación del nombre de una base de datos [5], [15]

El Código 3.8 presenta la sentencia T-SQL que permite modificar el nombre de una determinada base de datos. Como resultado de esta acción, SQL Server no retorna información [15].

```
ALTER DATABASE nombre_BD
MODIFY BY NAME = nuevo_nombre_BD
```

Código 3.8 Sentencia T-SQL para modificar el nombre de una base de datos [5]

3.1.1.5.5 Eliminación de una base de datos [5], [15]

El Código 3.9 muestra la sentencia T-SQL para eliminar una base de datos. Esta acción no provoca que SQL Server retorne información [15].

```
DROP DATABASE nombre_BD
```

Código 3.9 Sentencia T-SQL para eliminar una base de datos [5]

3.1.1.5.6 Creación de una tabla [5], [15]

El Código 3.10 presenta la sentencia T-SQL necesaria para crear una tabla de una determinada base de datos. SQL Server no retorna información al ejecutar esta sentencia [15].

```
USE nombre_BD
CREATE TABLE nombre_tabla
(nombre_atributo1 TIPO,
nombre_atributo2 TIPO, …)
```

Código 3.10 Sentencia T-SQL para crear una tabla de una base de datos [5]

Una tabla creada para una determinada base de datos corresponde a un subtema de una aplicación interactiva. Por lo tanto, con el servicio SOAP se podrán crear tantas tablas como el usuario requiera, para almacenar en cada una de ellas la información correspondiente a un subtema específico de una aplicación interactiva que se esté desarrollando.

3.1.1.5.7 Obtención de información de las tablas que conforman una base de datos [5], [15]

Para conocer qué tablas conforman una determinada base de datos se puede emplear la sentencia T-SQL que se presenta en el Código 3.11. El resultado de

esta operación es una tabla con la información de las tablas que son parte de la base de datos especificada con su nombre después del comando `USE` [15].

```
USE nombre_BD
SELECT *
FROM INFORMATION_SCHEMA.TABLES
```

Código 3.11 Sentencia T-SQL para obtener información de las tablas que conforman una base de datos [5]

3.1.1.5.8 Obtención de una tabla [5], [15]

Con el fin de obtener una determinada tabla de una base de datos, se puede emplear la sentencia T-SQL que presenta el Código 3.12. El resultado obtenido será la tabla completa de la base de datos especificada [15].

```
USE nombre_BD
SELECT *
FROM nombre_tabla
```

Código 3.12 Sentencia T-SQL para obtener una tabla completa de una base de datos [5]

3.1.1.5.9 Modificación del nombre una tabla [5], [15]

Es posible renombrar una tabla que conforma una base de datos mediante el comando `SP_RENAME`, el cual se utiliza en la sentencia que se muestra en el Código 3.13. SQL Server no retorna información al ejecutar esta sentencia [15].

```
USE nombre_BD
SP_RENAME anterior_nombre_tabla, nuevo_nombre_tabla
```

Código 3.13 Sentencia T-SQL para modificar el nombre de una tabla [5]

3.1.1.5.10 Inserción de filas en una tabla [5], [15]

Para insertar una nueva fila en una tabla de una determinada base de datos, se pueden emplear la sentencia T-SQL que presenta el Código 3.14. Esta sentencia

añade al final de la tabla una nueva fila con la información indicada a través del comando `VALUES`. SQL Server no retorna información al ejecutar esta sentencia [15].

```
USE nombre_BD
INSERT INTO nombre_tabla
VALUES (valor1, valor2, valor3, …)
```

Código 3.14 Sentencia T-SQL para insertar una fila en una tabla [5]

3.1.1.5.11 Eliminación de una tabla [5], [15]

El Código 3.15 presenta la sentencia T-SQL para eliminar una tabla de una determinada base de datos. SQL Server no retorna información al ejecutar esta sentencia [15].

```
USE nombre_BD
DROP TABLE nombre_tabla
```

Código 3.15 Sentencia T-SQL para eliminar una tabla [5]

3.1.1.6 Gestión de usuarios

El servicio SOAP posibilita la creación de usuarios, permitiendo el trabajo colaborativo en la introducción de información en las bases de datos para la creación de aplicaciones interactivas. Estos usuarios son propios del servcio SOAP, no usuarios de SQL Server. Los distintos usuarios que colaboran con el administrador en la gestión de bases de datos indican a SQL Server realizar las operaciones en las bases de datos en nombre del usuario administrador *adminNCL*.

Cada usuario que se crea podrá consultar información mediante el *bot* de búsqueda del servicio SOAP e ingresar la información que considere adecuada en las bases de datos.

La búsqueda de información es independiente entre cada uno de los usuarios, y se maneja de manera adecuada mediante la creación de instancias de la clase `Usuario`.

3.1.1.7 Casos de Uso

Para el análisis de los casos de uso de la aplicación cliente se consideran dos actores. El primer actor corresponde al administrador (usuario *adminNCL*), el cual es el encargado de realizar las acciones de búsqueda, almacenamiento y procesamiento de información, así como también de determinar los parámetros de configuración del servicio web y gestionar usuarios. El segundo actor corresponde a un usuario que ha sido creado por el administrador para ayudar en las acciones de búsqueda, almacenamiento y procesamiento de información de las aplicaciones interactivas que se estén desarrollando.

El diagrama de casos de uso general del servicio SOAP se presenta en la Figura 3.17.

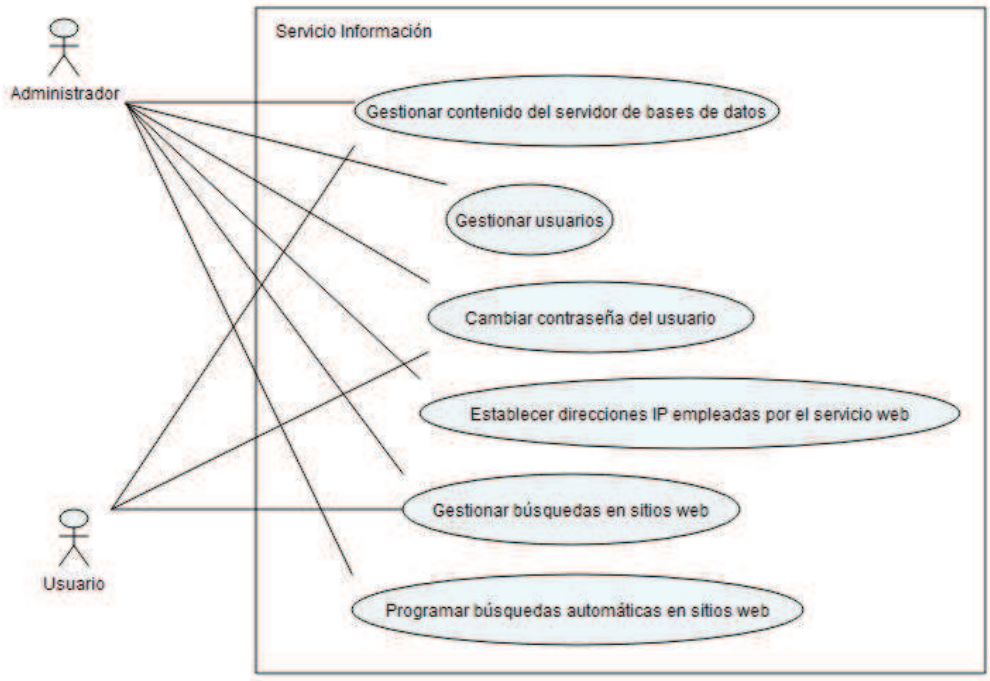


Figura 3.17 Diagrama de casos de uso general del servicio SOAP

El administrador puede realizar las siguientes acciones:

- Gestionar el contenido del servidor de bases de datos.
- Gestionar usuarios.
- Cambiar la contraseña del usuario, es decir, la contraseña de su propia cuenta.
- Establecer las direcciones IP empleadas por el servicio web.
- Gestionar búsquedas en sitios web.
- Programar búsquedas automáticas en sitios web, es decir, establecer los horarios de la búsqueda periódica de información.

Por otro lado, a un usuario creado por el usuario administrador únicamente le está permitido:

- Gestionar el contenido del servidor de base de datos.
- Cambiar la contraseña propia del usuario.
- Gestionar búsquedas en sitios web.

3.1.1.7.1 Gestionar contenido del servidor de base de datos

Los casos de uso para la gestión del contenido del servidor de base de datos se detallan en la Figura 3.18. Estos casos de uso comprenden acciones orientadas a crear, modificar y eliminar temas (bases de datos) y subtemas (tablas) en SQL Server, así como el ingreso y modificación de información.

Tanto el usuario administrador así como cualquier otro usuario que este haya creado pueden:

- **Crear un nuevo tema:** Crear una nueva base de datos en SQL Server especificando el nombre de la base de datos.
- **Editar un tema:** Modificar el nombre de una base de datos anteriormente creada.
- **Eliminar un tema:** Eliminar una base de datos anteriormente creada.
- **Crear un nuevo subtema:** Crear una nueva tabla para una determinada base de datos especificando el nombre de la tabla.
- **Editar un subtema:** Modificar el nombre de una tabla anteriormente creada.

- **Eliminar un subtema:** Eliminar una tabla anteriormente creada.
- **Cambiar las dimensiones del subtema (tabla):** Modificar las dimensiones de una tabla especificando el número de filas y de columnas que la conforman.
- **Editar el contenido de celdas del subtema (tabla):** Ingresar y modificar la información almacenada en las celdas de una determinada tabla. La información puede ser ingresada de forma manual o haciendo uso del *bot* de búsqueda.

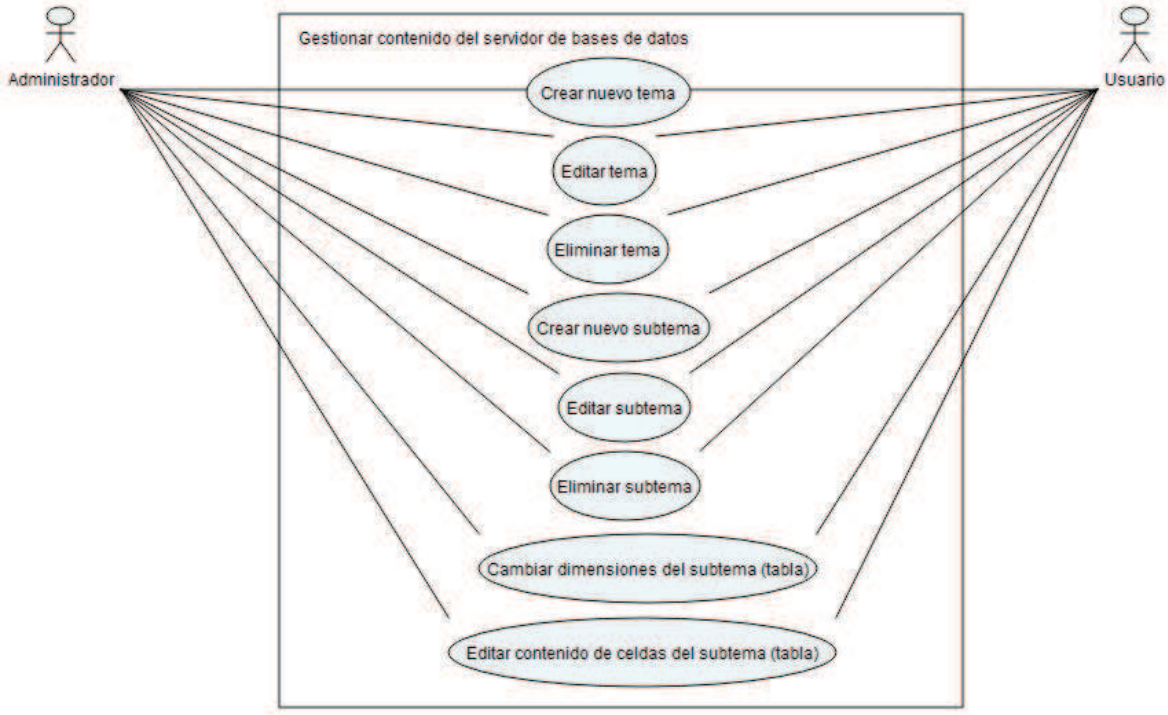


Figura 3.18 Diagrama de casos de uso para gestionar contenido del servidor de bases de datos

3.1.1.7.2 Gestionar usuarios

La gestión de usuarios únicamente la puede realizar el usuario administrador como se muestra en la Figura 3.19.

El usuario administrador puede:

- **Crear un nuevo usuario:** Para ello, el administrador deberá especificar el nombre de usuario y su contraseña.

- **Editar un usuario:** El administrador puede modificar el nombre de usuario y/o la contraseña (credenciales) de un determinado usuario.
- **Eliminar un usuario:** El administrador puede eliminar un usuario específico si así lo requiere.

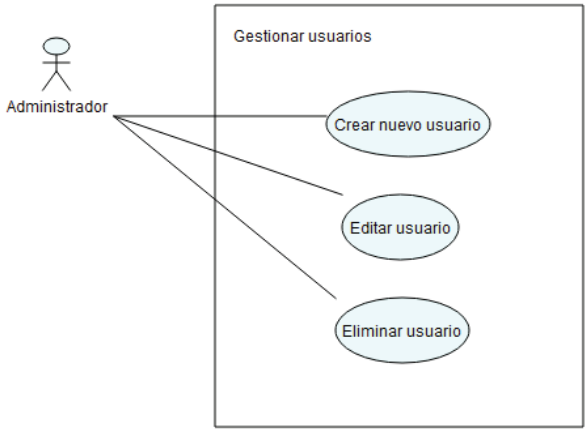


Figura 3.19 Diagrama de casos de uso para gestionar usuarios

3.1.1.7.3 Establecer las direcciones IP empleadas por el servicio web

Estos casos de uso comprenden el establecimiento de la dirección IP con la cual se realiza la conexión con el servidor de base de datos, así como de la dirección IP con la cual la aplicación Mixer generará el *script* Lua para al comunicación con el servidor que aloja al servicio RESTful (servidor de retorno), como se presenta en la Figura 3.20.

Únicamente el usuario administrador es capaz de establecer los valores de estas direcciones.

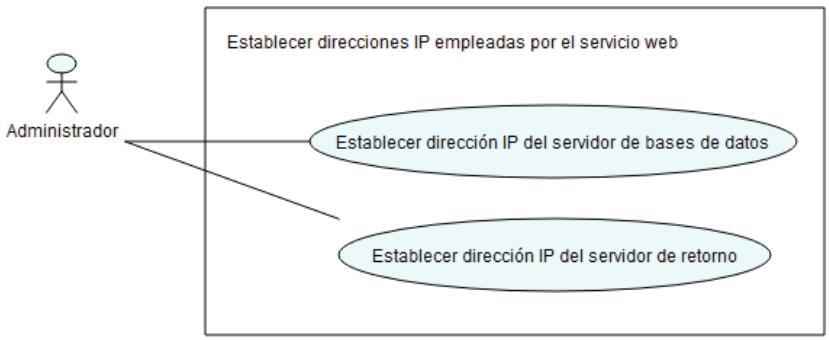


Figura 3.20 Diagrama de casos de uso para establecer las direcciones IP empleadas por el servicio web

3.1.1.7.4 Gestionar búsquedas en sitios web

La Figura 3.21 detalla los casos de uso para la gestión de búsquedas en sitios web, realizadas para obtener el contenido a ser ingresado en las bases de datos.

Tanto el usuario administrador así como cualquier otro usuario que este haya creado pueden:

- Ingresar las direcciones de sitios web de confianza.
- Eliminar las direcciones de sitios web de confianza.
- Ingresar las palabras clave para una búsqueda determinada.
- Seleccionar las direcciones de confianza para una búsqueda determinada.
- Realizar la búsqueda.
- Eliminar una búsqueda realizada.

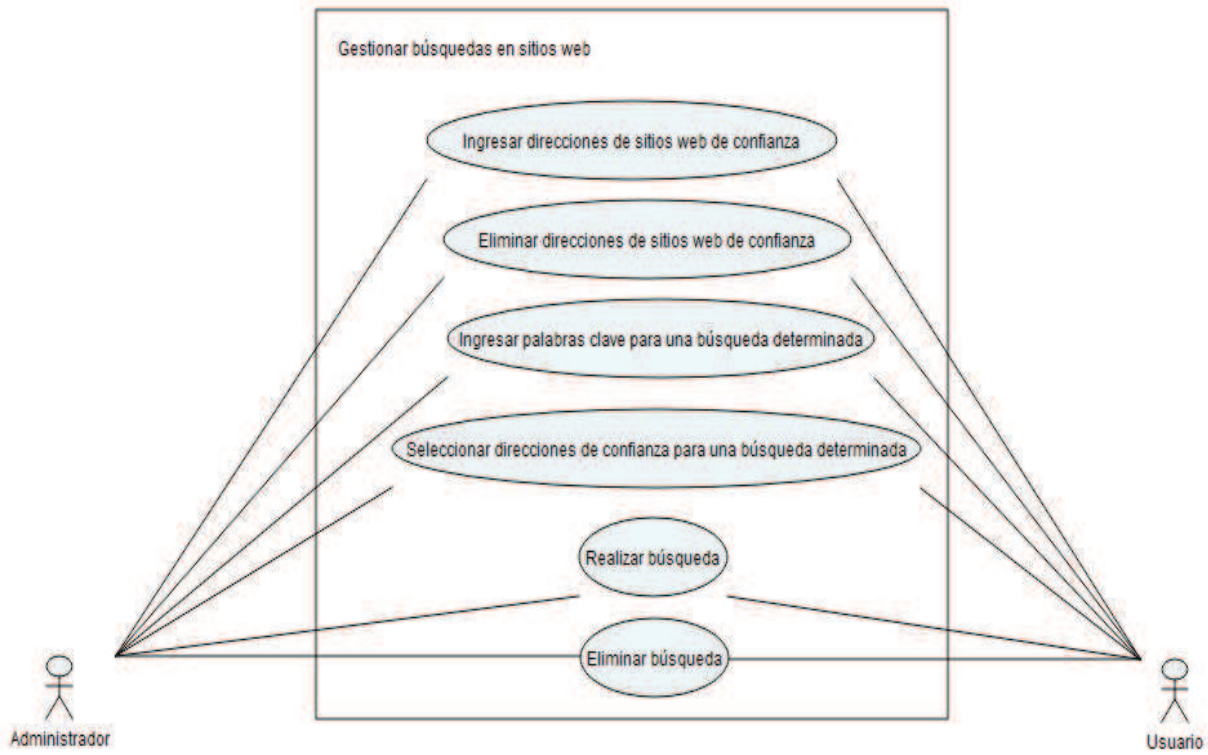


Figura 3.21 Diagrama de casos de uso para gestionar búsquedas en sitios web

3.1.1.8 Implementación

Para el servicio SOAP se ha implementado:

- Una interfaz, denominada `IServicioInformacion`, correspondiente al contrato del servicio SOAP.
- Una clase, denominada `ServicioInformacion`, que implementa los métodos de la interfaz `IServicioInformacion`.
- La estructura jérarquica de clases descrita en la sección 3.1.1.4.

3.1.1.8.1 Interfaz IServicioInformacion [23]

La interfaz `IServicioInformacion` del servicio SOAP corresponde al contrato del servicio (componente *Contract*) que determina los métodos que publica el servicio.

Antes de la firma[64] de la interfaz se deberá especificar el atributo `[ServiceContract]` para indicar que la interfaz corresponde al contrato del servicio, y de igual manera, antes de la firma de cada método se deberá especificar el atributo `[OperationContract]` para indicar que el método debe ser publicado por el servicio SOAP [23].

3.1.1.8.2 Clase ServicioInformacion [23]

La clase `ServicioInformacion` del servicio SOAP implementa los métodos cuyas firmas fueron declaradas en la interfaz `IServicioInformacion`.

La Figura 3.22 presenta el diagrama de la clase `ServicioInformacion` heredando de la interfaz `IServicioInformacion`. La clase `ServicioInformacion` está obligada a implementar todos los métodos cuyas firmas se especificaron en la interfaz `IServicioInformacion` [23].

[64] La firma de una clase, de una interfaz o de un método es la sentencia que se emplea para su declaración.

Figura 3.22 Diagrama de la clase `ServicioInformacion` e interfaz `IServicioInformacion` del servicio SOAP

Los atributos que emplea la clase `ServicioInformacion` del servicio SOAP son:

- **`datos:`** Corresponde a un objeto de la clase `Datos`.
- **`formateadorBinario:`** Atributo de tipo `BinaryFormatter`. Formateador binario necesario para el proceso de serialización y deserialización del objeto de la clase `Datos`.
- **`hiloProgramacion:`** Atributo de tipo `Thread`. Hilo de ejecución en segundo plano para realizar la búsqueda periódica de información.
- **`hiloDetener:`** Atributo de tipo `bool`. Si toma el valor de `true` indica que el hilo de ejecución en segundo plano debe ser detenido (pausado).
- **`hiloTerminar:`** Atributo de tipo `bool`. Si toma el valor de `true` indica que el hilo de ejecución en segundo plano debe ser eliminado.

Los métodos cuyas firmas se declararon en la interfaz `IServicioInformacion` y que son implementados por la clase `ServicioInformacion` se detallan en la Tabla 3.5. Estos métodos son publicados por el servicio SOAP para que puedan ser invocados por sus clientes.

Tabla 3.5 Métodos que publica el servicio SOAP (primera parte)

Método	Funcionalidad
`void ArrancarBusqueda ()`	Inicia el servicio de búsqueda periódica.
`void DetenerBusqueda ()`	Detiene el servicio de búsqueda periódica.
`bool VerificarConexionBaseDatos (string nombre)`	Verifica la conexión al servidor de bases de datos para un determinado `ID` de usuario.
`void CambiarDireccionIPBaseDatos (string direccionIP)`	Cambia la dirección IP empleada para la conexión con el servidor de bases de datos.
`string LeerDireccionIPBaseDatos ()`	Lee la dirección IP empleada para la conexión con el servidor de bases de datos.
`void CambiarDireccionIPRetorno (string direccionIP)`	Cambia la dirección IP empleada para la conexión con el servidor de retorno.
`string LeerDireccionIPRetorno ()`	Lee la dirección IP empleada para la conexión con el servidor de retorno.

Tabla 3.5 Métodos que publica el servicio SOAP (segunda parte)

Método	Funcionalidad
`void CambiarProgramacion (string[] programacion)`	Modifica la lista de horarios de programación para la realización de búsquedas periódicas.
`ArrayList LeerProgramacion ()`	Lee la lista de horarios de programación para la realización de búsquedas periódicas.
`string AnadirUsuario (string nombre, string contrasena)`	Añade un nuevo usuario al sistema.
`string EditarUsuario (string nombreAntiguo, string nombreNuevo, string contrasena)`	Edita las credenciales de un usuario.
`string EliminarUsuario (string nombre)`	Elimina un usuario del sistema.
`ArrayList ListarUsuarios ()`	Lista los usuarios registrados.
`bool AnadirUrlUsuario (string nombre, string url)`	Añade un nuevo URL de un sitio web de confianza a la lista de URL del usuario.
`bool EliminarUrlUsuario (string nombre, string url)`	Elimina un URL de la lista de URL de sitos web de confianza del usuario.
`ArrayList ListarUrlsUsuario (string nombre)`	Lista los URL de confianza del usuario.
`bool Autenticar (string nombre, string contrasena)`	Autentica a un usuario ante SQL Server, retornando el valor de `true` si el usuario es autenticado con éxito.
`string AnadirTema (string nombreUsuario, string nombreTema)`	Crea un nuevo tema (base de datos).
`string EditarTema (string nombreUsuario, string nombreAntiguoTema, string nombreNuevoTema)`	Edita el nombre de un tema (base de datos).
`string EliminarTema (string nombreUsuario, string nombreTema)`	Elimina un tema (base de datos).
`ArrayList ListarTemas (string nombreUsuario)`	Lista los temas (bases de datos) registrados en SQL Server.
`string AnadirSubTema (string nombreUsuario, string nombreTema, string nombreSubTema, int columnas)`	Crea un nuevo subtema (tabla).

Tabla 3.5 Métodos que publica el servicio SOAP (tercera parte)

Método	Funcionalidad
`string GuardarContenido (string nombreUsuario, string nombreTema, string nombreSubTema, string[] contenidoP)`	Guarda el contenido ingresado dentro de un subtema (tabla).
`string GuardarContenidoCelda (string nombreUsuario, string nombreTema, string nombreSubTema, string contenidoP, int fila, int columna)`	Guarda el contenido de una celda específica dentro de un subtema (tabla).
`ArrayList LeerContenido (string nombreUsuario, string nombreTema, string nombreSubTema)`	Lee el contenido existente dentro de un subtema (tabla).
`string EditarNombreSubTema (string nombreUsuario, string nombreTema, string nombreAntiguoSubTema, string nombreNuevoSubTema)`	Edita el nombre de un subtema (tabla).
`string EditarSubTema (string nombreUsuario, string nombreTema, string nombreSubTema, int filas, int columnas)`	Edita las dimensiones de un subtema (tabla).
`string EliminarSubTema (string nombreUsuario, string nombreTema, string nombreSubTema)`	Elimina un subtema (tabla).
`ArrayList ListarSubTemas (string nombreUsuario, string nombreTema)`	Lista los subtemas (tablas) que conforman un tema (base de datos).
`ArrayList AbrirConsulta (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna)`	Abre los resultados de una determinada consulta (búsqueda).
`ArrayList ListarUrlsConsulta (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna)`	Lista los URL de las páginas web que se han usado para realizar la consulta (búsqueda) en un sitio web.

Tabla 3.5 Métodos que publica el servicio SOAP (cuarta parte)

Método	Funcionalidad
`ArrayList ListarClavesConsulta (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna)`	Lista las palabras clave que se han usado para realizar la consulta (búsqueda) en un sitio web.
`string EliminarConsulta (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna)`	Elimina una consulta (búsqueda) realizada.
`ArrayList HacerConsulta (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna, string[] urlsConfianzaP, string[] clavesP)`	Realiza una consulta (búsqueda) en un sitio web de confianza y pospone la búsqueda periódica.

Además, en la clase `ServicioInformacion` del servicio SOAP se implementan métodos de apoyo que, aunque no son publicados, son invocados por los diferentes métodos que publica el servicio para poder cumplir con su funcionalidad. Estos métodos se detallan en la Tabla 3.6.

Tabla 3.6 Métodos de apoyo empleados por el servicio SOAP (primera parte)

Método	Funcionalidad
`ServicioInformacion()`	Constructor.
`void BusquedaPeriodica ()`	Inicia la búsqueda periódica de información.
`Usuario EncontrarUsuarioNombre (string nombre)`	Busca un usuario por su nombre.
`Consulta EncontrarConsulta (Usuario usuario, string nombreTema, string nombreSubtema, int fila, int columna)`	Encuentra una determinada consulta (búsqueda) realizada en un sitio web de confianza.

Tabla 3.6 Métodos de apoyo empleados por el servicio SOAP (segunda parte)

Método	Funcionalidad
`Consulta HacerConsultaNT (string nombreUsuario, string nombreTema, string nombreSubTema, int fila, int columna, string[] urlsConfianzaP, string[] clavesP)`	Realiza una consulta (búsqueda) en un sitio web de confianza.
`ArrayList ObtenerListaEnlaces (ArrayList urlsConfianza, ArrayList claves)`	Obtiene una lista de enlaces (URL) específicos a partir de una lista de URL de sitios web de confianza y de una lista de palabras clave. Hace uso del buscador Google.
`string ObtenerHtmlSinTags (HtmlAgilityPack.HtmlNode nodoHtml)`	Remueve los *tags* de un documento HTML.
`void BuscarClavesPuntuar (string textoHtml, ArrayList clavesP, Pagina nuevaPagina)`	Ejecuta el algoritmo de puntuaciones para obtener el mejor párrafo de una determinada página web.

Tanto los métodos que el servicio SOAP publica, así como los métodos de apoyo, manipulan la estructura jerárquica de clases empleada para guardar el trabajo de los distintos usuarios que emplean el servicio. El diagrama de clases empleadas para este fin se presenta en la Figura 3.23.

3.1.1.8.3 Clase Datos

La clase `Datos` modela de manera general el trabajo realizado por varios usuarios empleando el servicio SOAP.

Los atributos de esta clase son:

- **`direccionIPBaseDatos:`** Atributo de tipo `string`. Corresponde a la dirección IP empleada para la conexión con SQL Server. El valor de este atributo puede obtenerse o establecerse con la propiedad[65] `DireccionIPBaseDatos`.

[65] Una propiedad es un miembro que permite leer, escribir o calcular los valores de los atributos con visibilidad *private* de una clase.

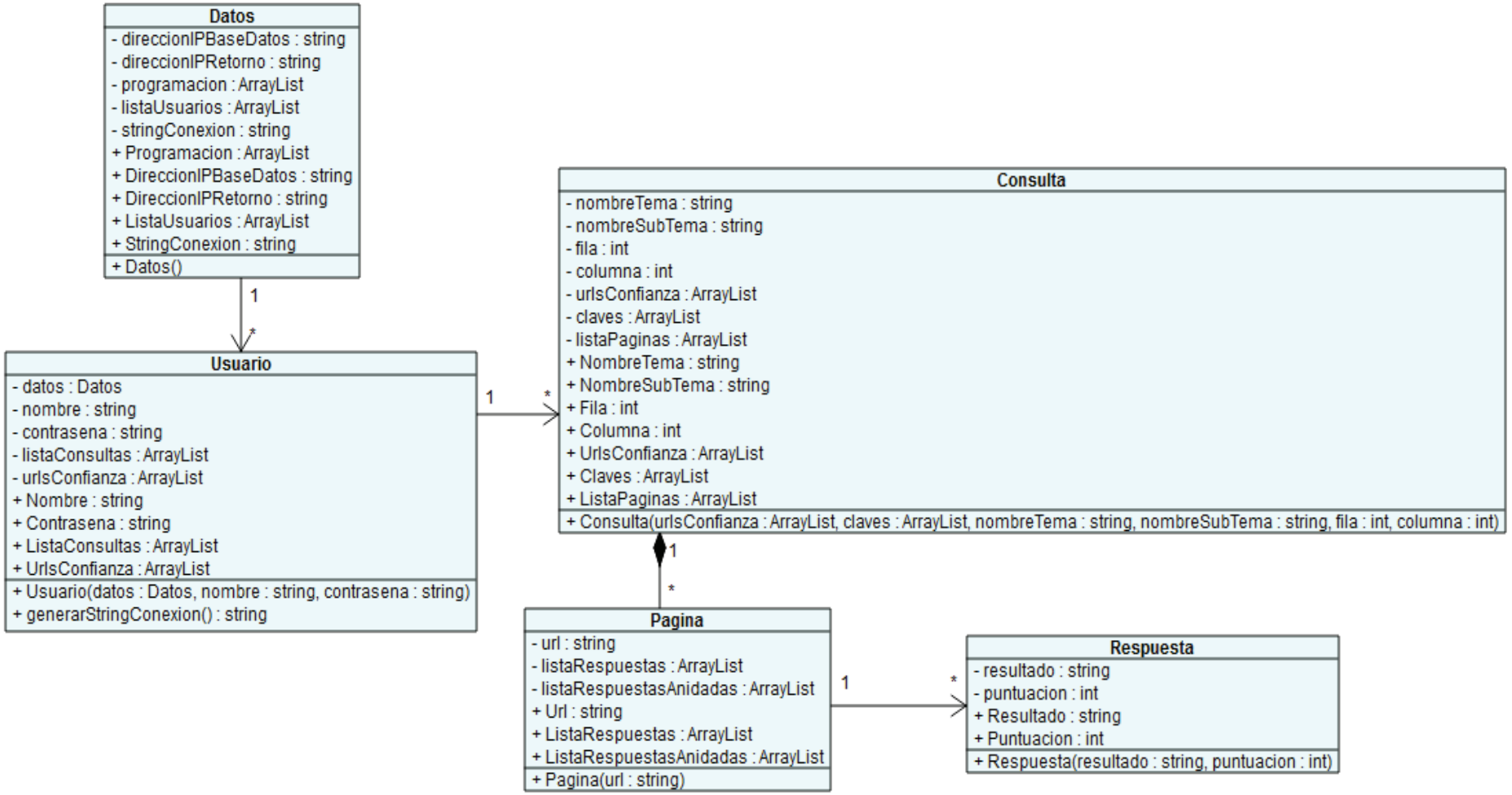


Figura 3.23 Diagrama de clases del servicio SOAP

- **direccionIPRetorno:** Atributo de tipo `string`. Corresponde a la dirección IP empleada para la conexión con el servidor de retorno (servicio RESTful). El valor de este atributo puede obtenerse o establecerse con la propiedad `DireccionIPRetorno`.
- **programacion:** Atributo de tipo `ArrayList`. Corresponde a una colección con los horarios de búsqueda periódica establecidos por el usuario administrador. El valor de este atributo puede obtenerse o establecerse con la propiedad `Programacion`.
- **listaUsuarios:** Atributo de tipo `ArrayList`. Consiste en una colección con referencias a objetos de la clase `Usuario`. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaUsuarios`.
- **stringConexion:** Atributo de tipo `string`. Corresponde a la cadena de conexión del usuario *adminNCL* para poder emplear las bases de datos. El valor de este atributo puede obtenerse o establecerse con la propiedad `StringConexion`.

El constructor de esta clase denominado `Datos()` no tiene parámetros.

3.1.1.8.4 Clase Usuario

La clase `Usuario` modela un usuario que emplea el servicio SOAP para hacer uso de las funciones de búsqueda, almacenamiento y procesamiento de información.

Los atributos de esta clase son:

- **datos:** Referencia al objeto de la clase `Datos` al cual el objeto de la clase `Usuario` pertenece.
- **nombre:** Atributo de tipo `string`. Corresponde al `ID` del usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Nombre`.
- **contrasena:** Atributo de tipo `string`. Corresponde a la contraseña del usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Contrasena`.

- **listaConsultas:** Atributo de tipo `ArrayList`. Consiste en una colección con referencias a objetos de la clase `Consulta`. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaConsultas`.
- **urlsConfianza:** Atributo de tipo `ArrayList`. Consiste en una colección con los URL de los sitios web de confianza ingresados por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `UrlsConfianza`.

El constructor de esta clase denominado `Usuario()` acepta como parámetros un objeto de la clase `Datos`, la cadena de texto correspondiente al `ID` de usuario y la cadena de texto correspondiente a la contraseña del usuario. El método `generarStringConexion()` retorna la cadena de conexión que se asigna al atributo `stringConexion` de la clase `Datos` para la conexión con SQL Server.

3.1.1.8.5 Clase Consulta

La clase `Consulta` modela la búsqueda de información que realiza un usuario en uno o varios sitios web de confianza para obtener la información a almacenarse en una celda de una tabla de una determinada base de datos.

Los atributos de esta clase son:

- **nombreTema:** Atributo de tipo `string`. Comprende el nombre de la base de datos para la cual se consulta la información. El valor de este atributo puede obtenerse o establecerse con la propiedad `NombreTema`.
- **nombreSubTema:** Atributo de tipo `string`. Comprende el nombre de una tabla de la base de datos para la cual se consulta la información. El valor de este atributo puede obtenerse o establecerse con la propiedad `NombreSubTema`.
- **fila:** Atributo de tipo `int`. Comprende el número de fila de la tabla para la cual se consulta la información. El valor de este atributo puede obtenerse o establecerse con la propiedad `Fila`.

- **columna:** Atributo de tipo `int`. Comprende el número de columna de la tabla para la cual se consulta la información. El valor de este atributo puede obtenerse o establecerse con la propiedad `Columna`.
- **urlsConfianza:** Atributo de tipo `ArrayList`. Comprende un listado de los URL de los sitios web de confianza ingresados por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `UrlsConfianza`.
- **claves:** Atributo de tipo `ArrayList`. Comprende un listado con las palabras clave ingresadas por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Claves`.
- **listaPaginas:** Atributo de tipo `ArrayList`. Corresponde a una colección con referencias a objetos de la clase `Pagina`. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaPaginas`.

El constructor de esta clase denominado `Consulta()` tiene como parámetros un `ArrayList` con los URL de los sitios web de confianza, un `ArrayList` con las palabras clave necesarias para la búsqueda, dos cadenas de texto correspondientes al nombre de la base de datos (nombre del tema) y al nombre de la tabla de la base de datos (nombre del subtema); y dos enteros correspondientes al número de fila y de columna a partir de los cuales se puede determinar la celda de la tabla para la cual se consulta la información.

3.1.1.8.6 Clase Pagina

La clase `Pagina` modela una página web con posible información de utilidad para el usuario, la cual es parte de un sitio web de confianza especificado por el usuario.

Los atributos de esta clase son:

- **url:** Atributo de tipo `string`. Corresponde al URL de la página web. El valor de este atributo puede obtenerse o establecerse con la propiedad `Url`.

- **`listaRespuestas:`** Atributo de tipo `ArrayList`. Consiste en una colección con referencias a objetos de la clase `Respuesta` correspondientes a las líneas de información contenidas en la página web. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaRespuestas`.
- **`listaRespuestasAnidadas:`** Atributo de tipo `ArrayList`. Consiste en una colección con referencias a objetos de la clase `Respuesta` correspondientes a los párrafos conformados por distintas líneas. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaRespuestasAnidadas`.

El constructor de esta clase denominado `Pagina()` tiene como parámetro una cadena de texto correspondiente al URL de la página web.

3.1.1.8.7 Clase Respuesta

La clase `Respuesta` modela una línea de texto o un párrafo (conjunto de líneas de texto) contenido en una página web.

Los atributos de esta clase son:

- **`resultado:`** Atributo de tipo `string`. Consiste en una cadena de texto correspondiente a una línea o un párrafo. El valor de este atributo puede obtenerse o establecerse con la propiedad `Resultado`.
- **`puntuacion:`** Atributo de tipo `int`. Corresponde a la puntuación obtenida por la línea o por el párrafo mediante el algoritmo de puntuaciones. El valor de este atributo puede obtenerse o establecerse con la propiedad `Puntuacion`.

El constructor de esta clase denominado `Respuesta()` tiene como parámetros una cadena de texto correspondiente a la línea o párrafo de la página web y un entero con la puntuación que la línea o párrafo ha obtenido mediante el algoritmo de puntuaciones.

3.1.2 SERVICIO RESTful [21]

El servicio RESTful será consumido por el STB del televidente con el fin de obtener información a través del canal de retorno. La razón principal para emplear el servicio RESTful es que la arquitectura REST permite retornar datos no tan solo en formato XML sino también en otros formatos [21], lo que se ha aprovechado para hacer que el servicio RESTful implementado retorne información como texto plano.

El evitar la envoltura del mensaje SOAP junto con el retorno de información en texto plano simplifica notablemente la programación del *script* Lua a ser ejecutado en el STB.

3.1.2.1 Consulta a la base de datos

El *script* Lua realiza una petición al servicio RESTful con el fin de obtener la información almacenada en una determinada celda de una tabla de una base de datos. Para ello, el servicio RESTful consulta la base de datos con el fin de obtener la tabla y a partir de ella la información solicitada como una cadena de texto.

El procedimiento empleado para consultar la base de datos es el mismo que el empleado por el servicio SOAP y la sentencia T-SQL corresponde a la detallada en la sección 3.1.1.5.8.

3.1.2.2 Establecimiento del formato de datos [2], [9], [13], [20], [21], [24]

Por defecto, un proyecto creado con Visual Studio 2012 permite que la información retornada por un método de un servicio RESTful sea en formato XML o en formato JSON.

Cada método declarado en el contrato de un servicio RESTful debe llevar el atributo `WebGet`, el cual asocia el método a un único URL que será accedido mediante una operación HTTP GET [2]. Este atributo tiene dos propiedades:

- **`UriTemplate:`** Especifica el formato del URI (*Uniform Resource Identifier*) para invocar el método [2].
- **`ResponseFormat:`** Especifica el formato de datos de la respuesta del método [20].

El formato de datos deseado puede ser elegido si dentro del atributo `WebGet` se asigna a la propiedad `ResponseFormat` la opción XML o JSON, de la enumeración `WebMessageFormat`, como se muestra en la Figura 3.24. Como se aprecia en la figura, el *intellisense* de Visual Studio al acceder a las opciones de la enumeración `WebMessageFormat` despliega tan solo los dos formatos de datos por defecto. Si no se especifica el formato de datos deseado con la propiedad `ResponseFormat`, el servicio retornará la información en formato XML.

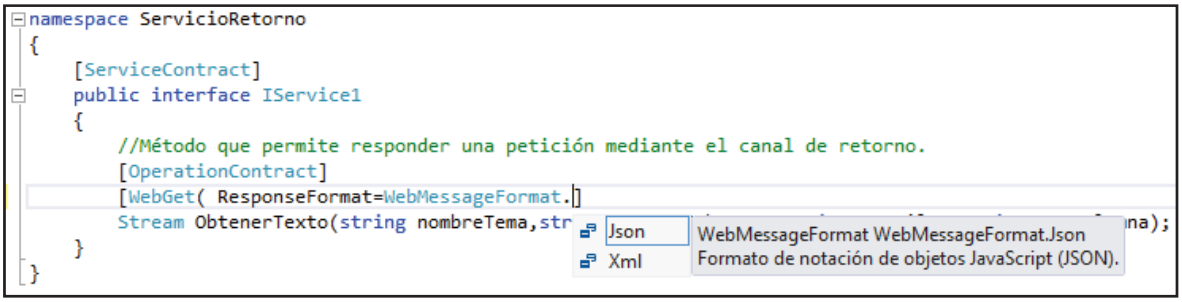


Figura 3.24 Formato de datos por defecto de un servicio RESTful

Sin embargo, una de las ventajas de los servicios RESTful es la posibilidad de retornar datos en otros formatos como HTML, texto plano e inclusive archivos multimedia como imágenes, documentos y videos [21], lo cual se puede hacer en Visual Studio 2012 cambiando el formato de datos que maneja el contexto actual del servicio web en tiempo de ejecución y con el manejo de *streams* [24].

Si se desea que un método de un servicio RESTful retorne la información en un determinado formato, el valor de retorno de dicho dato debe ser un objeto de la clase `Stream`. Dentro del bloque de implementación del método se puede obtener el contexto actual del servicio web mediante la propiedad `Current` de la clase `WebOperationContext`, y el formato de salida mediante la propiedad `OutgoingResponse` del contexto obtenido. La clase `WebOperationContext` puede ser empleada incorporando el espacio de nombres

`System.ServiceModel.Web`. El resultado es la obtención de un objeto de la clase `OutgoingWebResponseContext`, cuya propiedad `ContentType` consiste en una cadena de texto que determina el formato de datos del *stream* a ser retornado por el método. El Código 3.16 presenta las líneas necesarias para cambiar el formato de datos del contexto actual a texto plano, con lo cual se logra que el servicio RESTful implementado retorne la información en texto plano.

```
1:  OutgoingWebResponseContext contexto = WebOperationContext.Current.OutgoingResponse;
2:  contexto.ContentType = "text/plain";
```

Código 3.16 Código para cambiar el formato de datos de un servicio RESTful a texto plano

La Tabla 3.7 presenta algunos ejemplos de cadenas de texto que pueden asignarse a la propiedad `ContentType` de un objeto de la clase `OutgoingWebResponseContext`. La tabla completa que indica las cadenas de texto correspondientes a distintos formatos puede encontrase en [13].

Tabla 3.7 Cadenas de texto para diferentes formatos de datos [13]

Cadena de texto	Formato de dato que representa
text/plain	Texto plano
text/html	HTML
application/pdf	PDF
application/x-latex	LATEX
image/jpeg	JPG
image/png	PNG
image/gif	GIF
audio/wav	WAV
video/avi	AVI
video/mpeg	MPEG

Para formar el objeto de tipo `Stream` que retorna el método, es necesario crear un objeto de la clase `MemoryStream` [24]. La clase `MemoryStream` puede ser empleada incorporando el espacio de nombres `System.IO`. El constructor de esta clase recibe como argumento una colección de *bytes* correspondientes al mensaje. La manera en que la colección de *bytes* es obtenida depende del tipo de

información que se desee retornar, como por ejemplo una imagen, un video, un archivo o en este caso texto plano.

En particular, la colección de *bytes* correspondiente al texto puede ser obtenida mediante la clase `Encoding`, la cual codifica el texto en un determinado formato indicado por el usuario y retorna la colección de *bytes* correspondientes al texto codificado en dicho formato. Para la comunicación con el STB del televidente es necesario que el *stream* de *bytes* del texto correspondan a la codificación ASCII normal[66]. Mediante la propiedad `ASCII` de la clase abstracta `Encoding` se puede invocar al método `GetBytes()` que retornará la colección de *bytes* con codificación ASCII normal correspondiente al texto que se pasa como argumento [9].

Un método alternativo para que los *bytes* correspondan al texto con codificación ASCII normal es emplear la clase `ASCIIEncoding` que hereda de la clase `Encoding` [24]. Para indicar una codificación con ASCII normal se puede acceder al atributo `Default` de la clase `ASCIIEncoding`, para luego invocar al método `GetBytes()` que, al igual que el caso anterior, recibe como argumento una cadena de texto y retorna la colección de *bytes* correspondiente al texto con codificación ASCII normal. El objeto de tipo `MemoryStream` obtenido se retorna mediante la sentencia `return`, como se presenta en el Código 3.17[67].

```
1:      MemoryStream ms = new MemoryStream(ASCIIEncoding.Default.GetBytes(resultado));
2:      return ms;
```

Código 3.17 Creación y retorno de un objeto `MemoryStream`

3.1.2.3 Modificación del archivo Web.config [20], [21], [23]

El archivo Web.config es un archivo en formato XML que determina las normas bajo las cuales opera el servicio web WCF [23]. Si el servicio WCF emplea SOAP

[66] ASCII normal emplea siete bits de información y un bit de paridad para codificar un caracter.
[67] La clase `MemoryStream` hereda de la clase `Stream`, por lo que aunque el método especifique un objeto de la clase `Stream` como valor de retorno, se puede retornar un objeto de la clase `MemoryStream`. La única razón por la cual se emplea la clase `MemoryStream` en lugar de la clase `Stream` es para poder almacenar temporalmente en la memoria RAM el *stream* obtenido en lugar de hacer uso de un archivo.

normalmente este archivo no requiere ninguna modificación [20]. Sin embargo, para que el servicio WCF emplee la arquitectura REST en lugar del protocolo SOAP, el archivo de configuración Web.config debe ser modificado. El Código 3.18 presenta parte del archivo Web.config con las modificaciones necesarias para poder emplear la arquitectura REST.

```
1  <system.serviceModel>
2     <behaviors>
3        <serviceBehaviors>
4           <behavior>
5              <!-- To avoid disclosing metadata information, set the
6                 value below to false and remove the metadata
7                 endpoint above before deployment -->
8              <serviceMetadata httpGetEnabled="true"/>
9              <!-- To receive exception details in faults for debugging
10                purposes, set the value below to true.  Set to false
11                before deployment to avoid disclosing exception
12                information -->
13             <serviceDebug includeExceptionDetailInFaults="false"/>
14          </behavior>
15       </serviceBehaviors>
16       <endpointBehaviors>
17          <behavior>
18             <webHttp/>
19          </behavior>
20       </endpointBehaviors>
21    </behaviors>
22    <protocolMapping>
23       <add scheme="http" binding="webHttpBinding"/>
24    </protocolMapping>
25    <serviceHostingEnvironment multipleSiteBindingsEnabled="true"/>
26 </system.serviceModel>
```

Código 3.18 Parte del archivo Web.config modificado para emplear la arquitectura REST [21]

El elemento `endPointBehaviors` (líneas 16 a 20) indican que el servicio web emplea la arquitectura REST. El elemento `webHttp` anidado dentro del elemento `behavior` especifica que los clientes se comunicarán con el servicio web empleando el protocolo HTTP mediante el mecanismo de petición/respuesta [21].

El elemento `protocolMapping` (líneas 22 a 24) cambia el protocolo de comunicación empleado por el servicio web, el cual por defecto es SOAP, al protocolo `WebHttpBinding`, el cual es usado en la arquitectura REST para responder a peticiones HTTP [21].

3.1.2.4 Caso de uso

La Figura 3.25 describe el único caso de uso del servicio RESTful. El actor es el televidente, quien hace uso del servicio para obtener, a través del canal de retorno, la información almacenada en una determinada celda de una tabla de una base de datos.

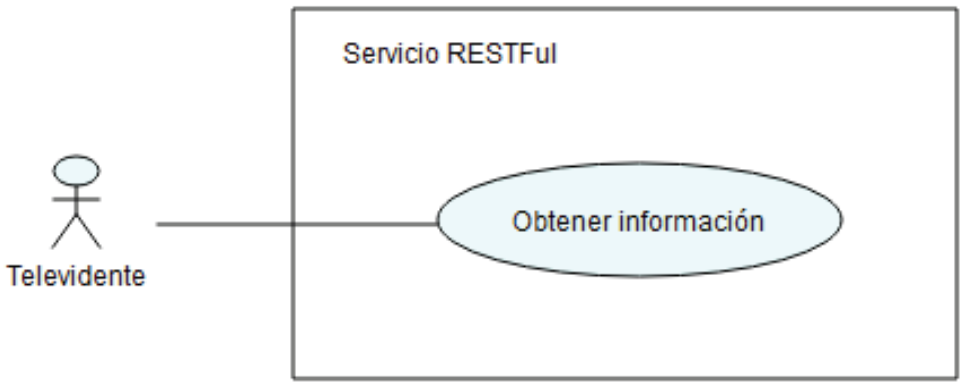


Figura 3.25 Diagrama de caso de uso del servicio RESTful

El servicio RESTful retorna dicha información en texto plano, de manera que el *script* Lua que se ejecuta en el STB del televidente pueda receptar la información y adecuar el texto para presentarlo en la pantalla del televisor.

3.1.2.5 Implementación

Para el servicio RESTful se ha implementado:

- Una interfaz, denominada `IServicioRetorno`, correspondiente al contrato del servicio RESTful.
- Una clase, denominada `ServicioRetorno`, que implementa el único método declarado en la interfaz `IServicioRetorno`.

3.1.2.5.1 Interfaz IServicioRetorno [2], [23], [25]

Al igual que en el servicio SOAP, la interfaz `IServicioRetorno` del servicio RESTful corresponde al contrato del servicio (componente *Contract*) que determina los métodos que publica el servicio.

De igual manera que en el caso de SOAP, antes de la firma de la interfaz se debe establecer el atributo `[ServiceContract]`, y antes de la firma de cada método el atributo `[OperationContract]` para indicar que el método es publicado por el servicio RESTful [23].

Adicionalmente, dado que el servicio RESTful maneja URL para la invocación de métodos [25], en esta interfaz se deberá establecer antes de la firma de cada método el atributo `WebGet` en el que, a través de la propiedad `UriTemplate`, se indique el URI con el cual cada cliente puede acceder al método que publica el servicio RESTful [2].

Dado que el servicio RESTful es únicamente utilizado para responder a peticiones del STB del televidente a través del canal de retorno, el servicio publica un único método llamado `ObtenerTexto()` cuya firma se declara en la interfaz `IServicioRetorno` como se presenta en el Código 3.19.

```
[ServiceContract]
public interface IService1
{
    //Método que permite responder una petición mediante el canal de retorno.
    [OperationContract]
    [WebGet(UriTemplate = "/obtenertexto/{nombreTema}/{nombreSubTema}/{numFila}/{numColumna}")]
    Stream ObtenerTexto(string nombreTema,string nombreSubTema, string numFila, string numColumna);
}
```

Código 3.19 Declaración del método del servicio RESTful

El URI establecido comprende un identificador que hace referencia a la funcionalidad del método para poder reconocerlo (obtenertexto), seguido de cada uno de los parámetros del método, indicados entre llaves, los cuales son:

- **`nombreTema:`** Corresponde al nombre de una base de datos.
- **`nombreSubTema:`** Corresponde al nombre de una tabla de la base de datos.
- **`numFila:`** Número de fila en la que se encuentra la información en la tabla.
- **`numColumna:`** Número de columna en la que se encuentra la información en la tabla.

3.1.2.5.2 Clase ServicioRetorno

La clase `ServicioRetorno` corresponde a la clase encargada de implementar el método cuya firma se declaró en la interfaz `IServicioRetorno`. Por lo tanto, este método se encargará de obtener de la base de datos indicada en su parámetro `nombreTema` la tabla indicada en el parámetro `nombreSubTema`, y a partir de ella la información almacenada en la celda indicada a través de los parámetros `numFila` y `numColumna`. Una vez obtenida la información, el método cambiará el formato de datos del contexto actual del servicio a texto plano y retornará la información como un *stream* al STB del televidente.

La Figura 3.26 presenta el diagrama de la clase `ServicioRetorno` heredando de la interfaz `IServicioRetorno` para implementar el método `ObtenerTexto()`.

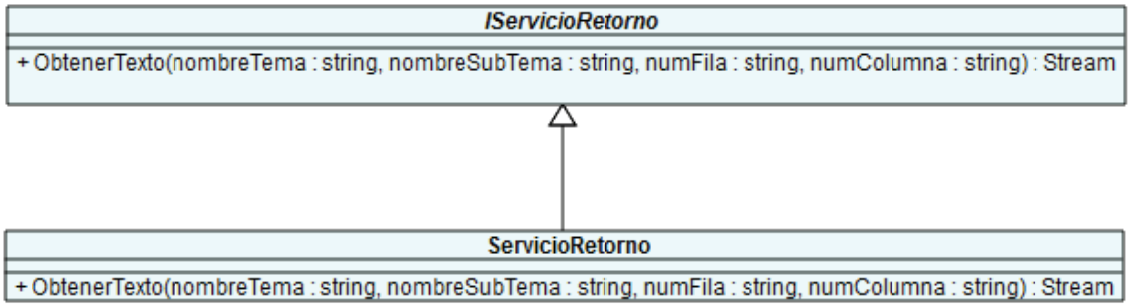


Figura 3.26 Diagrama de la clase `ServicioRetorno` e interfaz `IServicioRetorno` del servicio RESTful

3.1.3 ALOJAMIENTO EN INTERNET INFORMATION SERVICES (IIS)

Con el fin de que el servicio WCF esté constantemente operando, es necesario alojar tanto su parte SOAP como REST en servidores web. Windows brinda el servidor IIS (*Internet Information Services*) que permite convertir a una PC en un servidor web para una Intranet o Internet.

Para activar IIS en Windows 7[68], desde la opción *"Activar o Desactivar las características de Windows"* de la opción *"Programas"* del Panel de Control, se

[68] Windows 7 no es un sistema operativo servidor, por lo que para entornos de producción sería conveniente emplear Windows Server versión 2008 o posterior.

debe activar la casilla *"Internet Information Services"* con sus dos opciones *"Herramientas de administración web"* y *"Servicio World Wide Web"*, las cuales permiten administrar e incoporar servicios web al servidor IIS, respectivamente; como se muestra en la Figura 3.27.

Figura 3.27 Activación del servidor IIS en Windows 7

Una vez hecho esto, desde el *Administrador de Internet Information Services* (*IIS*) se pueden añadir fácilmente los servicios SOAP y RESTful, seleccionando la opción *"Agregar aplicación* de un sitio web*"* e indicando la ruta de acceso física de la carpeta del proyecto correspondiente al servicio que se agrega. La Figura 3.28 presenta los servicios SOAP y RESTful agregados al servidor IIS.

La Figura 3.29 presenta la vista de contenido del servicio SOAP en IIS, en donde se puede apreciar los archivos del proyecto del servicio SOAP creados con Visual Studio.

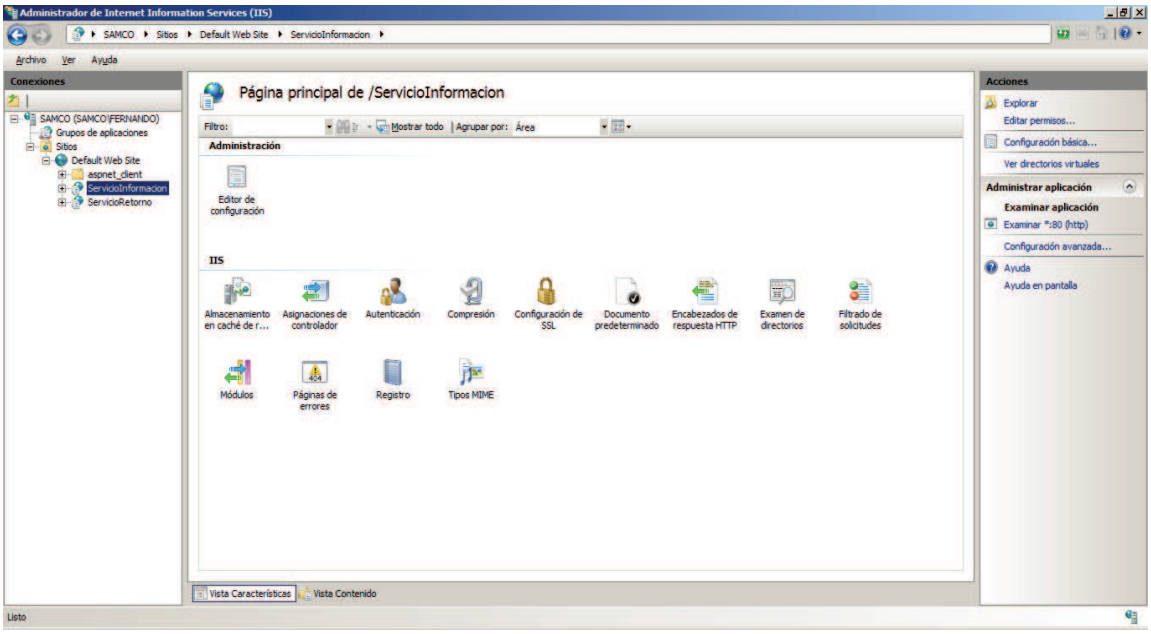


Figura 3.28 Servicios SOAP y RESTful agregados al servidor IIS

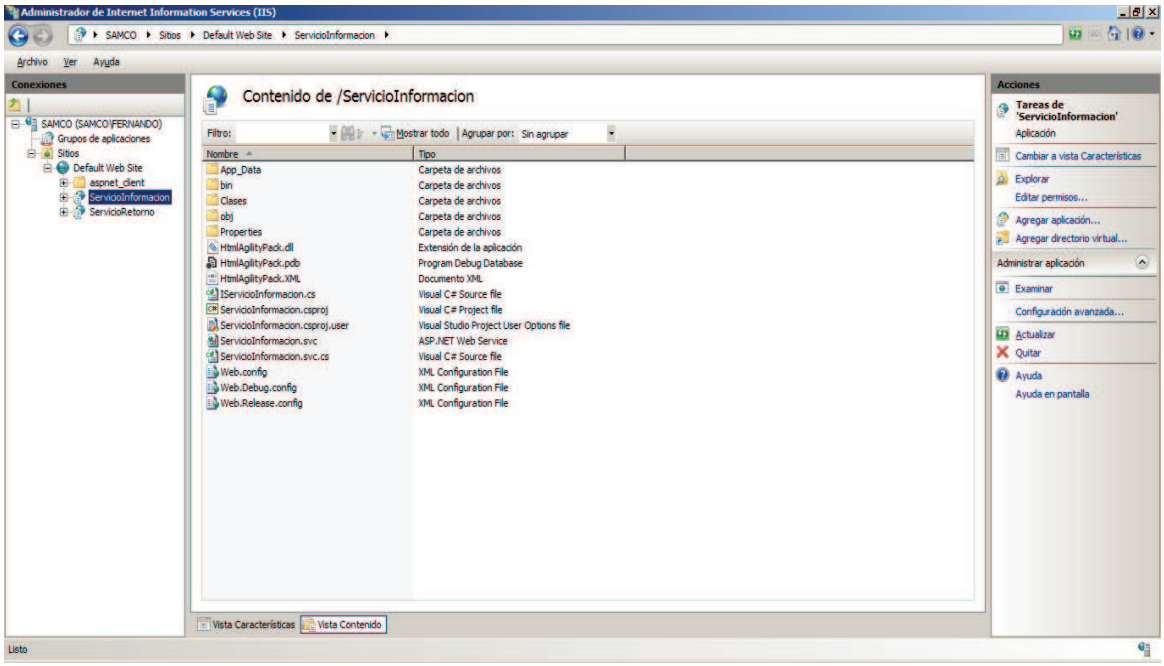


Figura 3.29 Vista de contenido del servicio SOAP en IIS

En la Figura 3.30 se puede apreciar la vista de contenido del servicio RESTful.

De esta manera, los servicios SOAP y RESTful quedan alojados para brindar una operación continua y pueden ser iniciados, detenidos o reiniciados desde el *Administrador de Internet Information Services* (*IIS*).

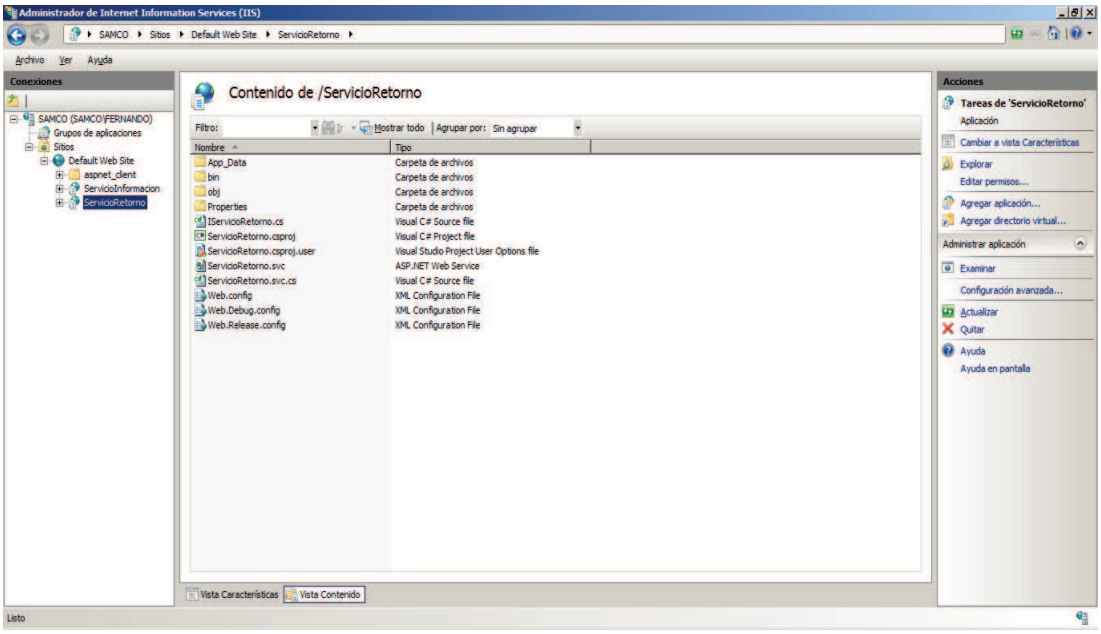


Figura 3.30 Vista de contenido del servicio RESTful en IIS

3.2 APLICACIÓN CLIENTE PARA CONSUMIR EL SERVICIO WEB WCF: "TEXTUAL DATA CREATOR"

La aplicación cliente para consumir el servicio web WCF consiste en una aplicación de escritorio desarrollada con Visual Studio 2012 empleando el lenguaje de programación C#, y ha sido nombrada Textual Data Creator. Esta aplicación brinda una interfaz gráfica amigable de manera que el usuario pueda hacer uso de la funcionalidad del servicio SOAP a través de la creación del *proxy*.

Por lo tanto, la aplicación permitirá al usuario, principalmente, hacer uso de las funciones de búsqueda en sitios web y de almacenamiento y procesamiento de información en las bases de datos.

3.2.1 FORMULARIOS DE PROPÓSITO GENERAL

La aplicación cliente consta de cinco formularios de propósito general, los cuales son parte de la interfaz gráfica de la aplicación.

Estos formularios son:

- `FrmSplash`
- `FrmAutenticar`

- FrmPrincipal
- FrmAyuda
- FrmAcercaDe

3.2.1.1 Formulario FrmSplash

El formulario FrmSplash se presenta en la Figura 3.31. Este formulario se presenta al usuario durante unos segundos cuando la aplicación es iniciada para indicar su nombre.

Figura 3.31 Formulario FrmSplash de la aplicación cliente

3.2.1.2 Formulario FrmAutenticar

El formulario FrmAutenticar se muestra en la Figura 3.32 y se presenta al usuario después del formulario FrmSplash. En este formulario el usuario debe ingresar su nombre de usuario (ID de usuario) y contraseña, a partir de los cuales el servicio SOAP desplegará las búsquedas en sitios web realizadas por el usuario anteriormente.

Si el usuario no ingresa las credenciales de autenticación correctas, la aplicación no le permitirá hacer uso del resto de formularios con el fin de emplear el servicio SOAP.

3.2.1.3 Formulario FrmPrincipal

El formulario FrmPrincipal se muestra en la Figura 3.33. Este formulario se presenta después de que el usuario ha ingresado correctamente sus credenciales

de autenticación y consiste en un formulario MDI (*Multiple Document Interface*) en el cual se abrirán otros formularios hijos.

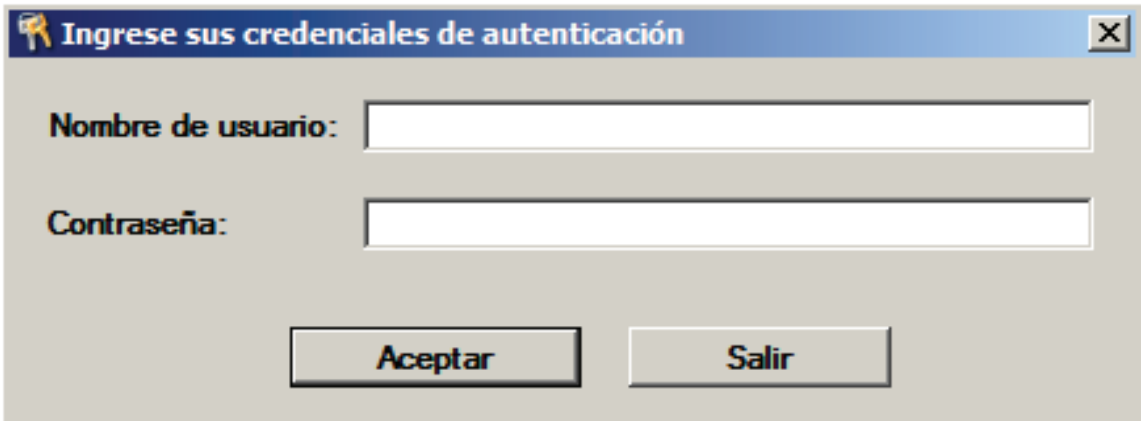


Figura 3.32 Formulario `FrmAutenticar` de la aplicación cliente

Figura 3.33 Formulario `FrmPrincipal` de la aplicación cliente

3.2.1.4 Formulario FrmAyuda

El `FrmAyuda` que se presenta en la Figura 3.34 consiste en un formulario en el que se detalla el funcionamiento de la aplicación cliente con el fin de ayudar al usuario a utilizar la aplicación.

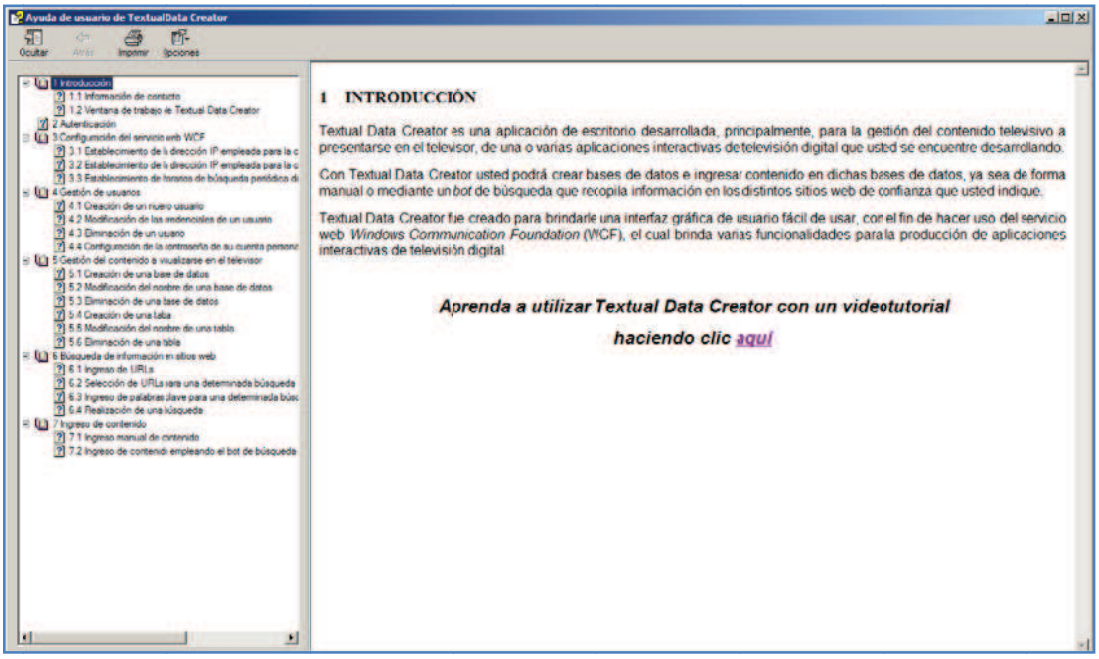


Figura 3.34 Formulario `FrmAyuda` de la aplicación cliente

3.2.1.5 Fomulario FrmAcercaDe

El formulario `FrmAcercaDe` se presenta en la Figura 3.35. Al abrir este formulario, el usuario puede visualizar información general de la aplicación cliente.

Figura 3.35 Formulario `FrmAcercaDe` de la aplicación cliente

3.2.2 FUNCIONALIDADES

Las funcionalidades de la aplicación cliente pueden ser resumidas en tres módulos:

- Módulo de configuración del servicio web
- Módulo de gestión de usuarios
- Módulo de ingreso de contenido a mostrarse en el televisor

3.2.2.1 Módulo de configuración del servicio web

El módulo de configuración del servicio web provee la interfaz gráfica para que el usuario pueda obtener y establecer la dirección IP empleda para la conexión con SQL Server, así como la dirección IP empleada para la conexión con el servicio RESTful (servidor de retorno). Además permite establecer los horarios de programación de búsqueda periódica de información.

El Formulario `FrmConfigurarServicioWeb` es el único formulario del módulo de configuración del servicio web y se presenta en la Figura 3.36. Este formulario puede ser empleado únicamente por el usuario *adminNCL*, por lo que para el resto de usuarios no es visible.

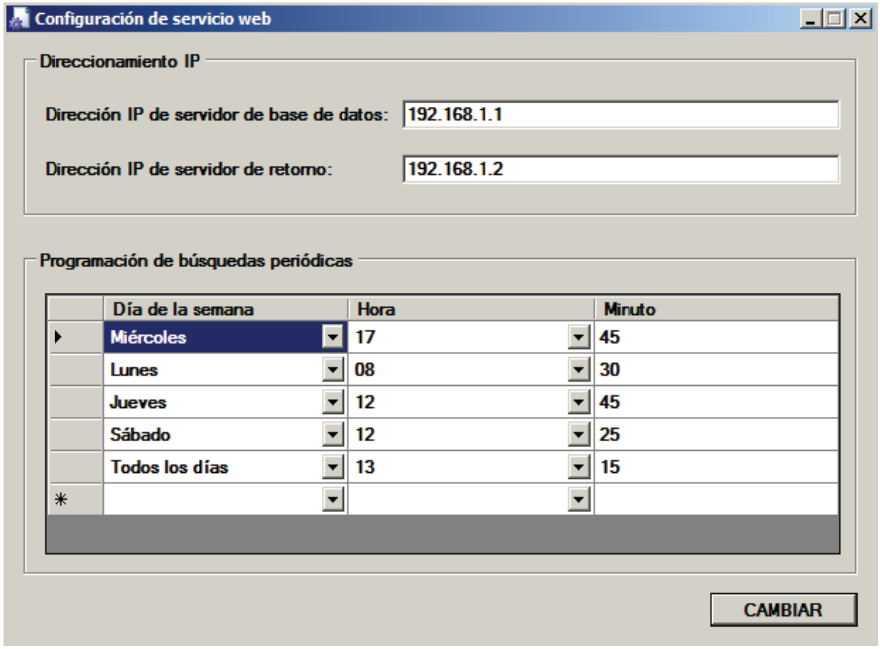


Figura 3.36 Fomulario `FrmConfigurarServicioWeb` de la aplicación cliente

Al abrir el formulario, se mostrarán las direcciones IP que están siendo actualmente empleadas para la conexión con SQL Server y con el servicio RESTful, así como también los horarios de búsqueda periódica de información que el usuario administrador ha programado.

3.2.2.2 Módulo de gestión de usuarios

El módulo de gestión de usuarios provee la interfaz gráfica necesaria para crear, eliminar y modificar la información de los distintos usuarios del sistema. Este módulo está conformado por dos formularios:

- `FrmUsuarios`
- `FrmMiCuenta`

3.2.2.2.1 Formulario FrmUsuarios

El formulario `FrmUsuarios` se presenta en la Figura 3.37. Este formulario puede ser usado únicamente por el usuario administrador (*adminNCL*), por lo que no es visible para el resto de usuarios.

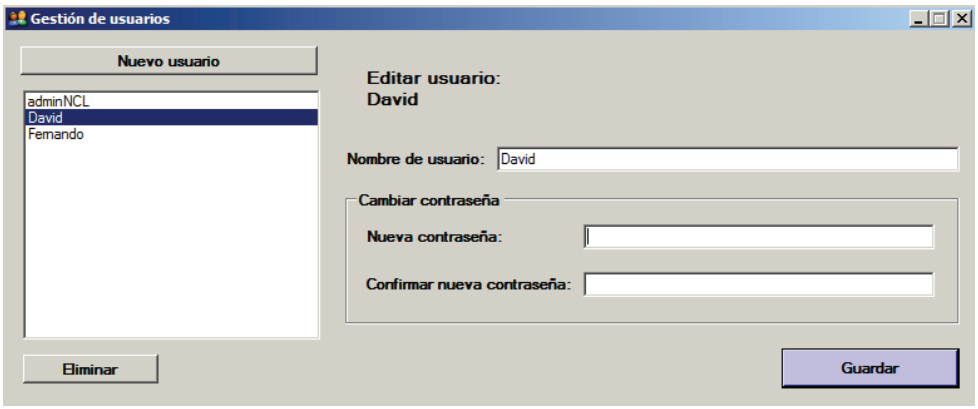


Figura 3.37 Formulario `FrmUsuarios`

Al abrir el formulario, se listarán en la parte izquierda los distintos usuarios del sistema, de manera que al seleccionar alguno, el usuario *adminNCL* pueda modificar las credenciales del usuario elegido si así lo requiere. De igual manera, a partir de este formulario el usuario administrador puede eliminar usuarios así como crear nuevos usuarios, como se muestra en la Figura 3.38.

3.2.2.2.2 Formulario FrmMiCuenta

El formulario `FrmMiCuenta` provee la interfaz gráfica necesaria para que cualquier usuario pueda cambiar su contraseña. Este formulario se presenta en la Figura 3.39.

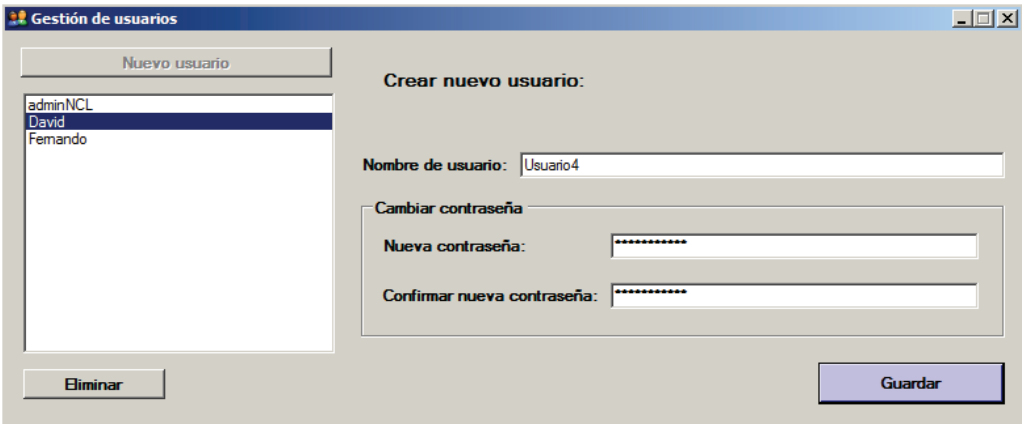


Figura 3.38 Creación de un nuevo usuario en el formulario `FrmUsuarios`

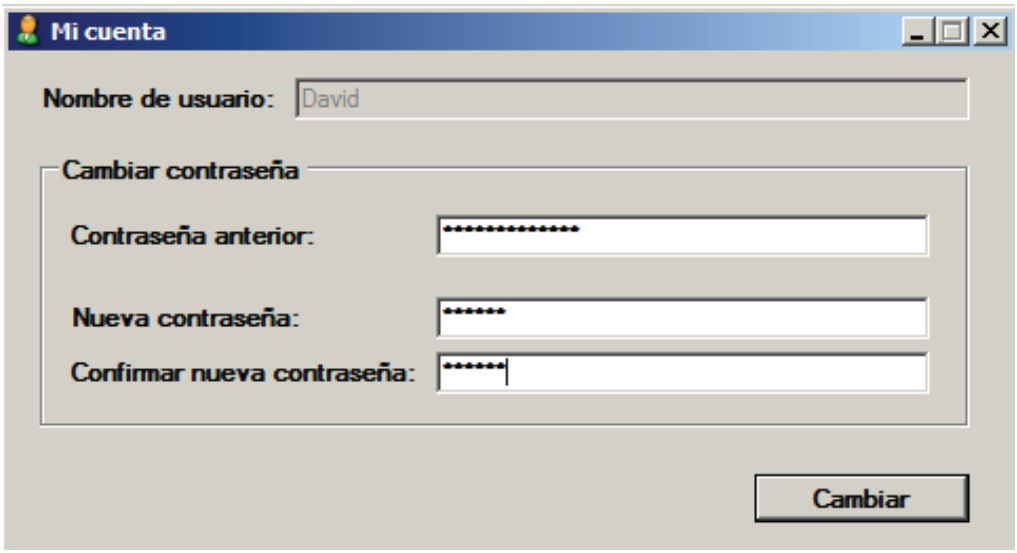


Figura 3.39 Formulario `FrmMiCuenta` de la aplicación cliente

3.2.2.3 Módulo de ingreso de contenido a mostrarse en el televisor

El módulo de ingreso de contenido a mostrarse en el televisor consta de tres formularios:

- `FrmUrls`
- `FrmIngreso`
- `FrmTemas`

Estos formularios pueden ser empleados por cualquier tipo de usuario con el fin de ingresar la información de cada una de las tablas que conforman las distintas

bases de datos, correspondientes a la distintas aplicaciones interactivas que se estén creando.

3.2.2.3.1 Formulario FrmUrls

El formulario `FrmUrls` provee la interfaz gráfica para que el usuario pueda ingresar el listado general de los URL de los sitios web de confianza que se empleará para las distintas búsquedas, como se presenta en la Figura 3.40.

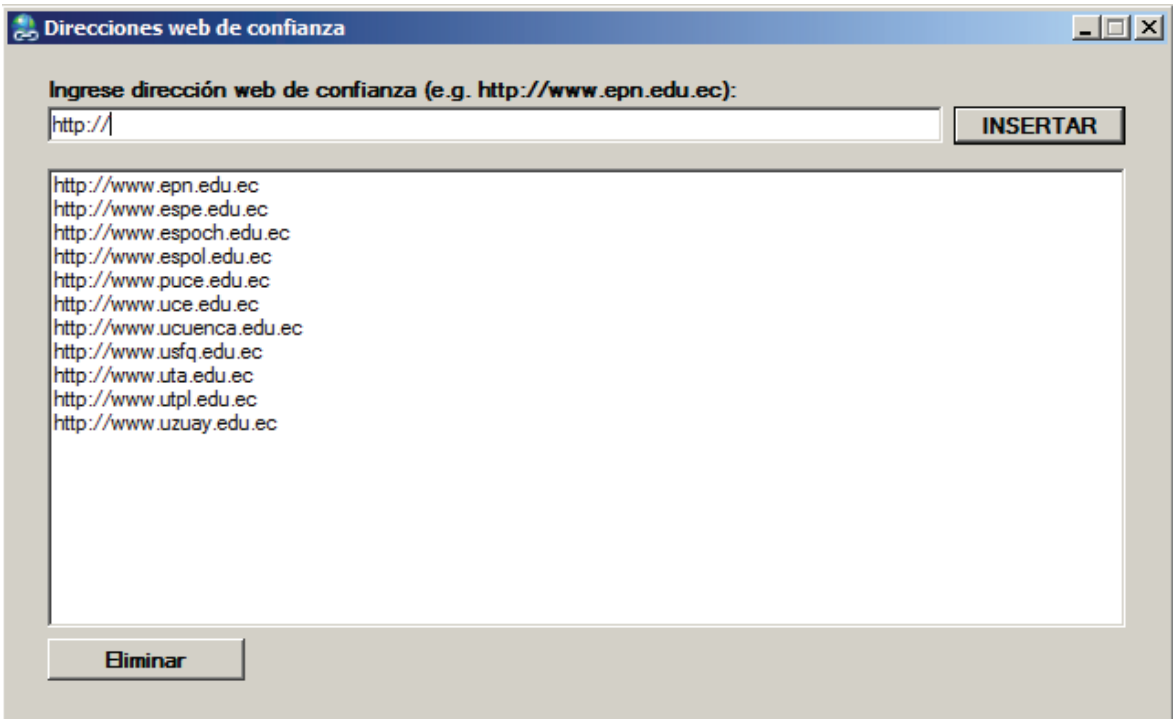


Figura 3.40 Formulario `FrmUrls` de la aplicación cliente

Al abrir este formulario, el listado de sitios web de confianza ingresado se mostrará al usuario. El usuario puede agregar más sitios web así como eliminarlos.

3.2.2.3.2 Formulario FrmIngreso

El formulario `FrmIngreso` se presenta en la Figura 3.41. Este formulario es un formulario de apoyo el cual brinda la interfaz gráfica para que el usuario pueda ingresar un dato específico. Así por ejemplo, cuando el usuario requiera ingresar el nombre de una base de datos que va a crear, el nombre de una tabla o una

palabra clave necesaria para una búsqueda, este formulario se presentará automáticamente para que pueda ingresar el dato.

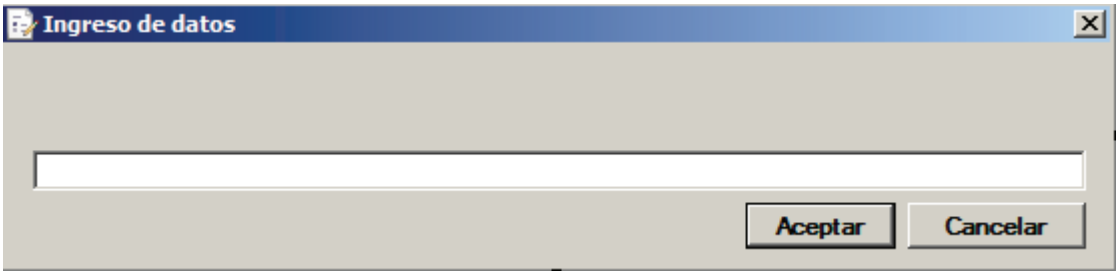


Figura 3.41 Formulario `FrmIngreso` de la aplicación cliente

El formulario desplegará un mensaje de petición que dependerá del tipo de dato para el cual se invoca al formulario. Así por ejemplo, la Figura 3.42 muestra el formulario que se presenta para que el usuario ingrese el nombre de un tema (base de datos) a crearse, mientras que la Figura 3.43 muestra el formulario que se presenta para que el usuario ingrese una palabra clave necesaria para realizar una búsqueda.

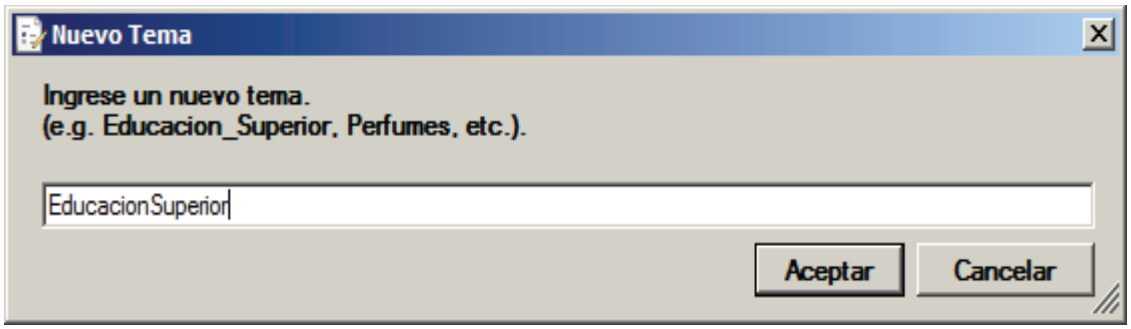


Figura 3.42 Formulario `FrmIngreso` pidiendo ingresar el nombre de un tema (base de datos) a crearse

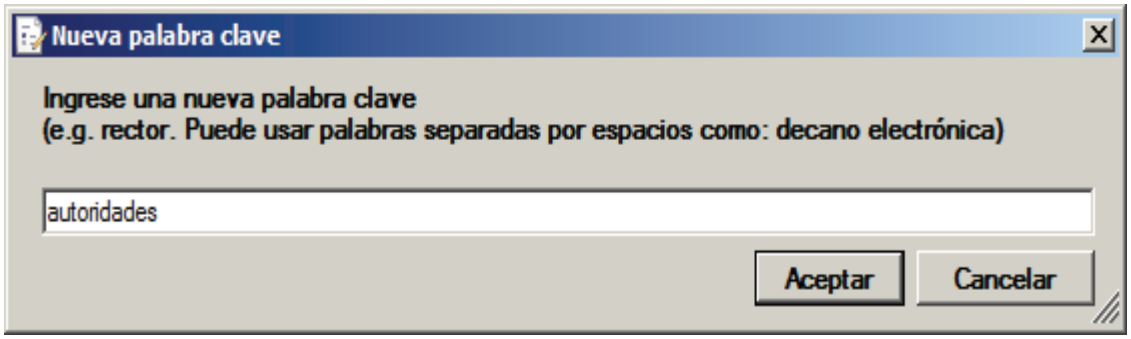


Figura 3.43 Formulario `FrmIngreso` pidiendo ingresar una palabra clave necesaria para realizar una búsqueda

3.2.2.3.3 Formulario FrmTemas

El formulario `FrmTemas` se presenta en la Figura 3.44. Este formulario provee la interfaz gráfica a través de la cual el usuario puede crear, cambiar o eliminar los temas (bases de datos) correspondientes a las distintas aplicaciones interactivas que se estén desarrollando; así como crear, cambiar o eliminar los subtemas (tablas de dichas bases de datos).

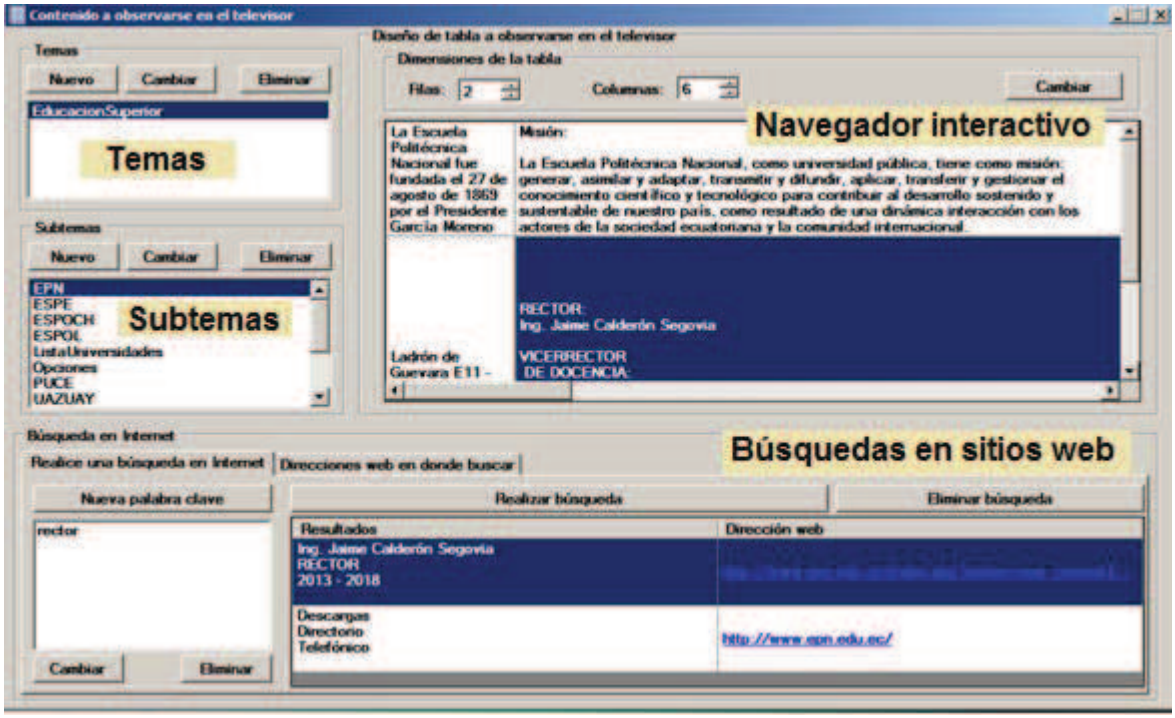


Figura 3.44 Formulario `FrmTemas` de la aplicación cliente

El formulario dispone de un navegador interactivo a través del cual el usuario puede movilizarse entre las distintas celdas de la tabla seleccionada. El usuario puede ingresar el contenido de la celda de forma manual o a través del *bot* de búsqueda.

En la parte inferior del formulario, el usuario cuenta con los recursos necesarios para poder realizar la búsqueda de información para la celda seleccionada con el navegador interactivo. El sitio o los sitios web seleccionados para realizar la búsqueda con el fin de obtener la información a almacenarse en la celda, pueden ser seleccionados en la pestaña *"Direcciones web en donde buscar"* que se presenta en la Figura 3.45.

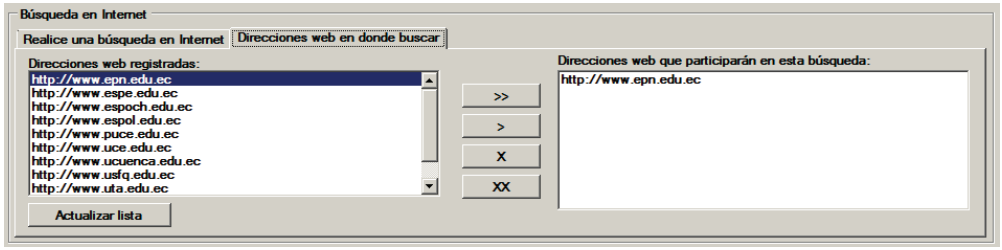


Figura 3.45 Pestaña *"Direcciones web donde buscar"*

Conforme el usuario se moviliza entre las distintas celdas con el navegador interactivo, los resultados de las búsquedas realizadas para cada celda se desplegarán en la parte inferior del formulario.

Al hacer doble clic sobre la pestaña *"Realice una búsqueda en Internet"*, los resultados de la búsqueda obtenidos para la celda se redimensionan para ocupar todo el formulario, como se presenta en la Figura 3.46. Al hacer nuevamente doble clic sobre la pestaña se regresa a la vista normal del formulario.

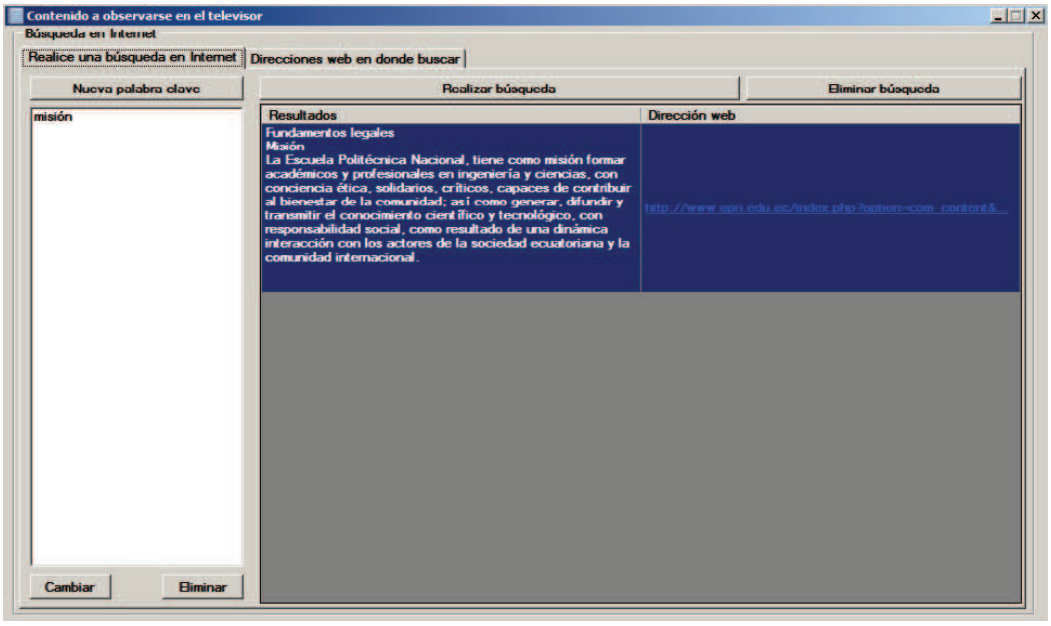


Figura 3.46 Resultado de búsqueda de información para una determinada celda redimensionado ocupando todo el formulario

De igual manera, cuando el usuario emplea el navegador interactivo, al hacer doble clic sobre éste, su tamaño se redimensiona para ocupar todo el formulario, brindando una mayor facilidad al usuario para el ingreso manual de datos, como se muestra en la Figura 3.47. Al hacer nuevamente doble clic sobre el navegador se regresa a la vista normal del formulario.

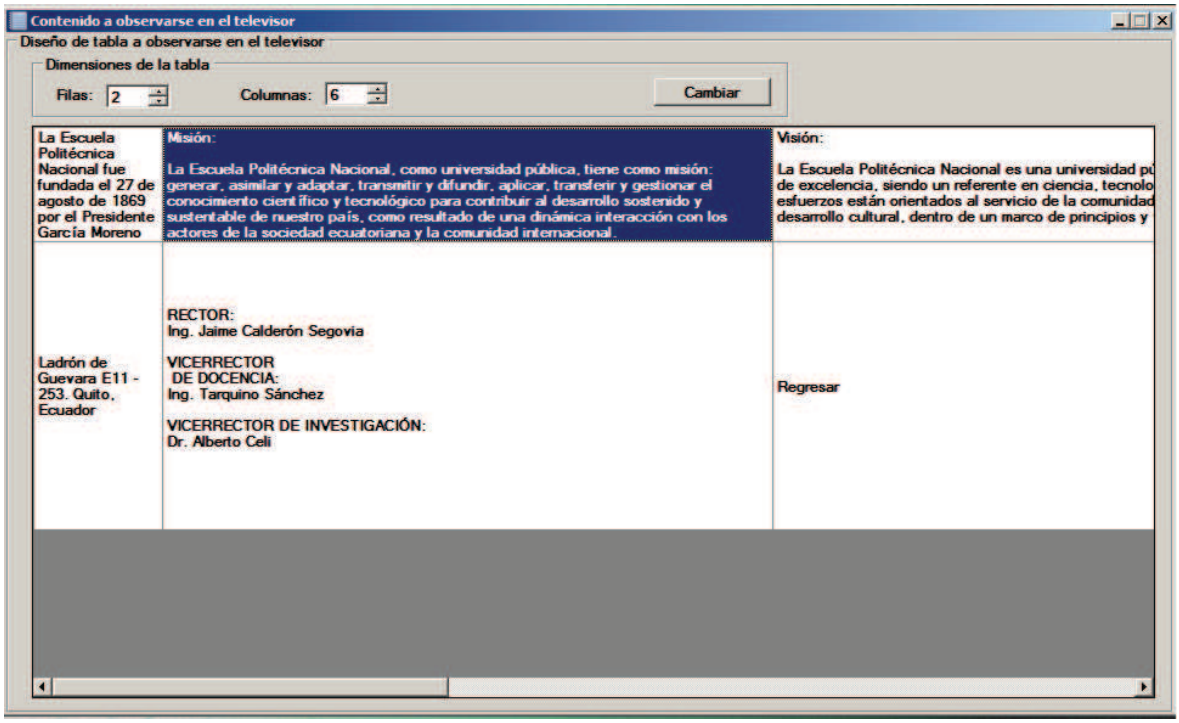


Figura 3.47 Navegador interactivo redimensionado ocupando todo el formulario

3.3 APLICACIÓN MIXER PARA GENERACIÓN DE DATOS NCL: "NCL-TEXTUAL DATA MIXER"

La aplicación Mixer consiste en una aplicación de escritorio desarrollada con Visual Studio 2012 empleando el lenguaje de programación C#, y ha sido nombrada NCL-Textual Data Mixer. El propósito de esta aplicación es asociar los menús desarrollados con el *plug-in* Menu Creator, así como nodos de contenido referentes a imágenes PNG o *scripts* Lua con el contenido de una base de datos.

Para ello, la aplicación Mixer trabaja haciendo uso de metadatos, los cuales deben ser declarados dentro del documento NCL a través de la etiqueta `<meta>`. La aplicación emplea tres tipos de metadatos cuya estructura se presenta en el Código 3.20.

```
METADATO 1: <meta name=id_contexto content=nombre_tabla>
METADATO 2: <meta name=id_media content=nombre_tabla/#fila/#columna>
METADATO 3: <meta name=id_media content=nombre_tabla/#fila/#columna/r>
```

Código 3.20 Tipos de metadatos

Estos metadatos son:

- **Metadato de tipo uno:** Asocia un menú desarrollado empleando el *plug-in* Menu Creator con el contenido almacenado en una tabla. La propiedad `name` de este metadato indica el `id` del nodo de contexto correspondiente al menú. La propiedad `content` del metadato indica el nombre de la tabla. Cada celda de la tabla se asociará con un elemento del menú, de manera que la aplicación Mixer genere para cada elemento del menú una imagen PNG transparente con el texto correspondiente.
- **Metadato de tipo dos:** Asocia un nodo de contenido referente a una imagen PNG con una celda de una tabla. La propiedad `name` del metadato indica el `id` del nodo de contenido. La propiedad `content` del metadato indica la celda mediante el nombre de la tabla y el número de fila y columna. La aplicación Mixer genera una imagen PNG con transparencia correspondiente al nodo de contenido declarado con el texto de la celda.
- **Metadato de tipo tres:** Este metadato es usado para traer información a través del canal de retorno. Asocia un nodo de contenido referente a un *script* Lua con una celda de una tabla. La propiedad `name` del metadato indica el `id` del nodo de contenido. La propiedad `content` del metadato indica la celda mediante el nombre de la tabla y el número de fila y columna; y culmina con una letra r (retorno) para diferenciar al metadato del metadato de tipo dos. La aplicación Mixer genera automáticamente el *script* Lua correspondiente al nodo de contenido declarado, de manera que al ejecutarse el *script* en el STB solicite al servicio RESTful la información a ser presentada al televidente.

La aplicación Mixer emplea el servicio SOAP para obtener la información almacenada en las bases de datos; así como para obtener la dirección IP del servidor con la cual se estructura el URL de petición del *script* Lua, necesario para traer información a través del canal de retorno.

La aplicación Mixer únicamente puede ser empleada por el usuario administrador (*adminNCL*). Al culminar el proceso de asociación, se obtiene la aplicación interactiva final que podrá ser ejecutada en el STB del televidente.

3.3.1 DISEÑO

3.3.1.1 Casos de uso

Para el análisis de los casos de uso de la aplicación Mixer se considera un único actor correspondiente al administrador (usuario *adminNCL*), el cual es el encargado de realizar el proceso de asociación de metadatos y obtener las aplicaciones interactivas finales a ser presentadas al televidente.

La Figura 3.48 detalla el diagrama de casos de uso general de la aplicación Mixer.

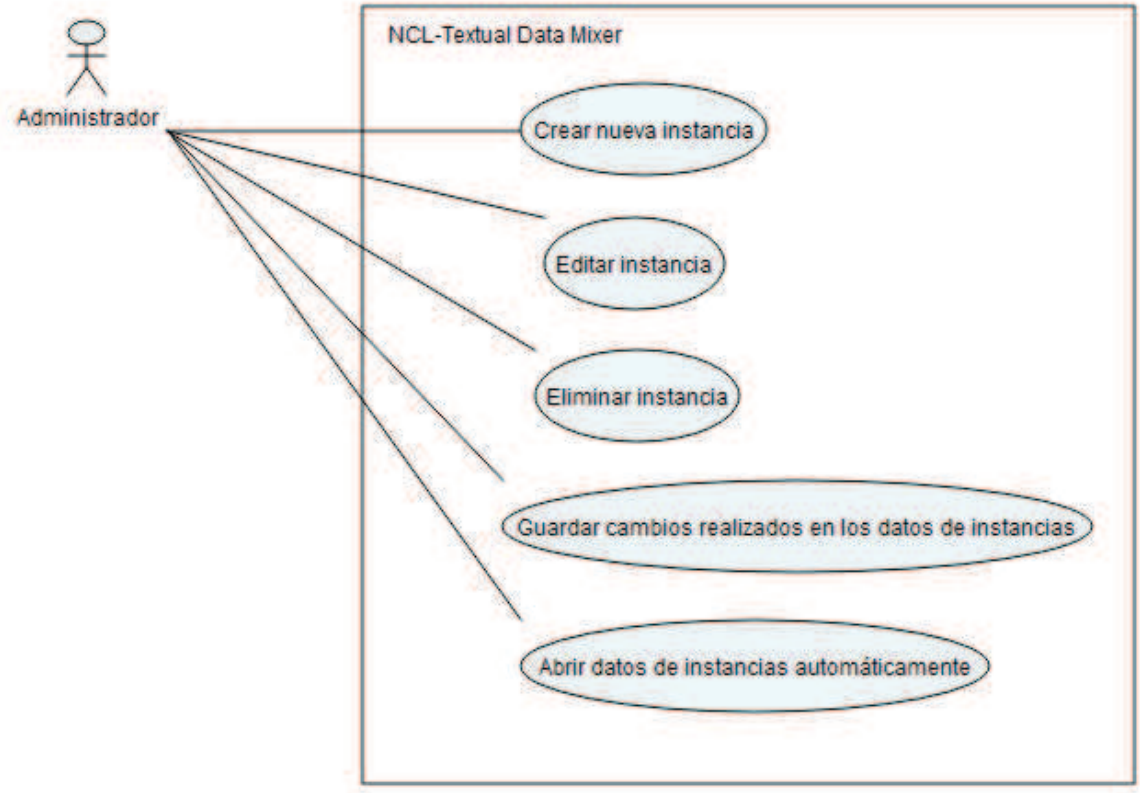


Figura 3.48 Diagrama de casos de uso general de la aplicación Mixer

El usuario administrador hace uso de la aplicación para:

- **Crear una nueva instancia:** Una instancia corresponde al proceso de asociación de datos que la aplicación realiza para una determinada aplicación interactiva que se está desarrollando. Al crear una nueva instancia, el usuario ordena a la aplicación llevar a cabo un nuevo proceso de asociación indicando parámetros como el nombre de la instancia, la ruta

hacia el documento NCL y la base de datos que se desea emplear para el proceso de asociación.

- **Editar una instancia:** Consiste en modificar los parámetros indicados por el usuario para una determinada instancia.
- **Eliminar una instancia:** Elimina una instancia anteriormente creada.
- **Guardar los cambios realizados en los datos de instancias:** Guarda los resultados del proceso de asociación con los cambios realizados por el usuario.
- **Abrir los datos de instancias automáticamente:** Abre los resultados del proceso de asociación, los cuales fueron anteriormente guardados.

3.3.1.1.1 Edición de una instancia

La Figura 3.49 detalla el diagrama de casos de uso de la opción para editar una instancia que ofrece la aplicación Mixer.

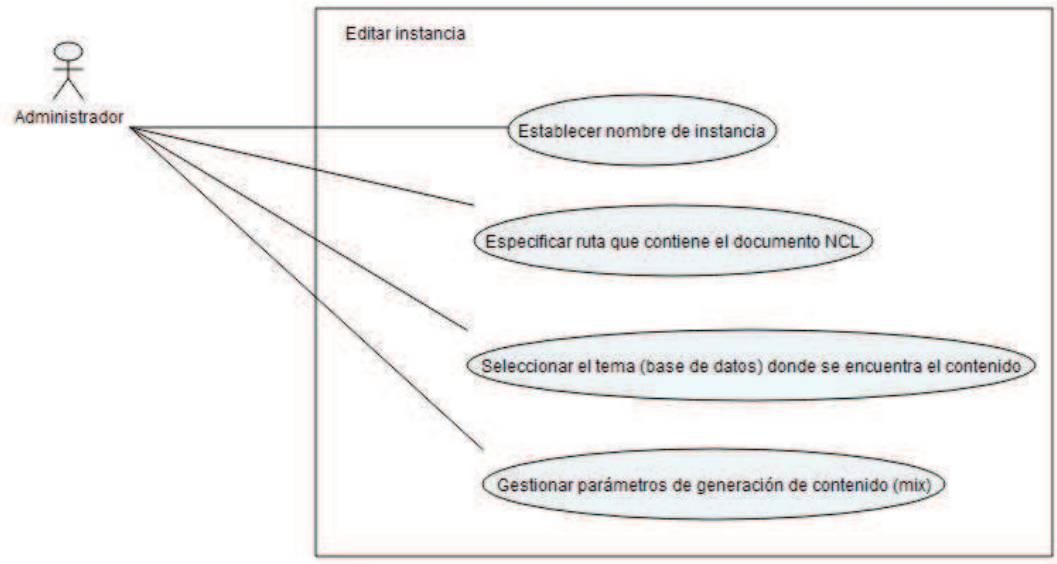


Figura 3.49 Diagrama de casos de uso para editar una instancia

El usuario administrador puede editar una instancia realizando acciones tales como:

- **Establecer el nombre de la instancia:** Consiste en introducir un nombre que identifique el proceso de asociación que lleva a cabo la aplicación Mixer para una determinada aplicación interactiva.

- **Especificar la ruta que contiene el documento NCL:** Determina la ruta hacia el documento NCL de la aplicación interactiva.
- **Seleccionar el tema (base de datos) donde se encuentra el contenido:** Determina la base de datos de SQL Server que posee las distintas tablas a ser empleadas para el proceso de asociación.
- **Gestionar parámetros de generación de contenido (mix):** Comprende acciones orientadas a la revisión de los resultados del proceso de asociación y determinación de parámetros para la generación de imágenes PNG con transparencia y de *scripts* Lua.

3.3.1.1.2 Gestión de parámetros de generación de contenido (mix)

La Figura 3.50 indica los casos de uso para gestionar los parámetros de generación de contenido (proceso de asociación o *mix*). El usuario puede:

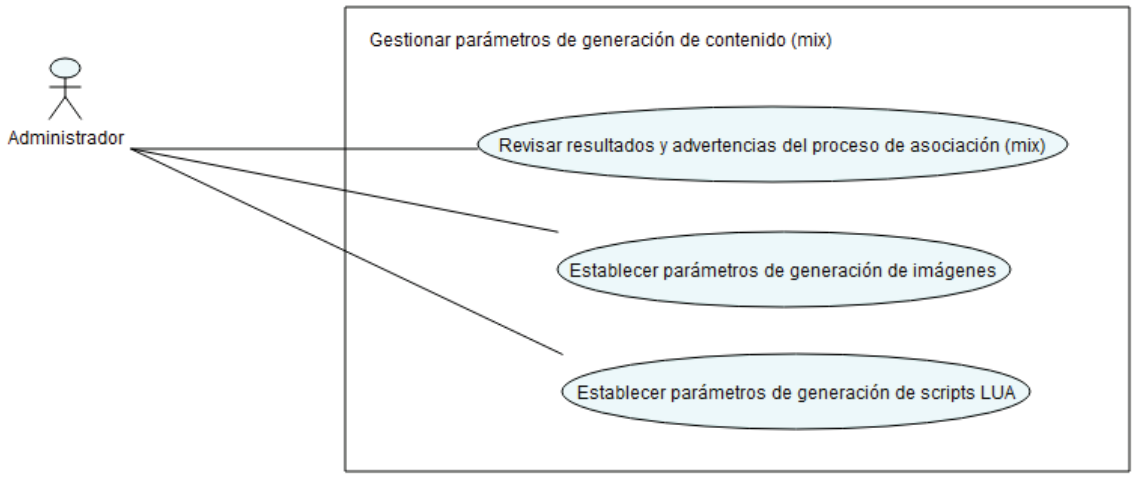


Figura 3.50 Diagrama de casos de uso para gestionar los parámetros de generación de contenido (*mix*)

- **Revisar los resultados y advertencias del proceso de asociación (*mix*):** La aplicación Mixer genera reportes por cada metadato indicando el éxito del proceso de asociación o generando advertencias (*warnings*), los cuales pueden ser visualizados por el usuario.
- **Establecer los parámetros de generación de imágenes:** Consiste en determinar, para cada metadato de tipo uno y dos, los parámetros con los cuales la aplicación Mixer generará la o las imágenes PNG con transparencia correspondientes al metadato.

- **Establecer los parámetros de generación de scripts Lua:** Consiste en determinar, para cada metadato de tipo tres, los parámetros con los cuales la aplicación Mixer generará el *script* Lua para obtener la información que requiere a través del canal de retorno.

3.3.1.1.3 Establecemiento de parámetros de generación de imágenes

La Figura 3.51 detalla los casos de uso con el fin de establecer los parámetros de generación de imágenes en metadatos de tipo uno y dos.

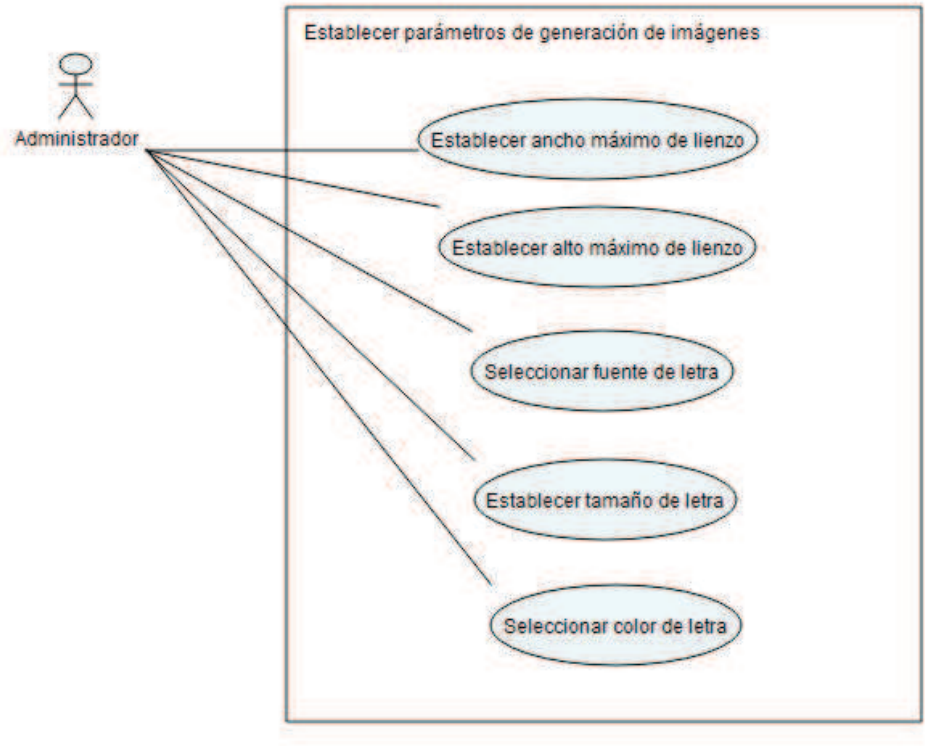


Figura 3.51 Diagrama de casos de uso para establecer los parámetros de generación de imágenes

Estos casos comprenden acciones tales como:

- **Establecer el ancho máximo del lienzo:** El usuario puede ingresar en píxeles el ancho máximo de la imagen PNG sobre la cual puede dibujar el texto la aplicación Mixer.
- **Establecer el alto máximo del lienzo:** El usuario puede ingresar en píxeles el alto máximo de la imagen PNG sobre la cual puede dibujar el texto la aplicación Mixer.

- **Seleccionar la fuente de letra:** Establece el tipo de letra para el texto.
- **Establecer el tamaño de letra:** Establece el tamaño de letra para el texto.
- **Seleccionar el color de letra:** Establece el color de letra para el texto.

3.3.1.1.4 Establecimiento de parámetros de generación de scripts Lua

La Figura 3.52 detalla los casos de uso con el fin de establecer los parámetros de generación de *scripts* Lua en metadatos de tipo tres.

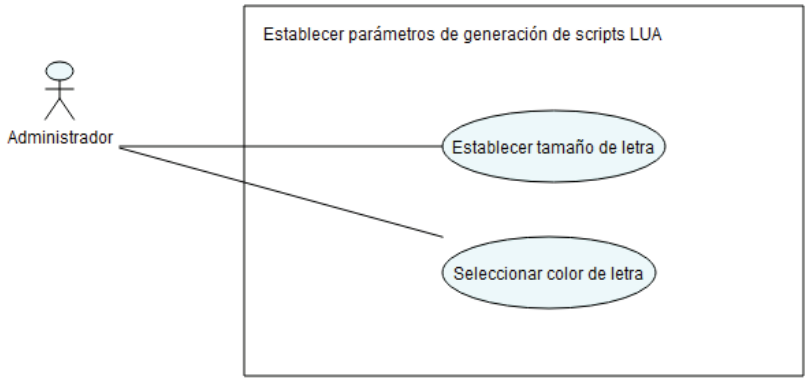


Figura 3.52 Diagrama de casos de uso para establecer los parámetros de generación de *scripts* Lua

Estos casos comprenden dos acciones:

- **Establecer el tamaño de letra:** Determina el tamaño de letra para el texto con el cual se presentará en la pantalla del televisor la información obtenida al través del canal de retorno.
- **Seleccionar el color de letra:** Determina el color de letra para el texto.

3.3.1.2 Estructura jerárquica de clases [3]

Para poder realizar el proceso de asociación, así como para poder guardar y abrir el trabajo del usuario en el proceso de asociación de las distintas aplicaciones interactivas (instancias), la aplicación Mixer emplea una estructura jerárquica de clases mediante asociaciones binarias *one way* y relaciones de composición, la cual facilita la manipulación de datos y permite guardar el trabajo del usuario mediante la serialización de un único objeto en un archivo binario [3]; de manera similar a la estructura jerárquica de clases empleada por el servicio SOAP.

Esta estructura jerárquica de clases se presenta en la Figura 3.53.

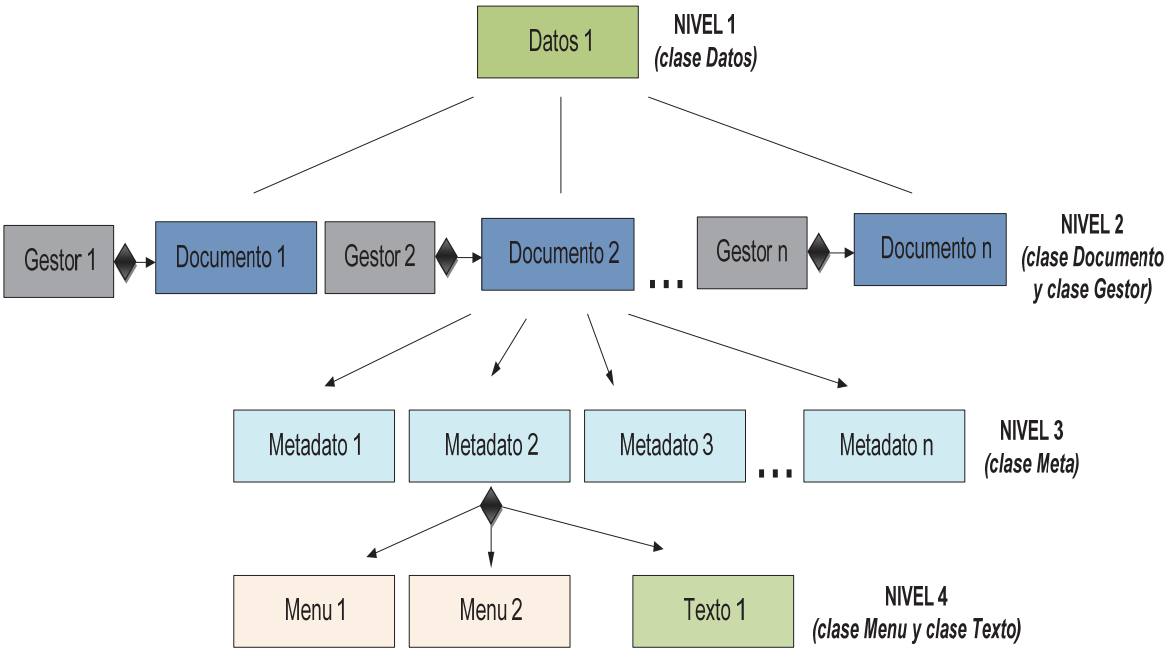


Figura 3.53 Estructura jerárquica de clases de la aplicación Mixer

La clase de nivel superior ha sido denominada `Datos`. Esta clase es instanciada una sola vez y tiene referencias a objetos de la clase `Documento`. Mediante la serialización del objeto de la clase `Datos` en un archivo binario se puede guardar el trabajo de asociación realizado por el usuario administrador.

La clase `Documento` representa un documento NCL, el cual contiene los metadatos con los cuales se realizará el proceso de asociación de una determinada aplicación interactiva. Un objeto de esta clase es manipulado por un objeto de la clase `Gestor`, la cual implementa los métodos necesarios para obtener los metadatos en el documento NCL, obtener los nodos que describen dichos metadatos, invocar al servicio SOAP para consultar la base de datos indicada por el usuario para el proceso de asociación y generar la o las imágenes PNG con el texto para los metadatos de tipo uno y dos, así como generar los *scripts* Lua para los metadatos de tipo tres.

La relación de composición entre la clase `Gestor` y la clase `Documento` tiene una multiplicidad de 1 a 1 en la que será creado un objeto de la clase `Gestor` siempre que sea creado antes un objeto de la clase `Documento` [3].

La clase `Meta` representa un metadato válido contenido en el documento NCL a partir del cual se ha podido generar la o las imágenes PNG o el *script* Lua que describe. Un metadato es válido únicamente si las propiedades `name` y `content` están acordes a uno de los tres tipos de metadatos con los que trabaja la aplicación Mixer, si en el documento NCL existe el nodo y si en la base de datos existe la celda o la tabla a los que hace referencia el metadato. Si el metadato no es válido, la aplicación Mixer lo ignora.

La clase `Menu` representa un nodo NCL que modela un menú desarrollado con el *plug-in* Menu Creator, o bien un menú formado en base a las dimensiones de la tabla encontrada en la base de datos. Por otro lado, la clase `Texto` representa las propiedades del texto que el usuario ha seleccionado para generar las imágenes PNG o el *script* Lua. Entre la clase `Meta` y las clases `Menu` y `Texto` existe una relación de composición.

Por cada metadato válido, el objeto de la clase `Gestor` que opera sobre el objeto de la clase `Documento` creará un objeto de la clase `Texto` basándose en las propiedades indicadas por el usuario y haciendo uso del servicio SOAP para obtener de la base de datos el texto. Si el metadato es de tipo uno, se intentará crear dos objetos de la clase `Menu`: uno correspondiente al menú obtenido del documento NCL y otro que se modela en base a la tabla de la base de datos asignada a través del metadato. Si el objeto de la clase `Gestor` encuentra en el documento NCL toda la información necesaria para crear dichos objetos, creará un objeto de la clase `Meta` correspondiente al metadato, asignándole una referencia al objeto de la clase `Texto` y dos referencias a los objetos de la clase `Menu` anteriormente creados. Luego, una referencia al nuevo objeto creado de la clase `Meta` será añadida a la colección de metadatos del objeto de la clase `Documento`.

La Figura 3.54 presenta el diagrama de clases que emplea la aplicación Mixer.

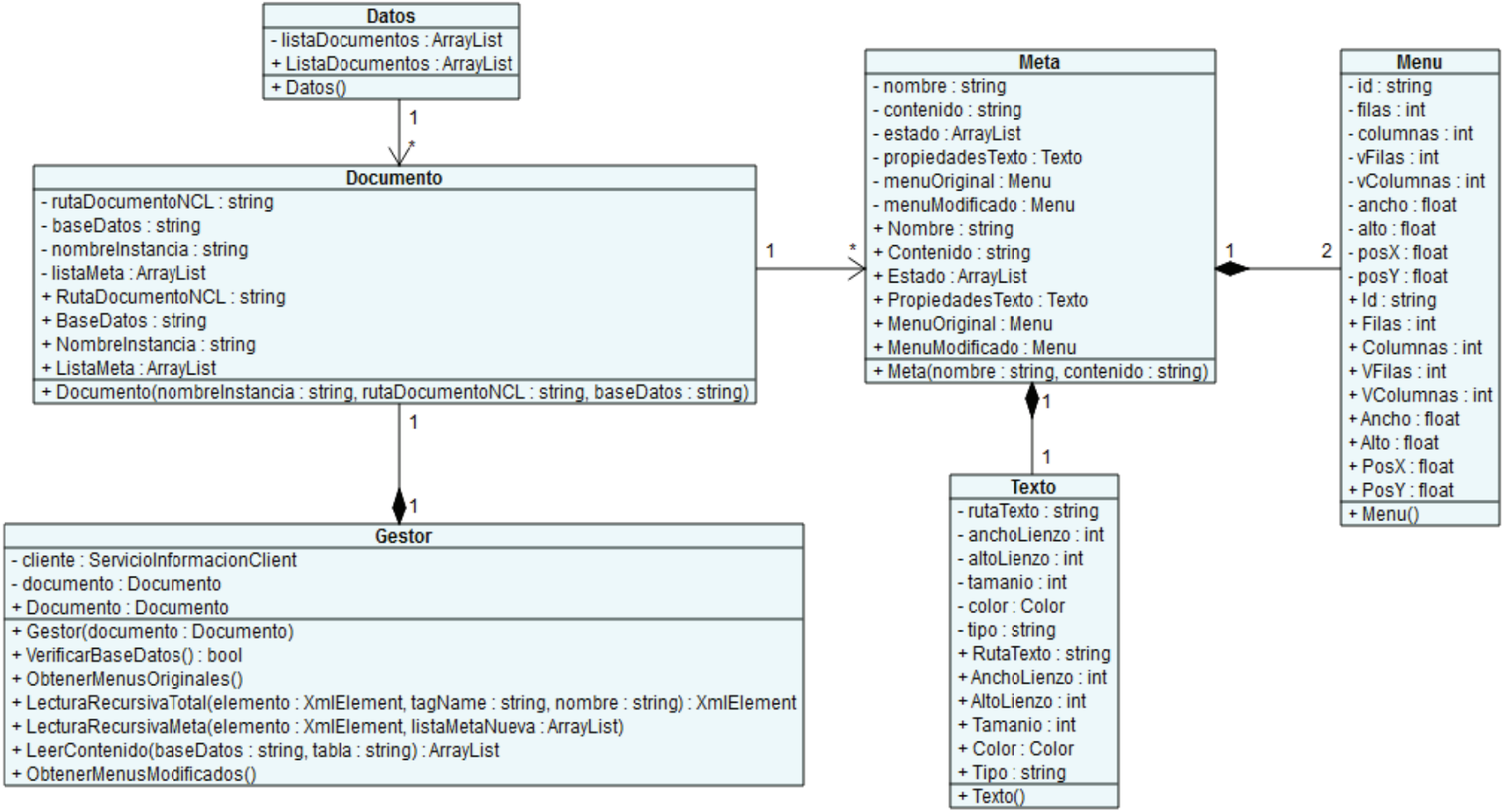


Figura 3.54 Diagrama de clases de la aplicación Mixer

3.3.1.2.1 Clase Datos

La clase `Datos` modela todo el trabajo de asociación realizado por la aplicación Mixer con uno o varios documentos NCL.

El único atributo de esta clase es una colección de tipo `ArrayList` de nombre `listaDocumentos`, en la cual se almacenan las referencias de los objetos de la clase `Documento` creados. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaDocumentos`.

3.3.1.2.2 Clase Documento

La clase `Documento` modela un documento NCL, el cual contiene los metadatos y nodos a partir de los cuales se realizará el trabajo de asociación.

Los atributos de esta clase son:

- **`rutaDocumentoNCL:`** Atributo de tipo `string`. Corresponde a la ruta hacia el documento NCL de una instancia creada. El valor de este atributo puede obtenerse o establecerse con la propiedad `RutaDocumentoNCL`.
- **`baseDatos:`** Atributo de tipo `string`. Corresponde al nombre de la base de datos que se empleará para el proceso de asociación. El valor de este atributo puede obtenerse o establecerse con la propiedad `BaseDatos`.
- **`nombreInstancia:`** Atributo de tipo `string`. Corresponde al nombre de la instancia creada. El valor de este atributo puede obtenerse o establecerse con la propiedad `NombreInstancia`.
- **`listaMeta:`** Atributo de tipo `ArrayList`. Consiste en una colección con referencias a objetos de la clase `Meta`. El valor de este atributo puede obtenerse o establecerse con la propiedad `ListaMeta`.

El constructor de esta clase requiere el valor de los atributos `nombreInstancia`, `rutaDocumentoNCL` y `baseDatos` para crear un objeto. Dichos atributos son indicados por el usuario a través de la interfaz gráfica de la aplicación.

3.3.1.2.3 Clase Meta

La clase `Meta` modela un metadato válido declarado en el documento NCL a partir del cual la aplicación Mixer puede generar la o las imágenes PNG o el *script* Lua que describe.

Los atributos de esta clase son:

- **`nombre:`** Atributo de tipo `string`. Corresponde al valor de la propiedad `name` de la etiqueta del metadato. El valor de este atributo puede obtenerse o establecerse con la propiedad `Nombre`.
- **`contenido:`** Atributo de tipo `string`. Corresponde al valor de la propiedad `content` de la etiqueta del metadato. El valor de este atributo puede obtenerse o establecerse con la propiedad `Contenido`.
- **`estado:`** Atributo de tipo `ArrayList`. Consiste en una colección en la que se almacenan paulatinamente las distintas advertencias (*warnings*) que se generan durante el proceso de asociación. El valor de este atributo puede obtenerse o establecerse con la propiedad `Estado`.
- **`propiedadesTexto:`** Atributo de tipo `Texto`. Referencia al objeto de la clase `Texto` del metadato. El valor de este atributo puede obtenerse o establecerse con la propiedad `PropiedadesTexto`.
- **`menuOriginal:`** Atributo de tipo `Menu`. Referencia al objeto de la clase `Menu` del metadato, correspondiente al menú que se obtiene desde el documento NCL y que ha sido diseñado mediante el *plug-in* Menu Creator. El valor de este atributo puede obtenerse o establecerse con la propiedad `MenuOriginal`.
- **`menuModificado:`** Atributo de tipo `Menu`. Referencia al objeto de la clase `Menu` del metadato, correspondiente al menú que se modela en base a las dimensiones de la tabla de la base de datos asignada en la propiedad `content` del metadato especificado en el documento NCL. El valor de este atributo puede obtenerse o establecerse con la propiedad `MenuModificado`.

El constructor de esta clase requiere de los atributos `nombre` y `contenido` para crear un objeto.

3.3.1.2.4 Clase Menu

La clase `Menu` modela un menú, para un metadato de tipo uno, ya sea obtenido mediante un nodo de contexto correspondiente a un menú creado con el *plug-in* Menu Creator, o bien un menú modelado en base a las dimensiones de la tabla de la base de datos asignada.

Los atributos de esta clase son:

- **`id:`** Atributo de tipo `string`. Corresponde al valor de la propiedad `id` de la etiqueta del nodo. El valor de este atributo puede obtenerse o establecerse con la propiedad `Id`.
- **`filas:`** Atributo de tipo `int`. Corresponde al número de filas del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `Filas`.
- **`columnas:`** Atributo de tipo `int`. Corresponde al número de columnas del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `Columnas`.
- **`vFilas:`** Atributo de tipo `int`. Corresponde al número de filas de cada vista del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `VFilas`.
- **`vColumnas:`** Atributo de tipo `int`. Corresponde al número de columnas de cada vista del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `VColumnas`.
- **`ancho:`** Atributo de tipo `float`. Corresponde al ancho del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `Ancho`.
- **`alto:`** Atributo de tipo `float`. Corresponde al alto del menú. El valor de este atributo puede obtenerse o establecerse con la propiedad `Alto`.

- **posX:** Atributo de tipo `float`. Corresponde a la coordenada horizontal de la posición del menú en la pantalla del televisor. El valor de este atributo puede obtenerse o establecerse con la propiedad `PosX`.
- **posY:** Atributo de tipo `float`. Corresponde a la coordenada vertical de la posición del menú en la pantalla del televisor. El valor de este atributo puede obtenerse o establecerse con la propiedad `PosY`.

El constructor de esta clase no tiene parámetros.

3.3.1.2.5 Clase Texto

La clase `Texto` modela las propiedades de texto seleccionadas por el usuario para la generación de la o las imágenes PNG en metadatos de tipo uno y dos, o para el *script* Lua en metadatos de tipo tres.

Los atributos de esta clase son:

- **rutaTexto:** Atributo de tipo `string`. Si el metadato hace referencia a un nodo de contexto, el valor de este atributo corresponde a la carpeta de texto en donde se almacenarán las imágenes PNG con el texto de cada elemento del menú. Si el metadato hace referencia a un nodo de contenido, el valor de este atributo corresponde al valor de la propiedad `src` de la etiqueta XML del nodo concatenado a la ruta hacia la carpeta del proyecto Composer de la aplicación interactiva. El valor de este atributo puede obtenerse o establecerse con la propiedad `RutaTexto`.
- **anchoLienzo:** Atributo de tipo `int`. Corresponde al valor del ancho máximo de la imagen o las imágenes PNG a crearse expresado en píxeles. El valor de este atributo puede obtenerse o establecerse con la propiedad `AnchoLienzo`.
- **altoLienzo:** Atributo de tipo `int`. Corresponde al valor del alto máximo de la imagen o las imágenes PNG a crearse expresado en píxeles. El valor de este atributo puede obtenerse o establecerse con la propiedad `AltoLienzo`.

- **tamanio:** Atributo de tipo `int`. Corresponde al tamaño de letra para el texto seleccionado por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Tamanio`.
- **color:** Atributo de tipo `Color`. Corresponde al color de letra para el texto seleccionado por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Color`.
- **tipo:** Atributo de tipo `string`. Corresponde al tipo de letra para el texto seleccionado por el usuario. El valor de este atributo puede obtenerse o establecerse con la propiedad `Tipo`.

El constructor de esta clase no tiene parámetros.

Si el metadato es de tipo tres, únicamente se establece el valor de los atributos `rutaTexto`, `tamanio` y `color` de esta clase.

3.3.1.2.6 Clase Gestor

La clase `Gestor` modela una clase encargada de realizar, de forma transparente, las distintas operaciones necesarias para el proceso de asociación sobre un objeto de la clase `Documento`.

Los atributos de esta clase son:

- **cliente:** Atributo de tipo `ServicioInformacionClient`[69]. Corresponde al *proxy* necesario para hacer uso de los métodos del servicio SOAP con el fin de obtener información de las bases de datos, así como para obtener la dirección IP necesaria para el formar el URL en el *script* Lua, en el caso de metadatos de tipo tres.
- **documento:** Atributo de tipo `Documento`. Referencia al objeto de la clase `Documento` con el cual opera el objeto de la clase `Gestor`. El valor de este atributo puede obtenerse o establecerse con la propiedad `Documento`.

[69] `ServicioInformacionClient` es el nombre de la referencia hacia el servicio SOAP. Esta referencia puede ser usuada en la aplicación al igual que una clase. El objeto creado a partir de esta referencia corresponde al *proxy* que permite la invocación de los métodos del servicio SOAP.

El constructor de esta clase requiere el valor del atributo `documento`.

La Tabla 3.8 detalla los métodos de la clase `Gestor`.

Tabla 3.8 Métodos de la clase `Gestor`

Método	Funcionalidad
`public bool VerificarBaseDatos()`	Retorna el valor de `true` si la base de datos indicada por el usuario existe en SQL Server.
`public void ObtenerMenusOriginales()`	Obtiene por cada metadato encontrado en el documento NCL el nodo que describe el metadato y crea los objetos de la clase `Menu` correspondientes a los nodos sin datos.
`public XmlElement LecturaRecursivaTotal(XmlElement elemento,string tagName,string nombre)`	Método recursivo para explorar cada etiqueta XML del documento NCL, con el fin de encontrar el elemento XML correspondiente a una entidad NCL con un determinado valor de su propiedad `id`.
`public void LecturaRecursivaMeta(XmlElement elemento,ArrayList listaMetaNueva)`	Método recursivo para explorar cada etiqueta XML del documento NCL, con el fin de obtener los metadatos válidos contenidos en el documento.
`public ArrayList LeerContenido(string baseDatos,string tabla)`	Obtiene una colección con la información almacenada en cada una de las celdas de una tabla de una determinada base de datos.
`public void ObtenerMenusModificados()`	Genera la imagen o las imágenes PNG con transparencia de cada metadato de tipo uno y dos, así como el *script* Lua de cada metadato de tipo tres. Además crea los objetos de la clase `Menu` en base a las dimensiones de la tabla de la base de datos asignada para los metadatos tipo uno.

3.3.1.3 Interfaz gráfica

La interfaz gráfica de la aplicación Mixer está conformada por los siguientes formularios:

- `FrmSplash`
- `FrmAutenticar`
- `FrmVentanaPrincipal`
- `FrmAyuda`
- `FrmAcercaDe`

3.3.1.3.1 Formulario FrmSplash y formulario FrmAutenticar

Los formularios `FrmSplash` y `FrmAutenticar`, que se presentan en la Figura 3.55 y en la Figura 3.56, respectivamente, cumplen con las mismas funcionalidades que los formularios de iguales nombres de la aplicación cliente, es decir, el formulario `FrmSplash` presenta el nombre de la aplicación y el formulario `FrmAutenticar` permite al usuario autenticarse ante el sistema.

Figura 3.55 Formulario *FrmSplash* de la aplicación Mixer

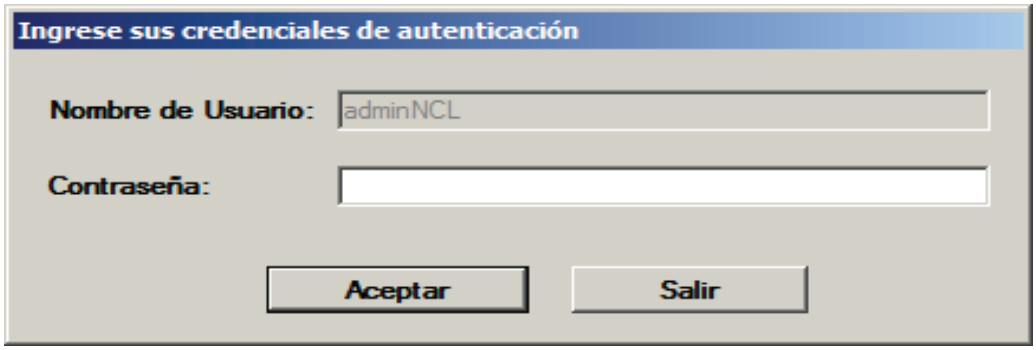


Figura 3.56 Formulario `FrmAutenticar` de la aplicación Mixer

3.3.1.3.2 Formulario FrmVentanaPrincipal

Una vez que el usuario administrador ha ingresado correctamente su contraseña, se presenta el formulario `FrmVentanaPrincipal` que se muestra en la Figura 3.57. El listado de las instancias guardadas anteriormente se listan en la parte izquierda del formulario. A partir de esta ventana, el usuario puede crear, modificar y eliminar instancias que realicen el proceso de asociación en base a metadatos para distintas aplicaciones interactivas que se estén desarrollando.

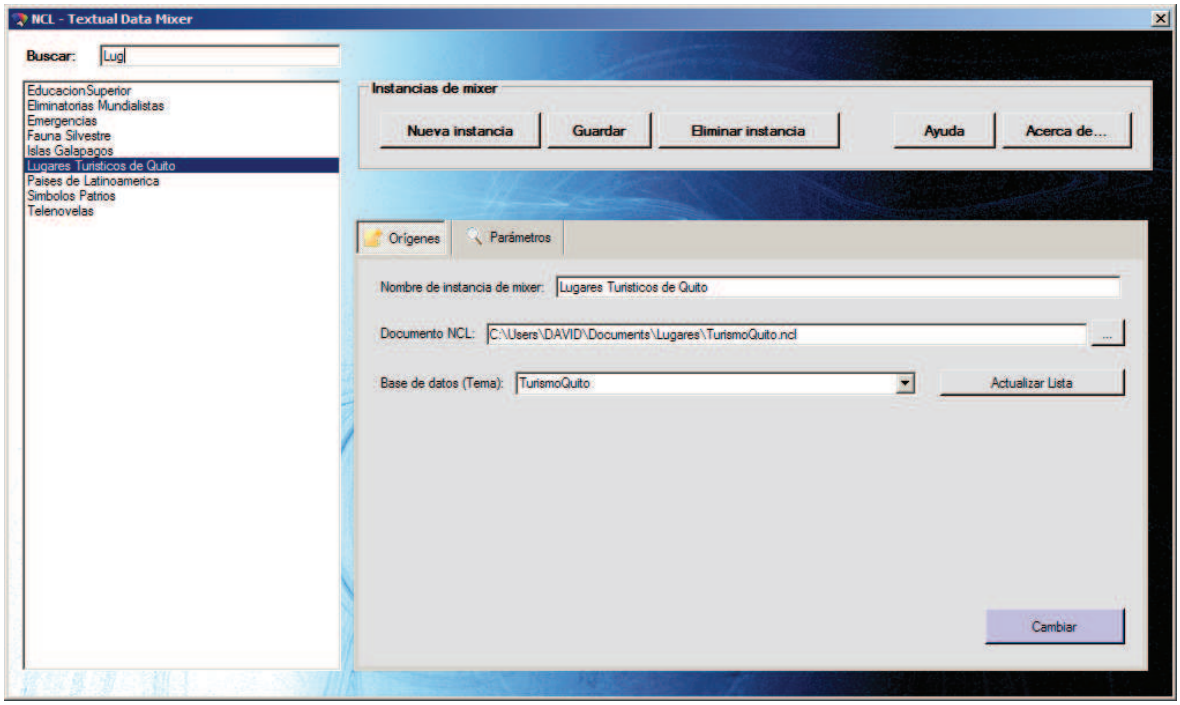


Figura 3.57 Formulario `FrmVentanaPrincipal` de la aplicación Mixer

La pestaña *"Origen"* del formulario presenta los parámetros correspondientes al proceso de asociación como el nombre de la instancia, la ruta al documento NCL y la base de datos empleada.

Al hacer clic en la pestaña *"Parámetros"*, la aplicación Mixer realiza el proceso de asociación en base a los metadatos encontrados en el documento NCL y presenta un reporte por cada metadato, indicando el éxito en el proceso de asociación mediante el mensaje CORRECTO señalado en color verde o presentando advertencias (*warnings*) mediante el mensaje ADVERTENCIA señalado en color amarillo.

La Figura 3.58 muestra cómo está conformada la pestaña *"Parámetros"*. La parte izquierda está destinada a presentar un listado con todos los metadatos encontrados, de manera que cuando el usuario haga un clic sobre un determinado metadato observe el estado del mismo (mensaje CORRECTO o ADVERTENCIA) y se desplieguen en el lado derecho las propiedades de texto empleadas, de manera que el usuario pueda modificarlas si así lo desea. En caso de que el proceso de asociación haya generado advertencias para un determinado metadato, estas se presentarán en la parte inferior cuando se selecciona el metadato.

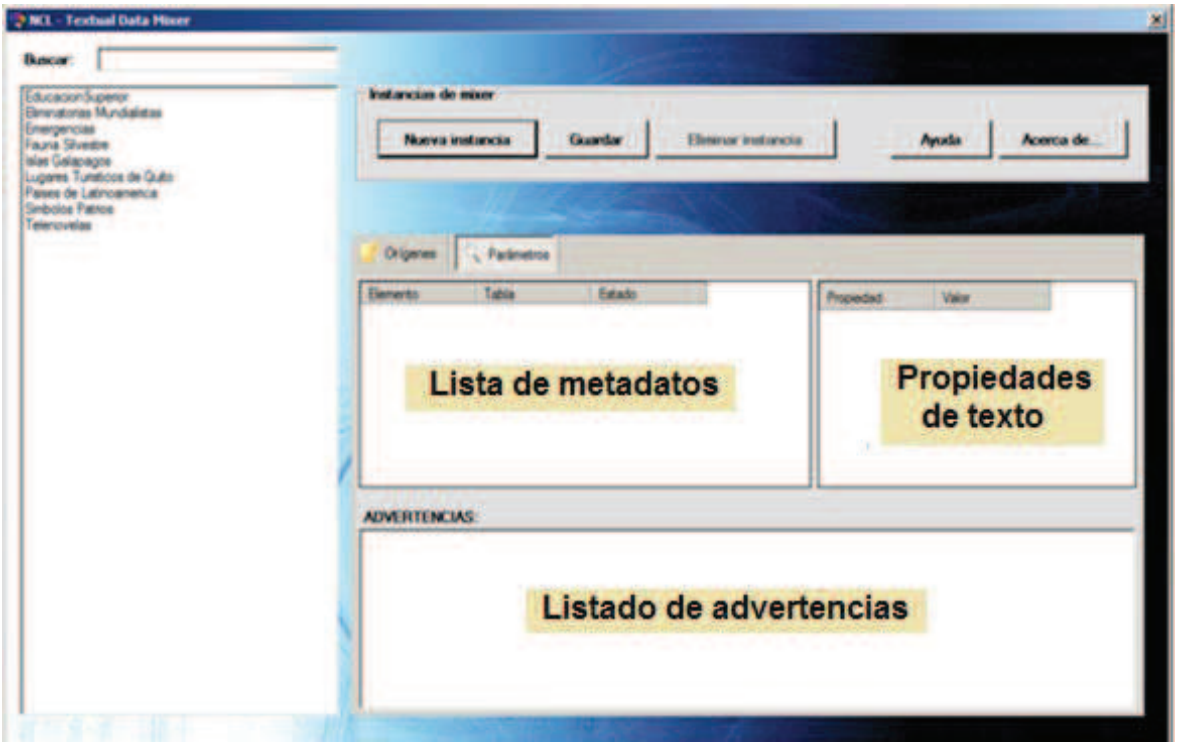


Figura 3.58 Pestaña "*Parámetros"* del formulario `FrmVentanaPrincipal` de la aplicación Mixer

La Figura 3.59 presenta un ejemplo de la aplicación Mixer asociando datos. En este caso, el proceso de asociación ha sido exitoso.

Si el metadato es de tipo uno o dos, en las propiedades de texto el usuario puede ingresar el valor máximo del ancho y alto del lienzo en píxeles, así como también el tamaño de letra. Para el tipo de letra, la aplicación pone a disposición del usuario un `ComboBox` que desplegará un listado con todas las fuentes disponibles, como se muestra en la Figura 3.60. En lo que corresponde al color, al

hacer clic sobre la casilla *"Valor"* de la opción *"Color"*, la aplicación Mixer pondrá a disposición del usuario una completa paleta de colores como la que se presenta en la Figura 3.61, lo cual permite al usuario seleccionar el color de letra con facilidad.

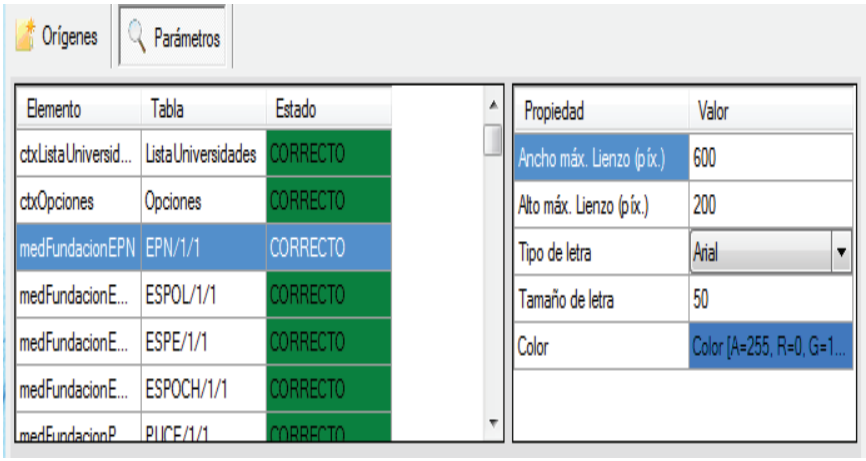


Figura 3.59 Asociación de datos con la aplicación Mixer

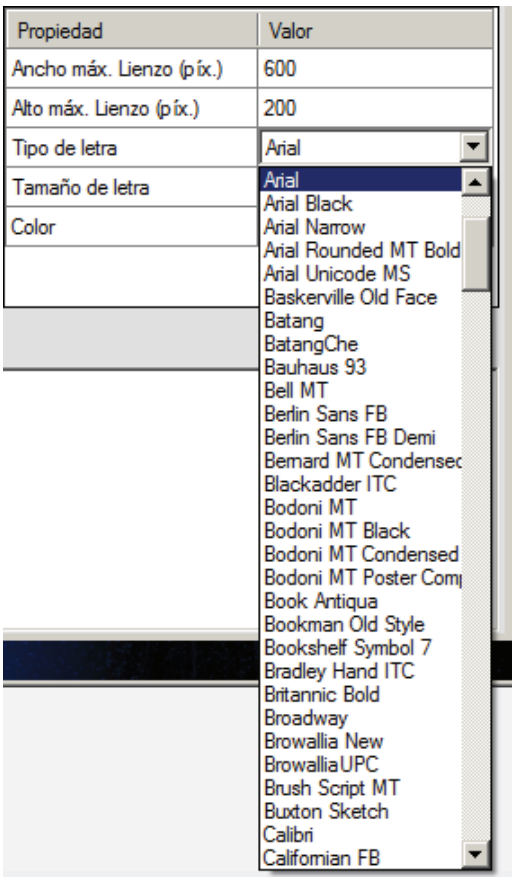


Figura 3.60 `ComboBox` con distintas fuentes de la aplicación Mixer

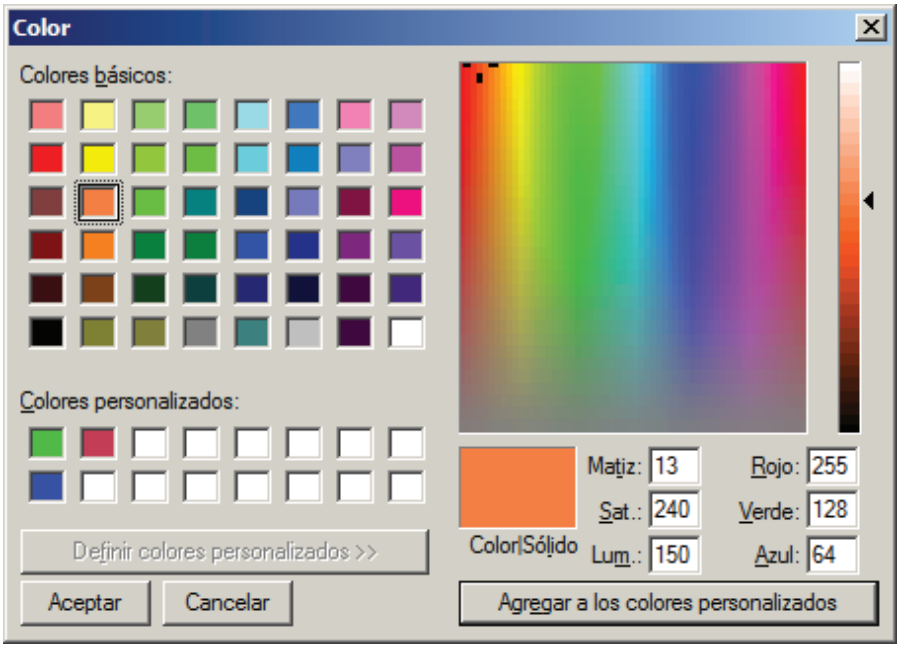


Figura 3.61 Paleta de colores de la aplicación Mixer

La Figura 3.62 presenta un ejemplo de asociación de datos para un metadato de tipo tres. En este caso, en las propiedades de texto únicamente se permite al usuario determinar el tamaño y el color de letra.

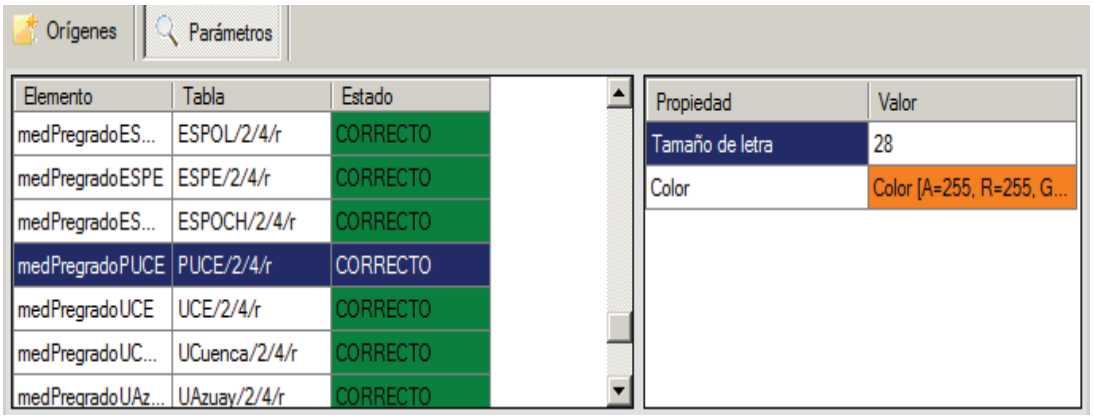


Figura 3.62 Ejemplo de asociación de datos para un metadato de tipo tres

Finalmente, la Figura 3.63 presenta un ejemplo de asociación de datos con generación de advertencias (*warnings*), las cuales se muestran en color amarillo. Las advertencias generadas para cada metadato se listan en la parte inferior.

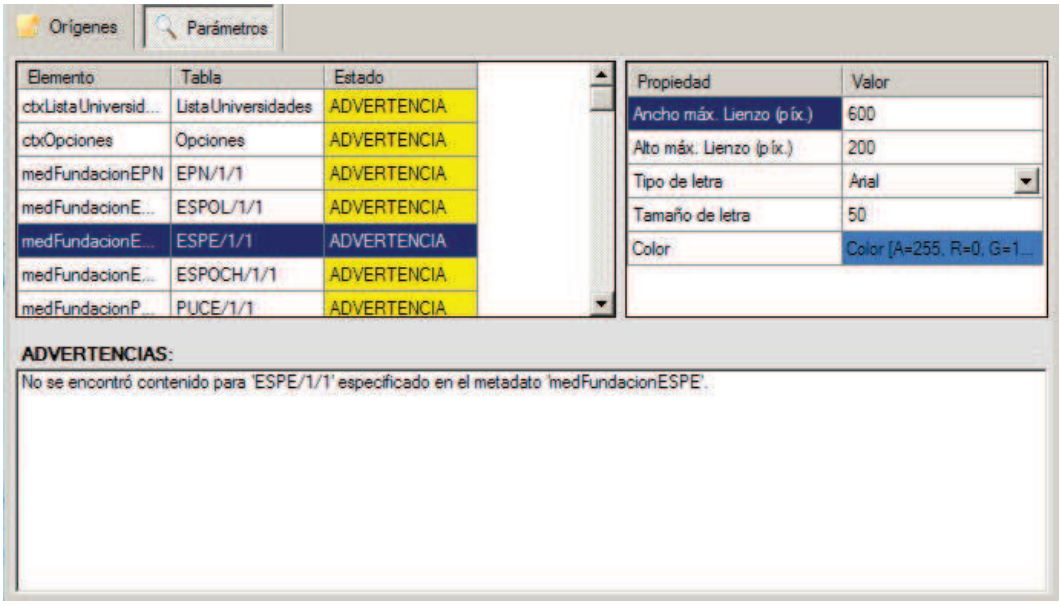


Figura 3.63 Ejemplo de asociación de datos con generación de advertencias (*warnings*)

3.3.1.3.3 Formulario FrmAyuda

El formulario `FrmAyuda` que se presenta en la Figura 3.64 detalla el funcionamiento de la aplicación Mixer con el fin de ayudar al usuario a utilizar la aplicación.

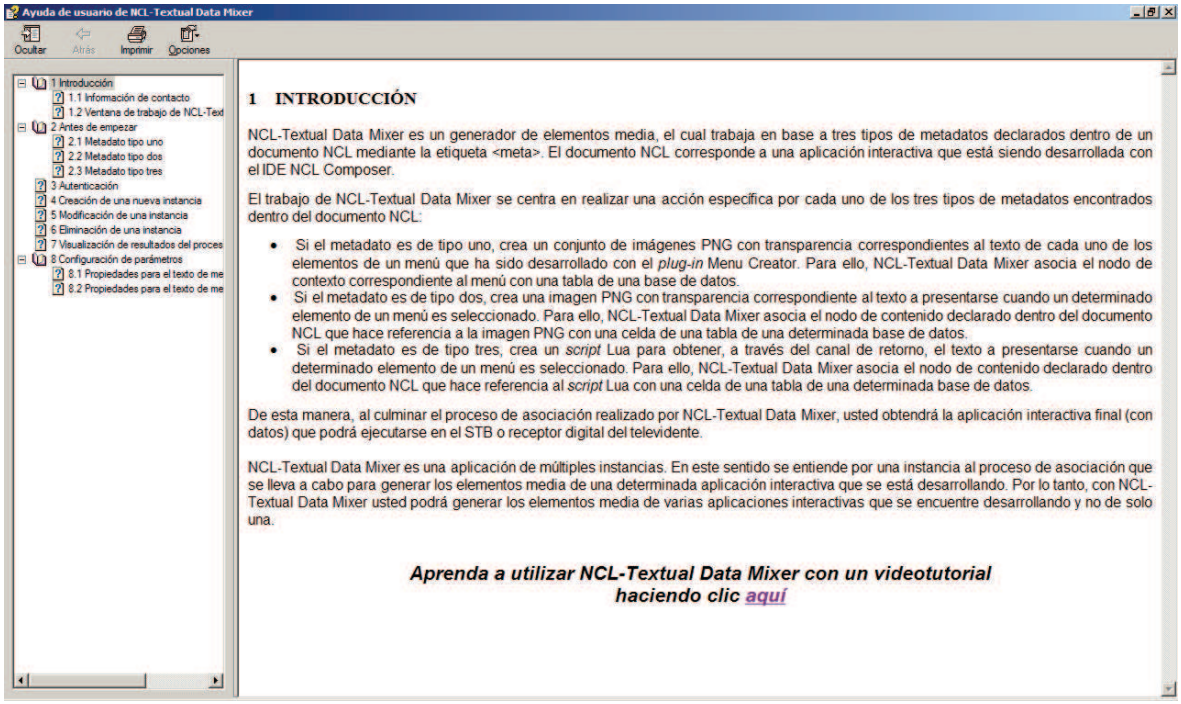


Figura 3.64 Formulario `FrmAyuda` de la aplicación Mixer

3.3.1.3.4 Formulario FrmAcercaDe

El formulario `FrmAcercaDe` se presenta en la Figura 3.65. Al abrir este formulario, el usuario puede visualizar información general sobre la aplicación Mixer.

Figura 3.65 Formulario `FrmAcercaDe` de la aplicación Mixer

3.3.2 FUNCIONALIDADES

Básicamente, la aplicación Mixer es un generador de elementos media (imágenes PNG con transparencia y *scripts* Lua), que realiza su trabajo en base a los tres tipos de metadatos anteriormente descritos.

3.3.2.1 Metadato de tipo uno

Un metadato de tipo uno describe la asociación entre un menú desarrollado con el *plug-in* Menu Creator y una tabla de una determinada base de datos. La aplicación Mixer crea, con el texto de cada celda de la tabla, una imagen PNG con transparencia correspondiente al texto de cada uno de los elementos del menú.

Puede darse el caso de que el menú desarrollado no sea de las mismas dimensiones que la tabla con la cual se asocia. En este caso, la aplicación Mixer compara el número de filas y de columnas del menú con el número de filas y columnas de la tabla, y en base a esta comparación toma el menor número de filas y de columnas para generar las imágenes PNG. Las imágenes serán creadas

en la carpeta de texto que el usuario especifica con el *plug-in* Menu Creator en la pestaña *"Parámetros"*.

Así por ejemplo, en caso de que haya alguna discordancia entre las dimensiones del menú y la tabla de la base de datos que se asocia, la aplicación enviará un mensaje de advertencia (*warning*) y asociará los elementos de manera normal dejando casillas vacías del menú en caso de que las dimensiones de la tabla sean menores a las del menú, o ignorando datos en caso de que las dimensiones del menú sean menores a las de la tabla de la base de datos. Si no hay ningún inconveniente en el proceso de asociación se indicará el éxito del mismo mediante el mensaje CORRECTO.

Por otro lado, si las dimensiones del texto con las propiedades seleccionadas por el usuario superan las dimensiones del lienzo, la aplicación Mixer recorta automáticamente parte del texto para colocar puntos suspensivos. La Figura 3.66 presenta un menú desarrollado en el que el texto de cada elemento ha sido recortado y completado con puntos suspensivos.

Figura 3.66 Menú con texto recortado y completado con punto suspensivos

3.3.2.2 Metadato de tipo dos

Un metadato de tipo dos describe la asociación entre un nodo de contenido referente a una imagen PNG declarado dentro del documento NCL y una celda de una tabla de una determinada base de datos.

La idea del uso de este metadato es presentar la información correspondiente a un determinado elemento de un menú, una vez que este ha sido seleccionado por el televidente con el control remoto. El metadato facilita el trabajo en el desarrollo de la aplicación interactiva, pues el usuario únicamente deberá declarar el nodo en el proyecto Composer, lo cual se puede realizar fácilmente con el *plug-in* NCL Outline View, y la aplicación Mixer se encargará de generar la imagen PNG transparente con el texto correspondiente y de almacenarla en la ruta conformada por la ruta de la carpeta del proyecto Composer y el valor de la propiedad *src* del nodo.

Si las dimensiones de las líneas de texto, con las propiedades seleccionadas por el usuario superan el ancho del lienzo, la aplicación Mixer hace que el texto se muestre en varias líneas automáticamente para que pueda caber en el tamaño del lienzo disponible. De esta manera, el televidente podrá visualizar la información en la pantalla del televisor sin problemas. La Figura 3.67 presenta una imagen PNG transparente con texto creada con un metadato de tipo dos, superpuesta sobre una segunda imagen de fondo color verde. El texto se muestra en varias líneas, debido a que las líneas del texto superan el ancho establecido para el lienzo.

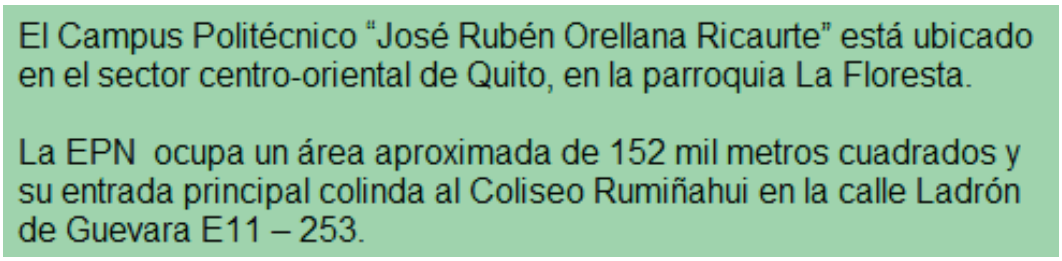


Figura 3.67 Imagen PNG transparente con texto superpuesta sobre una segunda imagen de fondo color verde

Así, cada vez que el televidente seleccione otro elemento de un menú, el usuario únicamente deberá detener el nodo de contenido actual que presenta el texto e iniciar otro nodo de contenido referente a otra imagen PNG transparente con texto correspondiente al nuevo elemento seleccionado, nuevamente superpuesto sobre la misma imagen de fondo.

3.3.2.3 Metadato de tipo tres [16]

Un metadato de tipo tres describe la asociación entre un nodo de contenido referente a un *script* Lua declarado dentro del documento NCL y una celda de una tabla de una determinada base de datos. El fin de este metadato es el mismo que el de un metadato de tipo dos, pero el texto se obtiene mediante el canal de retorno a través del *script* Lua que se ejecuta en el STB del televidente.

Hacer uso de este tipo de metadato es recomendable cuando se tiene información dinámica[70] (en vivo) o cuando los datos son demasiado grandes, lo que involucraría radiar una aplicación interactiva de mayor tamaño.

La aplicación Mixer genera de forma automática el *script* Lua correspondiente al nodo de contenido declarado en el documento NCL, el cual consiste en un archivo de texto que contiene las sentencias en el lenguaje Lua para ser ejecutado en el STB del televidente.

Para ello, la aplicación Mixer crea un objeto de la clase `StreamWriter` mediante el espacio de nombres `System.IO`. El constructor de esta clase toma tres parámetros [16]:

- La ruta en donde se creará el archivo de texto indicando su extensión (en este caso .lua), expresada mediante una cadena de texto.
- Un valor booleno (`true` o `false`). El valor de `true` indica que en caso de existir el archivo, debe añadirse a este las operaciones de escritura que se realicen. El valor de `false`, por el contrario, indica que el archivo deberá sobreescribirse con las operaciones de escritura realizadas.
- El tipo de codificación del archivo. Para que el *script* pueda ser ejecutado en el STB del televidente es necesario que el archivo esté codifcado en ASCII normal.

Una vez creado un objeto de la clase `StreamWriter` se podrá realizar las operaciones de escritura sobre el archivo mediante la invocación de los métodos

[70] Es decir información que varía a través del tiempo, como por ejemplo: datos estadísticos, información del clima, indicadores económicos, etc.

`Write()` o `WriteLine()`, los cuales escriben una cadena texto y una cadena de texto con un salto de línea, respectivamente [16].

La Figura 3.68 resume la funcionalidad del *script* Lua generado por la aplicación Mixer. En primer lugar, la información a presentarse al usuario se obtiene a través del canal de retorno como texto plano. Luego, se adecua la información reconociendo los caracteres de salto de línea, diviendo el texto de cada línea en varias líneas más si es necesario y justificando el texto. Finalmente, la información se muestra en la pantalla del televisor del televidente mediante el módulo `Canvas`.

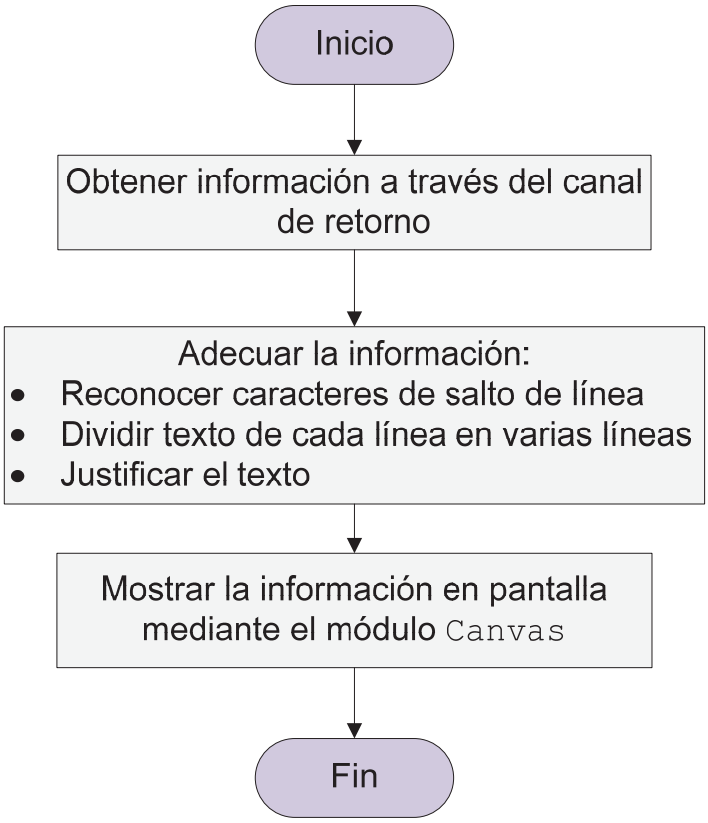


Figura 3.68 Diagrama de flujo simplificado del *script* Lua

Para cumplir con estas tres tareas, el *script* Lua implementa dos funciones que se detallan en la Tabla 3.9. La función `PedirDatos()` se encarga únicamente de solicitar la información a través del canal de retorno como texto plano. La función `MostrarDatos()` es la encargada de adecuar la información y presentarla en la pantalla del televidente mediante el módulo `Canvas`.

Tabla 3.9 Funciones del *script* Lua

Función	Propósito
`PedirDatos()`	Realiza una petición HTTP GET al servicio RESTful para obtener, a través del canal de retorno, la información de una celda de una tabla de una determinada base de datos como texto plano.
`MostrarDatos()`	Adecua la información obtenida a través del canal de retorno y muestra la información en pantalla mediante el módulo `Canvas`.

Ambas funciones son monitoreadas por medio de la función `pcall` de Lua para controlar posibles excepciones (errores en tiempo de ejecución). Así, en caso de que ocurra un error al intentar traer la información a través del canal de retorno, se establecerá una cadena de texto con un mensaje de no disponibilidad que indicará al usuario que no ha sido posible traer el contenido que está solicitando. En caso de que ocurra algún error al intentar mostrar la información en pantalla, el *script* finaliza. De esta manera, en caso de que se produzcan errores durante la ejecución del *script*, la aplicación interactiva no será detenida permitiendo al usuario seguir empleándola.

3.3.2.3.1 Obtención de la información a través del canal de retorno

La obtención de información a través del canal de retorno se realiza mediante la función `PedirDatos()` del *script* Lua. En esta función, el *script* crea un *socket* TCP mediante la librería `socket.http`. Invocando a la función `request()` de dicho *socket*, se envía una petición HTTP GET al servicio RESTful, el cual responderá con la información solicitada como texto plano.

El URL necesario para realizar la petición y que se pasa como argumento de la función `request()` se estructura de la forma que sugiere el Código 3.21.

```
http://dirección_de_red/ServicioRetorno/ServicioRetorno.svc/obtenertexto/nombre_Base_Datos/nombre_Tabla/#fila/#columna
```

Código 3.21 Estructura del URL para petición HTTP GET

La dirección IP del servidor donde está alojado el servicio RESTful se obtiene mediante el servicio SOAP. `ServicioRetorno` es el nombre del proyecto del

servicio RESTful, `ServicioRetorno.svc` es el nombre de la clase en la que se implementa el método del servicio y `obtenerTexto` es el identificador del método que se empleó al indicar el URI para acceder al método en el atributo `WebGet`. El nombre de la base de datos es indicada por el usuario al crear una nueva instancia en la aplicación Mixer. El nombre de la tabla de la base de datos y el número de fila y de columna son indicados por el metadato de tipo tres para el cual se crea el *script* Lua.

Únicamente si el código de estado HTTP resultante de la operación es igual a 200 (el cual indica que la operación fue exitosa), se procesará el texto obtenido a través del canal de retorno, de lo contrario, se establece una cadena de texto con un mensaje indicando que no está disponible la información que el televidente está solicitando.

3.3.2.3.2 Manejo de cadenas de texto con Lua [6], [22]

Lua incorpora varias funciones con el fin de manipular cadenas de texto. El *script* Lua hace uso de cuatro de estas funciones para adecuar el texto que se trae a través del canal de retorno para posteriormente presentarlo en pantalla. Estas cuatro funciones se detallan en la Tabla 3.10.

Todas estas funciones pueden invocarse mediante el operador dos puntos (`:`) de una variable de tipo `string` que contiene la cadena de texto. También pueden ser invocadas directamente si se antepone al listado de parámetros de cada función el nombre de la variable de tipo `string`, es decir, como si la variable fuese un parámetro más de la función.

Cuando se procede a adecuar el texto, en primer lugar, el *script* Lua dividirá el texto en varias líneas reconociendo el caracter de salto de línea `\n`, para luego volver a dividir, si es necesario, el texto de cada línea en varias líneas más, de manera que cada línea obtenida se coloque una debajo de otra. Luego, el *script* Lua se encargará de justificar cada línea de texto obtenida insertando espacios en blanco hasta que las dimensiones del texto de la línea se ajuste al ancho de la región.

Tabla 3.10 Funciones para manejo de cadenas de texto empleadas en el *script* Lua [6], [22]

Función	Parámetros	Funcionalidad
gmatch(patrón)	**patrón:** expresión de comparación empleada para determinar coincidencias dentro de la cadena de texto.	Devuelve una función iteradora que, cada vez que se invoca, retorna las siguientes capturas del `patrón` en la cadena de texto. Si el `patrón` no produce capturas entonces la coincidencia completa se devuelve en cada llamada.
len		Devuelve la longitud de una cadena de texto. Una cadena de texto vacía tiene una longitud de 0.
sub(i [,j])	**i:** número que determina el inicio de la subcadena dentro de la cadena de texto. **j:** número que determina el final de la subcadena dentro de la cadena de texto. Este parámetro es opcional.	Retorna la subcadena de la cadena que comienza en `i` y continúa hasta `j`; `i` y `j` pueden ser negativos. Si `j` está ausente entonces se asume que vale -1 (equivalente a la longitud de la cadena de texto).
gsub(patrón,reemplazo [,n])	**patrón:** expresión de comparación empleada para determinar coincidencias dentro de la cadena de texto. **reemplazo:** expresión a ser insertada en la cadena de texto por cada coincidencia encontrada. **n:** número de coincidencias para las cuales se realizará el reemplazamiento. Este parámetro es opcional.	Devuelve una copia de la cadena de texto en la que todas (o las `n` primeras, si se especifica el parámetro opcional) las apariciones del `patrón` han sido reemplazadas por el `reemplazo` especificado, que puede ser una cadena de texto, una tabla o una función; `gsub` también devuelve, como segundo valor, el número total de coincidencias detectadas.

Se emplea la función `gmatch()` para dividir el texto en varias líneas, de acuerdo al caracter de salto de línea `\n`. Dado que el STB no es capaz de reconocer este caracter, el *script* Lua formará, empleando esta función una colección de líneas separadas por el carácter `\n`.

Se usa la función `len()` para comparar la longitud de una determinada línea con el ancho de la región del nodo de contenido. Además, se usa la función `sub()` conjuntamente con esta función para dividir una línea en dos o más líneas, con el fin de que el texto de la línea no sea recortado y se pueda visualizar correctamente.

Finalmente, se emplea la función `gsub()` para realizar el proceso de justificación línea por línea. El *script* Lua hace uso de esta función para incorporar en cada línea resultante espacios en blanco entre palabra y palabra para justificar el texto.

La Figura 3.69 presenta un ejemplo en el que la longitud del texto de una línea obtenido a través del canal de retorno supera las dimensiones de la región, y la presentación no ha sido adecuada por lo que no puede ser visualizado completamente.

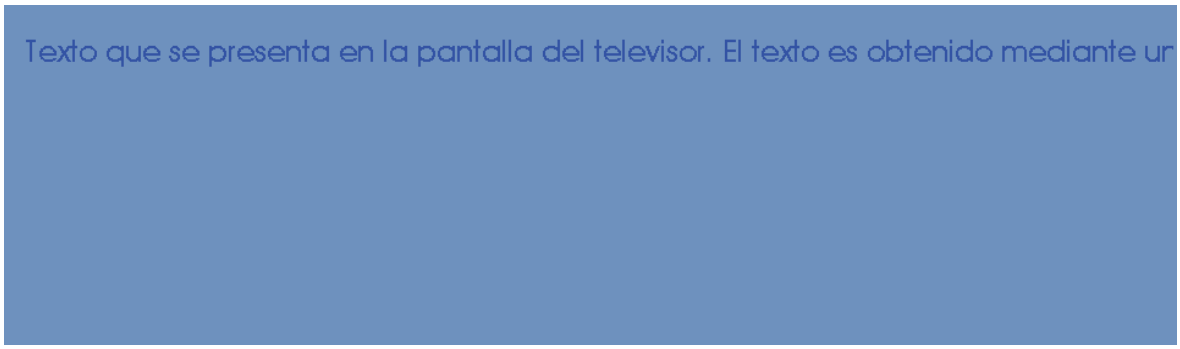


Figura 3.69 Texto sin dividirse en varias líneas y sin justificar

La Figura 3.70 presenta el mismo texto presentado adecuadamente mediante la funcionalidad del *script* Lua. La línea de texto ha sido dividida en varias líneas, de acuerdo al ancho de la región evitando que parte del texto quede recortado y no se pueda visualizar. Además, cada línea es justificada para una mejor presentación.

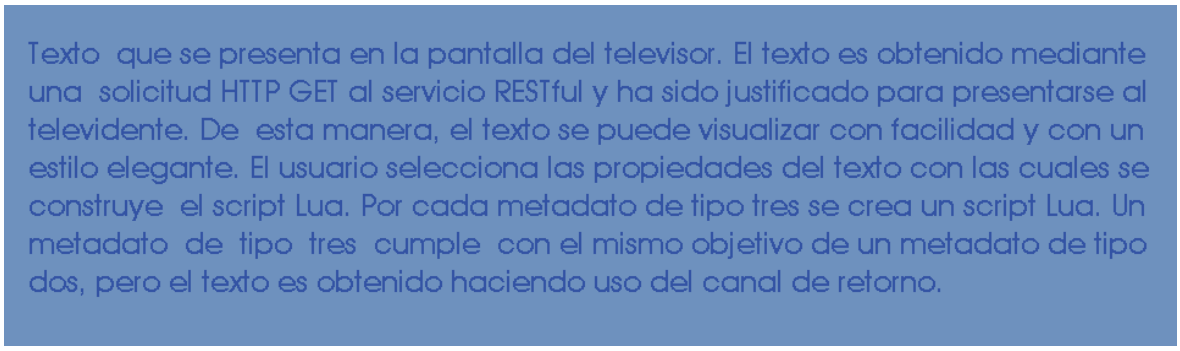


Figura 3.70 Texto dividido en varias líneas y justificado mediante el *script* Lua

3.3.2.3.3 Mostrar información en pantalla con el módulo Canvas

Las distintas funciones que componen el módulo `Canvas` permiten realizar operaciones gráficas. En el *script* Lua, se emplea el módulo `Canvas` para mostrar en pantalla la información obtenida a través del canal de retorno, o bien la cadena de texto con el mensaje de no disponibilidad generado al intentar traer el texto.

Dado que cuando se inicia el *script* Lua se crea de manera automática un objeto gráfico asignado a la variable global `canvas`, se puede invocar las distintas funciones sin preocuparse por instanciar un objeto. La Tabla 3.11 lista las funciones del módulo `Canvas` empleadas en el *script* Lua.

Tabla 3.11 Funciones del módulo `Canvas` empleadas en el *script* Lua

Función	Uso
`canvas:attrColor(R,G,B,A)`	Establece el color de fuente con el cual se graficará el texto, indicando sus componentes RGB según el color seleccionado por el usuario y sin transparencia (A=255).
`canvas:attrFont(face,size,style)`	Establece el tipo de fuente según el tipo de fuente seleccionado por el usuario.
`canvas:attrSize()`	Obtiene las dimensiones de la región correspondiente al nodo de contenido asociado con el *script* Lua.
`canvas:measureText(text)`	Obtiene las dimensiones de la cadena de texto que se pasa como argumento a la función.
`canvas:flush()`	Muestra la información en pantalla.

El Código 3.22 presenta un ejemplo de un *script* Lua generado por la aplicación Mixer. Todos los *scripts* Lua generados para los metadatos de tipo tres tienen la misma estructura pero no son los mismos, ya que la aplicación Mixer genera el *script* estructurando el URL de acuerdo a la base de datos, tabla y números de fila y columna indicados, y estableciendo los parámetros de letra seleccionados por el usuario.

```
-- -----------------------------------------------------------------------
-- Script creado por: NCL-Textual Data Mixer
-- Desarrollado por: David Cevallos
--                   Fernando Cevallos
-- Escuela Politecnica Nacional
-- Quito-Ecuador
-- -----------------------------------------------------------------------


function PedirDatos()
   -- Descarga de datos
   http = require "socket.http"
   resultado, codigoEstado, contenido = http.request("http://192.168.1.1/ServicioRetorno/ServicioRetorno.svc/obtenertexto/EducacionSuperior/EPN/
   1/2")
   if codigoEstado ~=200 then
      resultado = "De momento no disponible"
   end
end

function MostrarDatos()
   -- Establecimiento de propiedades de letra
   canvas:attrColor(0,0,0,255)
   canvas:attrFont('Verdana',32,'bold')

   -- Determina el ancho y alto de la region y del texto
   anchoTot, altoTot = canvas:attrSize()
   ancho, alto = canvas:measureText(resultado)
   anchoTot = anchoTot - 40

   -- Coleccion de lineas de texto a mostrarse al usuario
   coleccion = {}

   -- Ingreso de lineas que conforman el texto dentro de la coleccion
   for linea in resultado:gmatch("([^\n]*)\n?") do

      if linea:len() == 0 then
         table.insert(coleccion,linea)
      else

         -- Adapta el texto de manera optima para que se muestre en varias lineas en caso de superar el
         -- ancho de la region
         siguiente = 1
         repeat
            lineaNueva = linea:sub(siguiente)
            an, al = canvas:measureText(lineaNueva)

            while an>anchoTot do
               lineaNueva = lineaNueva:sub(1,-2)
               an, al = canvas:measureText(lineaNueva)
            end

            -- Justifica el texto
            if siguiente+lineaNueva:len()-1 ~= linea:len() then
               -- Segmenta en palabras
               lineaNueva = linea:sub(siguiente,siguiente+lineaNueva:len())
               lineaNueva,coincidencias = string.gsub(lineaNueva,"(.* ).*","%1")
               lineaNueva = linea:sub(siguiente,siguiente+lineaNueva:len()-2)
               siguiente = siguiente + lineaNueva:len() + coincidencias

               if coincidencias == 1 then
                  -- Expande el texto con espacios hasta excederse
                  anJ,alJ = canvas:measureText(lineaNueva)
```

Código 3.22 Ejemplo de un *script* Lua generado por la aplicación Mixer (primera parte)

```
                while anJ <= anchoTot do
                    lineaNueva,coincidenciasJ = string.gsub(lineaNueva,"(%S+) ","%1  ")
                    if coincidenciasJ==0 then
                        break;
                    end
                    anJ,alJ = canvas:measureText(lineaNueva)
                end

                if coincidenciasJ > 0 then
                    -- Contrae el texto en exceso
                    lineaNueva = string.gsub(lineaNueva,"(%S+) ","%1")
                    -- Acrecenta hasta ajustar el texto de manera adecuada
                    repeat
                        coincidenciasJ = coincidenciasJ - 1
                        lineaJustificada = string.gsub(lineaNueva,"(%S+) ","%1  ",coincidenciasJ)
                        anJ,alJ = canvas:measureText(lineaJustificada)
                    until anJ <= anchoTot
                    lineaNueva = lineaJustificada
                end
            end
            else
                siguiente = siguiente + lineaNueva:len()
            end
            table.insert(coleccion, lineaNueva)
        until linea:len() <= siguiente
    end
end

contador = 20

-- Salida en pantalla
for i, valor in ipairs (coleccion) do
        canvas:drawText(20, contador, valor)
        contador = contador + alto
end

canvas:flush()
end

if not pcall(PedirDatos) then
    resultado = "De momento no disponible"
end

if not pcall(MostrarDatos) then
    -- Nada que hacer
end
```

Código 3.22 Ejemplo de un *script* Lua generado por la aplicación Mixer (segunda parte)

REFERENCIAS

[1] CodePlex. HTMLAgilityPack. [Online]. Disponible en: http://htmlagilitypack.codeplex.com/ (Consultado el 2 de enero de 2014).

[2] D. Mejía. (2012) Servicios WCF (Windows Communication Foundation). [Diapositivas]. Quito-Ecuador. Escuela Politécnica Nacional.

[3] iCarnegie, “Modelando Clases – Diagramas UML de clases”, Departamento de Ciencias de Computación Universidad Carnegie Mellon, Pittsburgh.

[4] G. Colouris, J. Dollimore, T. Kindberg y G. Blair, *Distributed Systems Concepts and Design*, Quinta edición, Addison-Wesley, USA: Pearson, 2012.

[5] J. R. Groff y P. N. Weinberg, *Manual de Referencia SQL*, Segunda edición, Ed. Madrid, España: McGraw-Hill, 2003.

[6] Lua Reference Manual. [Online]. Disponible en: http://pgl.yoyo.org/luai/i/about (Consultado el 15 de enero de 2013).

[7] Microsoft Technet. Motor de base de datos de SQL Server. [Online]. Disponible en: http://technet.microsoft.com/es-es/library/ms187875.aspx (Consultado el 25 de noviembre de 2013).

[8] Microsoft. BinaryFormatter (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.runtime.serialization.formatters.binary.binaryformatter(v=vs.110).aspx (Consultado el 2 de enero de 2014).

[9] Microsoft. Encoding (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.text.encoding%28v=vs.110%29.aspx (Consultado el 2 de enero de 2014).

[10] Microsoft. File (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.io.file(v=vs.110).aspx (Consultado el 27 de noviembre de 2013).

[11] Microsoft. HttpUtility (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.web.httputility%28v=vs.110%29.aspx (Consultado el 2 de enero de 2014).

[12] Microsoft. Introducción a las aplicaciones de Servicios Windows. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/d56de412%28v=vs.110%29.aspx (Consultado el 3 de enero de 2014).

[13] Microsoft. MIME Type Detection in Windows Internet Explorer. [Online]. Disponible en: http://msdn.microsoft.com/en-us/library/ms775147%28VS.85%29.aspx (Consultado el 3 de enero de 2014).

[14] Microsoft. SerializableAtributte (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.serializableattribute.aspx (consultado el 27 de noviembre de 2013).

[15] Microsoft. SqlCommand (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqlcommand(v=vs.110).aspx (Consultado el 25 de noviembre de 2013).

[16] Microsoft. StreamWriter (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.io.streamwriter%28v=vs.110%29.aspx **(C**onsultado el 15 de enero de 2014).

[17] Microsoft. System.Data.SqlClient (Espacio de nombres). [Online]. Disponible en:http://msdn.microsoft.com/es-es/library/system.data.sqlclient(v=vs.110).aspx (Consultado el 25 de noviembre de 2013).

[18] Microsoft. Tutorial: Crear una aplicación de servicios de Windows en el Diseñador de componentes. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/zt39148a%28v=vs.110%29.aspx (Consultado el 3 de enero de 2014).

[19] Microsoft. WebClient (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.net.webclient%28v=vs.110%29.aspx (Consultado el 2 de enero de 2014).

[20] MSDN Magazine. An Introduction to RESTful Services with WCF. [Online]. Disponible en: http://msdn.microsoft.com/es-es/magazine/dd315413.aspx (Consultado el 3 de enero de 2014).

[21] P. Deitel y H. Deitel, *C# 2010 for programmers*, Cuarta edición ed., RR Donnelley, Ed. Crawfordsville, USA: Prentice Hall, 2010.

[22] R. Ierusalimschy, L. Henrique de Figueiredo, y W. Celes. (2011, Septiembre) Manual de Referencia de Lua 5.1. [Online]. Disponible en: http://www.lua.org/manual/5.1/es/manual.html (Consultado el 15 de enero de 2013).

[23] S. Cheng, *Microsoft Windows Communication Foundation Cookbook for Developing SOA Applications*, Primera edición, Ed. Loncoln Road, USA: Packt Publishing, 2010.

[24] StackOverFlow Forum. WCF ResponseFormat for WebGet. [Online]. Disponible en: http://stackoverflow.com/questions/992533/wcf-responseformat-for-webget (Consultado el 6 de enero de 2014).

[25] T. García. SISTEMAS DISTRIBUIDOS RESTful. [Online]. Disponible en: http://tedhygarcia.blogspot.com/2012/06/restful.html#!/2012/06/restful.html (Consultado el 3 de enero de 2014).

[26] Visual Studio. Sintaxis de XPATH. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/vstudio/ms256471%28v=vs.100%29.aspx (Consultado el 8 d enero de 2014).

[27] Wikinsonpc. Codificando y decodificando una dirección URL. [Online]. Disponible en: http://www.wilkinsonpc.com.co/free/articulos/codificar-decodificar-url.html (Consultado el 8 de enero de 2014).

[28] Wikipedia. Microsoft SQL Server. [Online]. Disponible en: http://es.wikipedia.org/wiki/Microsoft_SQL_Server (Consultado el 25 de noviembre de 2013).

CAPÍTULO 4

DESARROLLO DE UNA APLICACIÓN INTERACTIVA, PRUEBAS Y RESULTADOS

En este capítulo se presenta el desarrollo de una aplicación interactiva híbrida, enfocada en la temática de la Educación Superior del Ecuador, haciendo uso del sistema de búsqueda, almacenamiento y procesamiento de información. La aplicación interactiva presenta información general acerca de las once universidades que conformaron la categoría A durante el período 2009-2013, como resultado de la evaluación realizada por el CONEA[71]. A partir de esta aplicación se ha desarrollado, como ejemplo de la flexibilidad que el sistema brinda en la creación de aplicaciones interactivas, una segunda aplicación que presenta información general acerca de las cinco universidades que en la actualidad[72] conforman esta categoría, como resultado de la evaluación realizada por el CEAACES[73].

Se presenta el desarrollo de ambas aplicaciones interactivas, empleando las herramientas del sistema de búsqueda, almacenamiento y procesamiento de información. Finalmente, se presentan las pruebas de funcionamiento de las aplicaciones interactivas desarrolladas y los resultados finales obtenidos.

4.1 PROTOTIPO DEL SISTEMA DE BÚSQUEDA, ALMACENAMIENTO Y PROCESAMIENTO DE INFORMACIÓN [1], [4]

La Figura 4.1 detalla el prototipo del sistema de búsqueda, almacenamiento y procesamiento de información que se ha empleado para el desarrollo de las

[71] Consejo Nacional de Evaluación y Acreditación de la Educación Superior del Ecuador [3].
[72] La redacción de este proyecto fue culminada en el año 2014.
[73] Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEAACES) [2]. En noviembre de 2013, el CEAACES publicó los resultados de una nueva evaluación en la que cinco universidades constan en la categoría A, de las cuales tres universidades tienen oferta académica de pregrado y postgrado, y las dos restantes únicamente oferta de postgrado [2].

aplicaciones interactivas y las pruebas de funcionamiento. Para este prototipo se ha establecido una pequeña red LAN empleando un *router* de marca Linksys con conexión a Internet, el cual permite tener conectividad entre los distintos elementos que conforman el sistema.

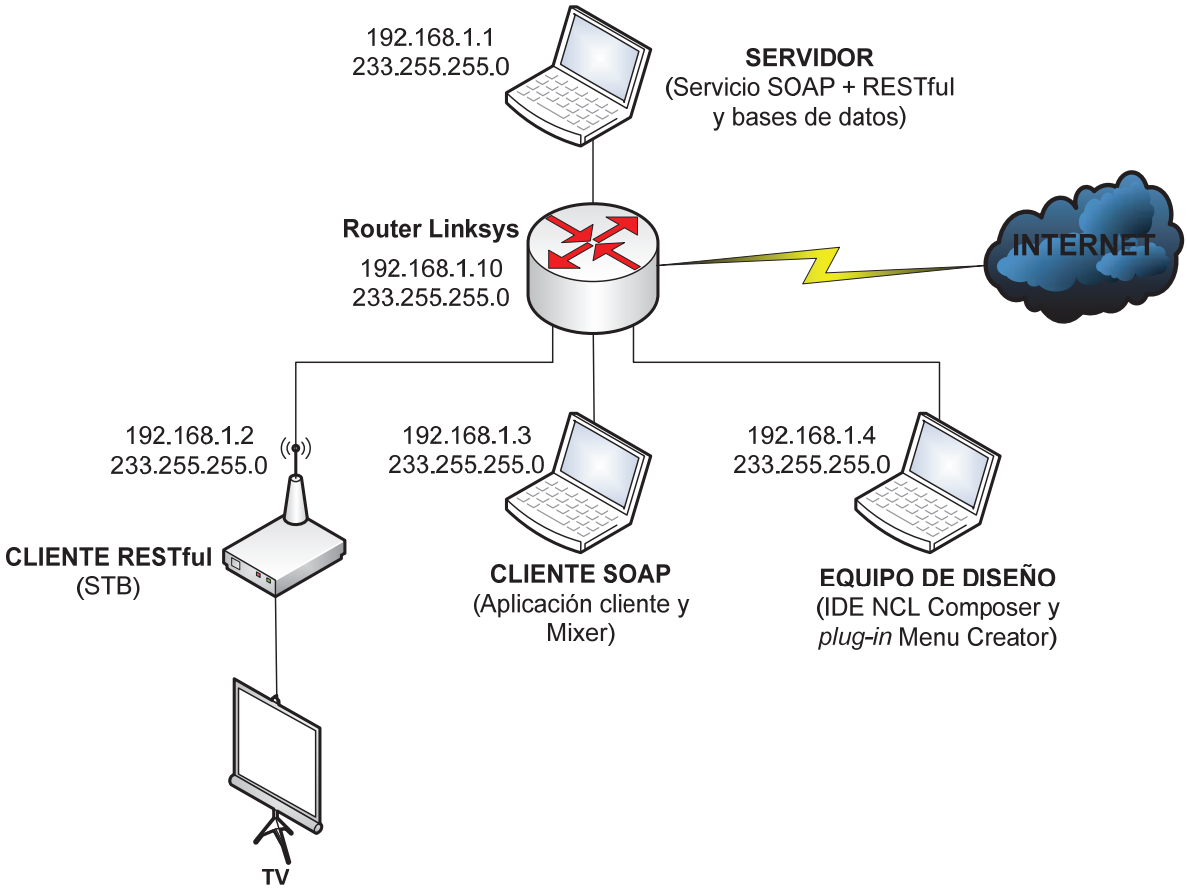


Figura 4.1 Diagrama del prototipo del sistema de búsqueda, almacenamiento y procesamiento de información

Los servicios SOAP, RESTful y de bases de datos han sido alojados en un único computador central que cumple con el rol de servidor. De igual manera, las aplicaciones cliente (Textual Data Creator) y Mixer (NCL-Textual Data Mixer) se han instalado en un único computador, el cual cumple con el rol de cliente, de manera que las aplicaciones puedan consumir el servicio SOAP. El STB del televidente también toma el rol de cliente para consumir el servicio RESTful.

En un tercer computador se ha instalado el IDE NCL Composer con el *plug-in* Menu Creator. Este computador no consume ningún servicio, pero dado que la aplicación Mixer requiere del documento NCL creado por el IDE para realizar el

trabajo de asociación de metadatos, se ha optado por compartir desde este computador una carpeta en red que contenga los documentos NCL de las aplicaciones interactivas desarrolladas con el IDE.

La Tabla 4.1 detalla el direccionamiento empleado en la implementación del prototipo.

Tabla 4.1 Direccionamiento empleado en la implementación del prototipo

Dispositivo	Dirección IP	Máscara de subred	Gateway
Servidor	192.168.1.1	255.255.255.0	192.168.1.10
Cliente RESTful (STB)	192.168.1.2	255.255.255.0	192.168.1.10
Cliente SOAP	192.168.1.3	255.255.255.0	192.168.1.10
Equipo de diseño	192.168.1.4	255.255.255.0	192.168.1.10
Router Linksys	192.168.1.10	255.255.255.0	-

La Tabla 4.2 presenta las características técnicas del computador empleado como servidor.

Tabla 4.2 Características técnicas del computador empleado como servidor

Característica	Especificación
Disco duro	50 GB
Procesador	Intel(R) Core(TM) i5-2430M CPU @ 2.40 GHz
Memoria RAM	4 GB
Sistema Operativo	Microsoft Windows 7 Professional
Resolución Pantalla	1366 x 768, colores de 32 bits

El televisor empleado en el prototipo es de marca Samsung. El STB empleado es un STB híbrido de marca EiTV, el cual decodifica señales de televisión digital terrestre del estándar ISDB-Tb y de televisión sobre el protocolo de Internet (IPTV). La Tabla 4.3 resume las principales características técnicas de este STB y en el Anexo F.2 se presentan en detalle sus características técnicas.

Tabla 4.3 Principales características técnicas del STB híbrido EiTV [4]

Característica	Especificación
Procesador	STi 7105 (CPU 450 MHz)
Memoria RAM	256 MB
Memoria Flash	128 MB
Frecuencia de entrada	UHF: 470MHz (CH14) a 806MHz (CH69) VHF: 174MHz (CH7) a 216MHz (CH13)
Interfaces de comunicación	High Speed USB 2.0 Ethernet – 100 Mbps (RJ45)

En la Figura 4.2 se presenta la vista en perspectiva del STB híbrido EitV. En la Figura 4.3 se puede apreciar la vista posterior de este STB.

Figura 4.2 STB híbrido EiTV (vista en perspectiva) [4]

Figura 4.3 STB híbrido EiTV (vista posterior) [4]

La Tabla 4.4 presenta las características técnicas del computador empleado como cliente SOAP.

Tabla 4.4 Características técnicas del computador empleado como cliente SOAP

Característica	Especificación
Disco duro	60 GB
Procesador	Intel(R) Core(TM) i3-350M CPU @ 2.27 GHz
Memoria RAM	4 GB
Sistema Operativo	Microsoft Windows 7 Professional
Resolución Pantalla	1366 x 768, colores de 32 bits

La Tabla 4.5 presenta las características técnicas del computador empleado para el uso del IDE NCL Composer y el *plug-in* Menu Creator (equipo de diseño).

Tabla 4.5 Características técnicas del computador empleado como equipo de diseño

Característica	Especificación
Disco duro	30 GB
Procesador	Intel(R) Core(TM) i5-2450M CPU @ 2.50 GHz
Memoria RAM	8 GB
Sistema Operativo	Linux Fedora 17
Resolución Pantalla	1366 x 768, colores de 32 bits

Las principales características técnicas del *router* Linksys empleado se presentan en la Tabla 4.6.

Tabla 4.6 Características técnicas del *router* Linksys [1]

Característica	Especificación
Modelo	WRT120N
Funcionalidades	*Router*, *switch* y *access point*
Puertos	Internet, FastEthernet (1-4)
Número de antenas	2 (internas)
Estándares	IEEE 802.3u, 802.11g, 802.11b, versión 802.11n

En la Figura 4.4 se presenta la vista en perspectiva del *router* Linksys, y en la Figura 4.5 se puede apreciar la vista posterior del *router*.

Figura 4.4 *Router* Linksys (vista en perspectiva) [1]

Figura 4.5 *Router* Linksys (vista posterior) [1]

4.2 DISEÑO DE LA PRIMERA APLICACIÓN INTERACTIVA [3]

Las once universidades que pertenecieron a la categoría A durante el período 2009-2013, y sobre las cuales la primera aplicación interactiva presentará información son [3]:

- Escuela Politécnica Nacional (EPN)
- Escuela Superior Politécnica del Litoral (ESPOL)
- Escuela Politécnica del Ejército (ESPE)
- Escuela Superior Politécnica del Chimborazo (ESPOCH)
- Pontificia Universidad Católica de Quito (PUCE)
- Universidad Central del Ecuador (UCE)
- Universidad de Cuenca (UCUENCA)
- Universidad del Azuay (UAZUAY)
- Universidad San Francisco de Quito (USFQ)
- Universidad Técnica de Ambato (UTA)
- Universidad Técnica Particular de Loja (UTPL)

Para que el televidente pueda seleccionar una universidad específica, la aplicación contará con un primer menú ubicado en la parte izquierda del televisor que presentará los acrónimos de las once universidades. Este menú contará con once filas y una columna, y se presentará mediante vistas conformadas por cuatro elementos, para optimizar el espacio. Las viñetas de navegación permitirán que el televidente pueda moverse entre las distintas vistas y seleccionar una determinada universidad.

La Figura 4.6 presenta un bosquejo del diseño del primer menú, en el que se presenta su segunda vista, y su ubicación en la pantalla del televisor.

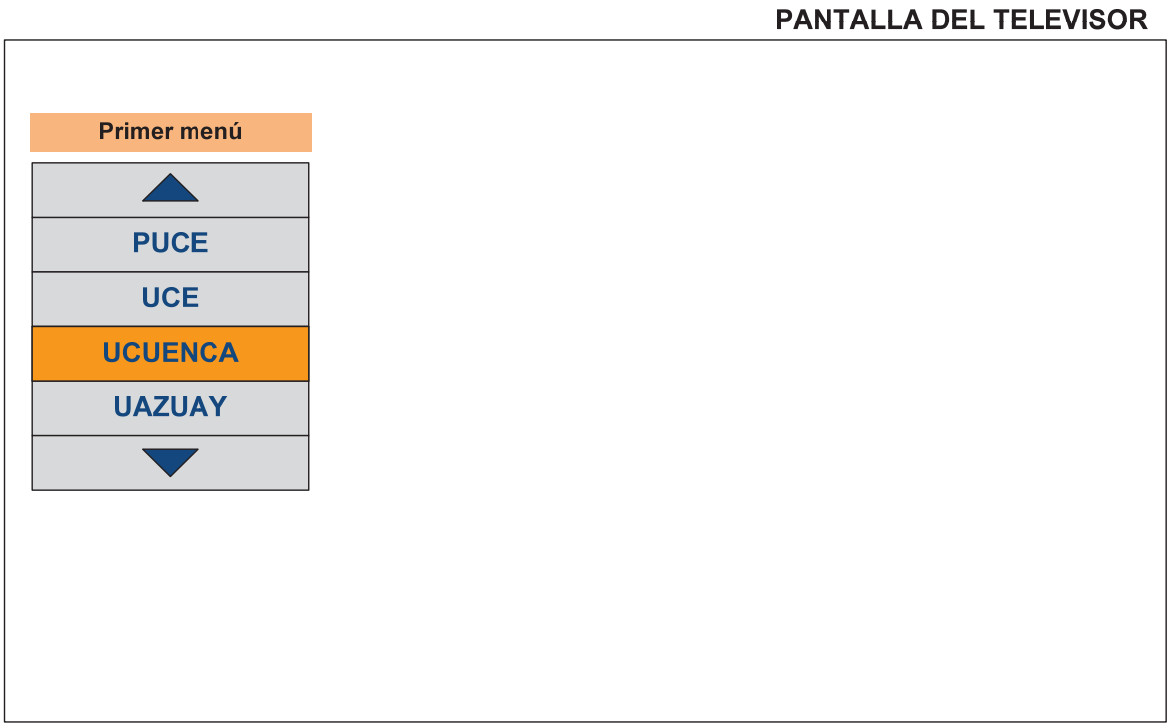


Figura 4.6 Bosquejo del diseño del primer menú

Por cada universidad, la aplicación presentará información acerca de aspectos como su fundación, misión, visión, lema, sitio web, contacto, ubicación, autoridades, oferta de pregrado y de postgrado. Para que el televidente pueda visualizar la información sobre un determinado aspecto, la aplicación dispondrá de un segundo menú en la parte central del televisor que se presentará después de que el televidente haya seleccionado una determinada universidad. Este segundo menú listará cada uno de los aspectos y estará conformado por dos filas y seis columnas, y se presentará mediante dos vistas conformadas por seis elementos cada una. En cada vista, el televidente tendrá a disposición la opción “Regresar” a través de la cual podrá volver al primer menú para seleccionar otra universidad. El botón AZUL detendrá la presentación de la aplicación.

La Figura 4.7 y la Figura 4.8 presentan un bosquejo del diseño del segundo menú, presentando la primera y la segunda vista del menú respectivamente, y su ubicación en la pantalla del televisor junto al primer menú.

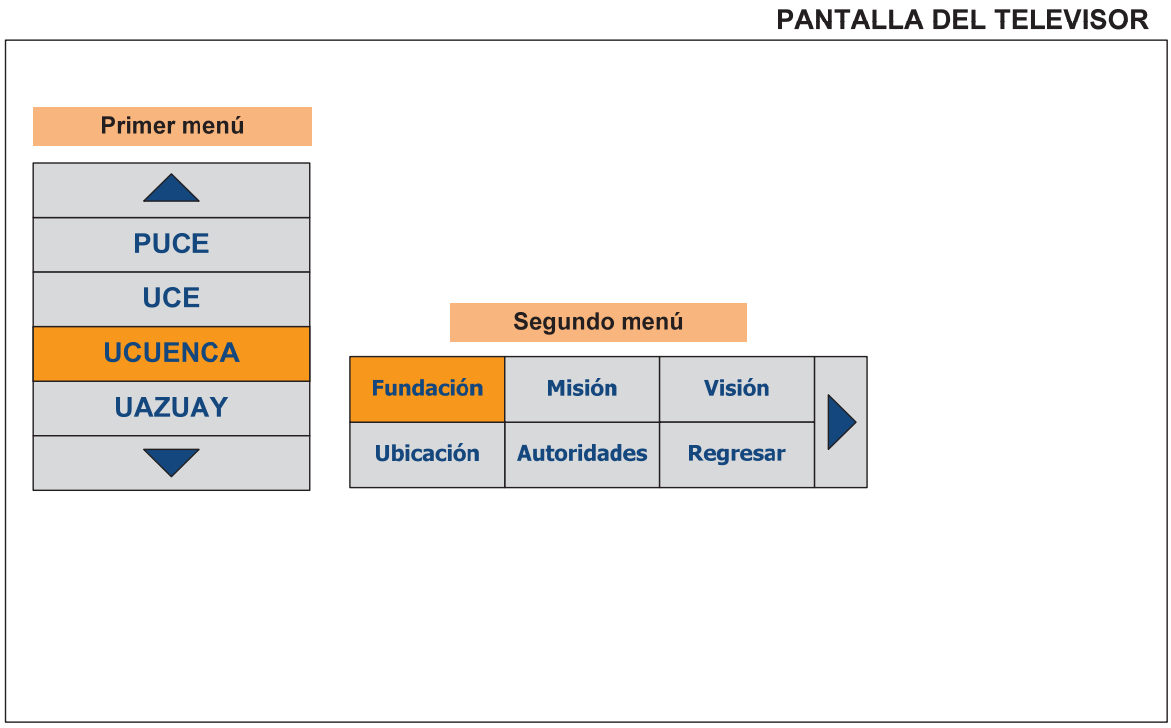


Figura 4.7 Bosquejo del diseño del segundo menú (primera vista)

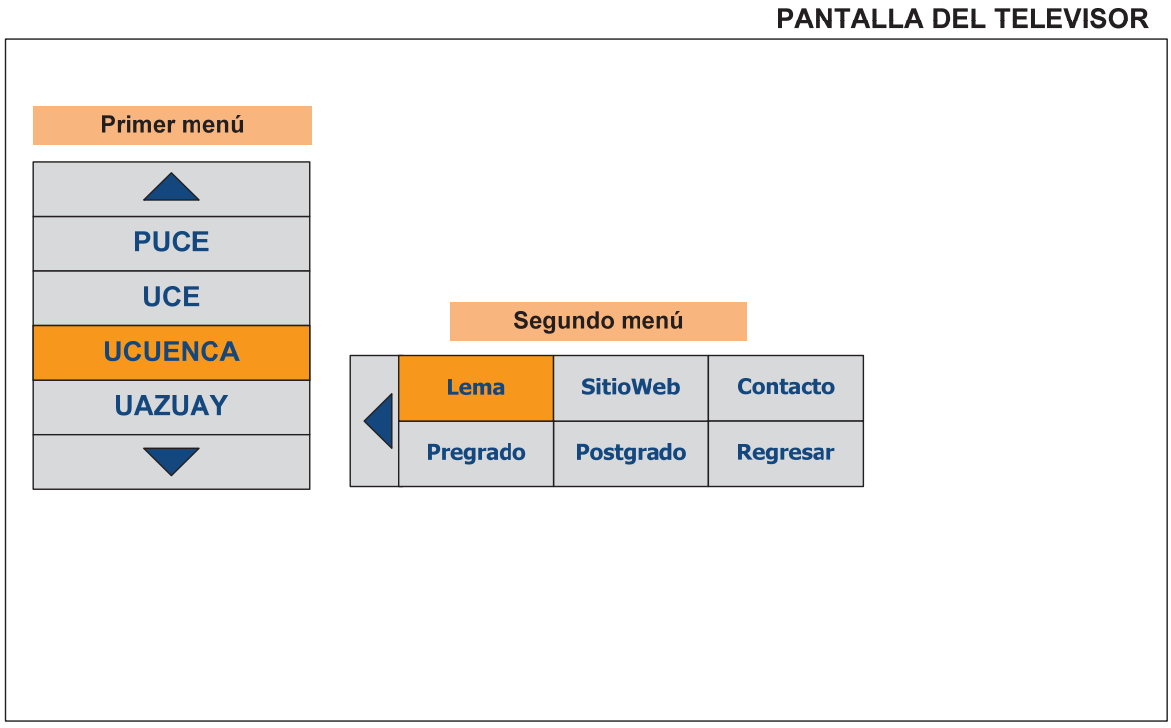


Figura 4.8 Bosquejo del diseño del segundo menú (segunda vista)

Como la misión, visión y oferta de pregrado y postgrado involucran una gran cantidad de texto, la información de estos aspectos será obtenida a través del

canal de retorno como texto plano. La información de los aspectos restantes se presentará mediante imágenes PNG con transparencia. Estas imágenes serán generadas, al igual que los *scripts* Lua que permiten obtener la información a través del canal de retorno, mediante la aplicación Mixer y estarán contenidas dentro del proyecto Composer de la aplicación interactiva. La Figura 4.9 presenta un bosquejo de la aplicación interactiva presentando información que fue obtenida a través del canal de retorno.

La aplicación interactiva presentará en primer lugar un ícono de interactividad en la parte superior derecha del televisor, para indicarle al televidente que la programación dispone de una aplicación, y cuando el televidente pulse el botón OK del control remoto se mostrará una imagen de fondo, el video de la programación redimensionado en un costado y el primer menú para que el usuario mire el listado con los acrónimos de las once universidades.

Al navegar entre cada uno de los acrónimos de las universidades listadas en el primer menú, se presentará al televidente en la parte inferior de la pantalla el escudo de la universidad y en la parte superior el significado de dicho acrónimo (un título con el nombre de la universidad). Al seleccionar una determinada universidad, se mostrará una fotografía del campus de la universidad seleccionada y el segundo menú que permite obtener mayor información de la misma. Si el televidente selecciona un aspecto cuya información es obtenida a través del canal de retorno, se presentará dicha información en una nueva pantalla con una segunda imagen de fondo y el texto debidamente justificado tratando de abarcar gran parte de la pantalla del televisor. El televidente podrá regresar a la pantalla principal pulsando el botón ROJO del control remoto. Por el contrario, si el televidente selecciona un aspecto cuya información está contenida en una imagen PNG, se presentará dicha imagen bajo el segundo menú.

La Figura 4.10 presenta un bosquejo general de la aplicación interactiva presentando al televidente información contenida en una imagen PNG.

Figura 4.9 Bosquejo de la aplicación interactiva presentando información que fue obtenida a través del canal de retorno

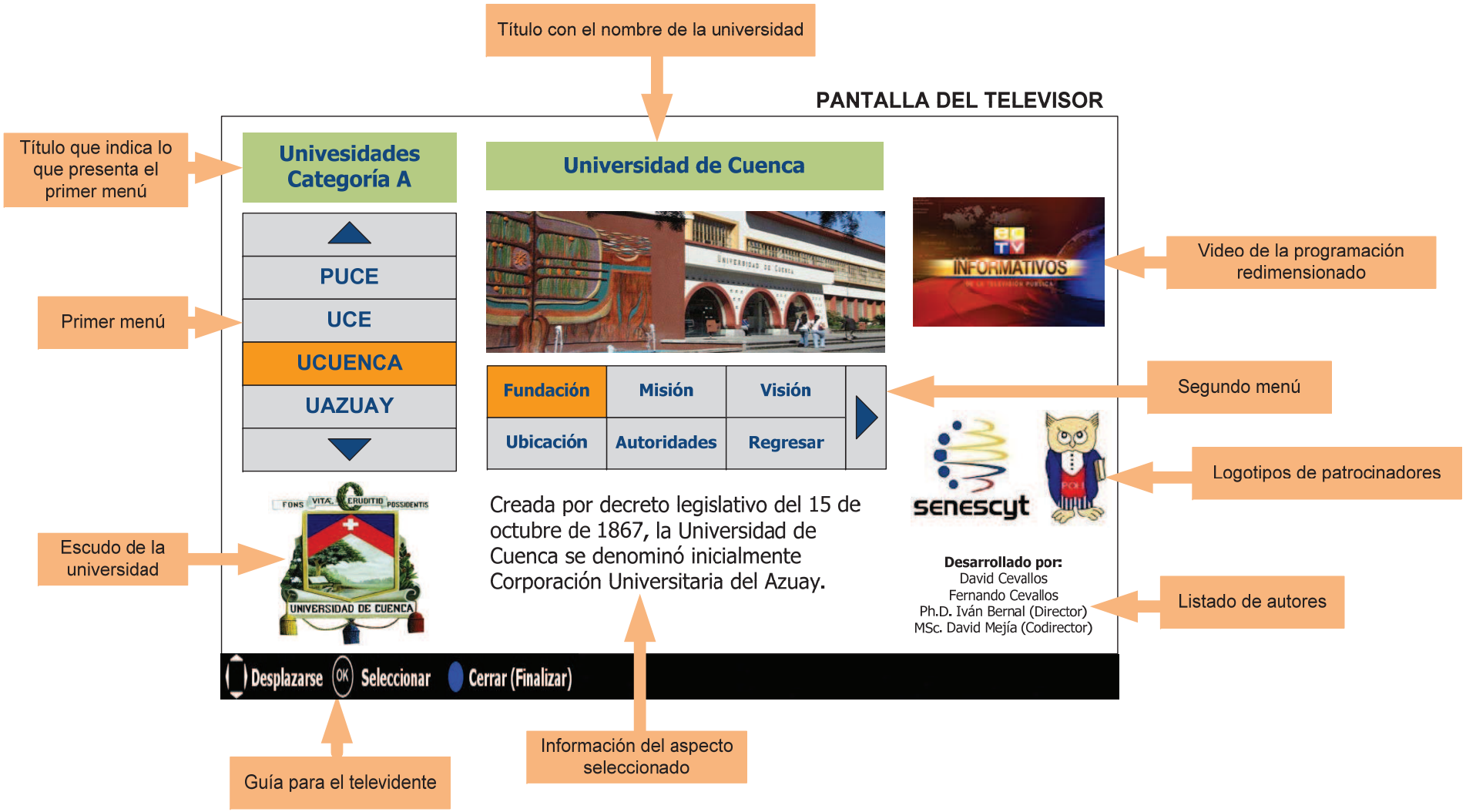


Figura 4.10 Bosquejo de la aplicación interactiva presentando información contenida en una imagen PNG

4.3 INGRESO DE INFORMACIÓN MEDIANTE LA APLICACIÓN CLIENTE (TEXTUAL DATA CREATOR)

Empleando la aplicación cliente Textual Data Creator, se puede hacer uso del servicio SOAP para generar una base de datos en SQL Server, así como las tablas en las que se almacenará la información para la primera aplicación interactiva.

Textual Data Creator maneja la información mediante la creación de temas y subtemas. Un tema corresponde a una base de datos que se genera para una aplicación interactiva que se está desarrollando. Un subtema corresponde a una tabla de una base de datos que se genera para almacenar información específica de la aplicación interactiva.

4.3.1 GENERACIÓN DEL TEMA (BASE DE DATOS)

Para el almacenamiento de la información se ha considerado la creación de una base de datos que tiene como nombre EducacionSuperior. En esta base datos se almacenará toda la información de la aplicación que se presentará al televidente.

4.3.2 GENERACIÓN DE SUBTEMAS (TABLAS)

Dado que la aplicación interactiva ha sido diseñada para presentar dos menús, en primer lugar se crean dos tablas para posteriormente almacenar la información a presentarse en cada uno de los menús. La primera tabla almacenará los acrónimos de las once universidades y será utilizada para el primer menú. La segunda tabla almacenará diversos aspectos de las universidades y será utilizada para el segundo menú.

Finalmente, por cada una de las once universidades se crea una tabla en la que se almacenará la información concerniente a cada universidad. La Tabla 4.7 detalla las tablas que se han creado, así como la información que almacenan y sus dimensiones.

Tabla 4.7 Tablas creadas para la primera aplicación interactiva

Nombre de la tabla	Información que almacena cada tabla	Dimensiones (filas x columnas)
ListaUniversidades	Acrónimo de cada universidad	11 x 1
Opciones	Aspectos para cada universidad	2 x 6
EPN	Información de la Escuela Politécnica Nacional	2 x 6
ESPE	Información de la Escuela Politécnica del Ejército	2 x 6
ESPOCH	Información de la Escuela Politécnica del Chimborazo	2 x 6
ESPOL	Información de la Escuela Politécnica del Litoral	2 x 6
PUCE	Información de la Pontificia Universidad Católica del Ecuador	2 x 6
UAZUAY	Información de la Universidad del Azuay	2 x 6
UCE	Información de la Universidad Central del Ecuador	2 x 6
UCUENCA	Información de la Universidad de Cuenca	2 x 6
USFQ	Información de la Universidad San Francisco de Quito	2 x 6
UTA	Información de la Universidad Técnica de Ambato	2 x 6
UTPL	Información de la Universidad Técnica Particular de Loja	2 x 6

La Figura 4.11 muestra el resultado de Textual Data Creator al generar la base de datos y sus tablas, las cuales están listas para ingresar información.

4.3.3 INGRESO DE INFORMACIÓN

Para ingresar la información en las tablas generadas se tienen dos alternativas: la primera alternativa es ingresar la información correspondiente a cada celda de una tabla escribiendo la información directamente en ella, es decir, ingresando la información de forma manual; y la segunda alternativa es hacer uso del *bot* de búsqueda para consultar en sitios web de confianza la información correspondiente a la celda, ofreciendo la posibilidad de editar la información encontrada por el *bot* como se requiera. Ambas alternativas pueden ser usadas indistintamente.

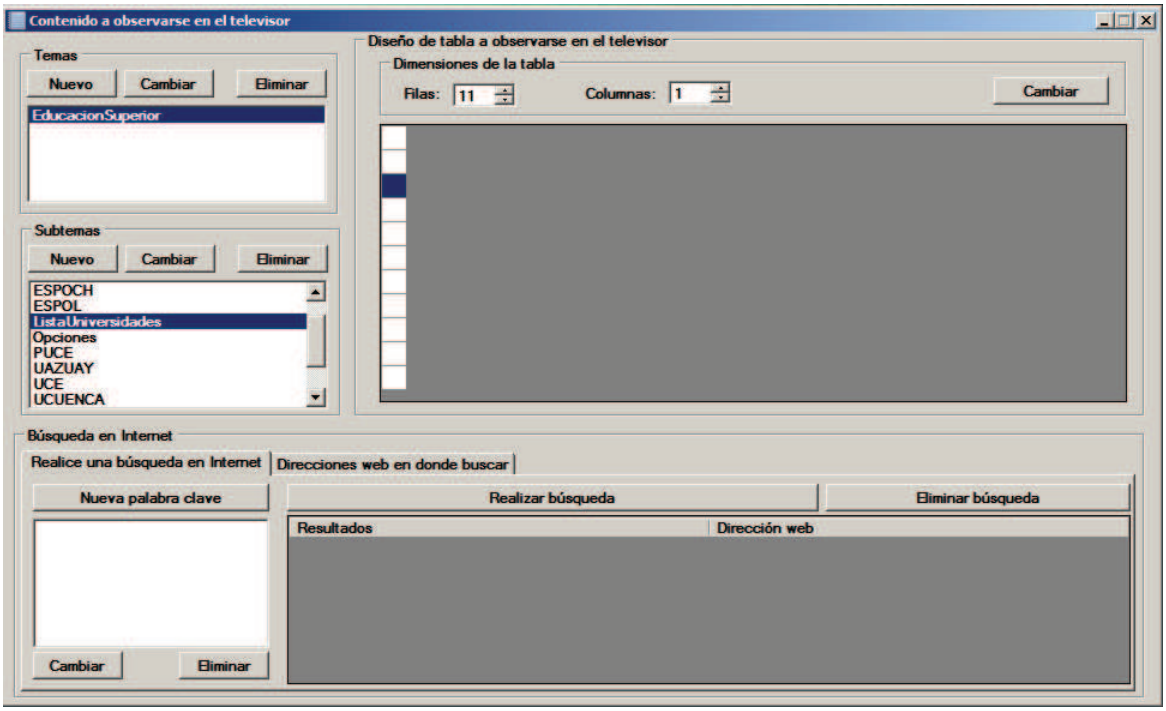


Figura 4.11 Base de datos y tablas generadas con Textual Data Creator

4.3.3.1 Ingreso manual de información

Dado que la primera tabla almacena los acrónimos de las once universidades y la segunda tabla los nombres de los aspectos que podrá seleccionar el televidente para cada universidad, se ha ingresado la información de las dos tablas de forma manual. Dentro de la tabla a editar se selecciona la celda en la cual se desea ingresar la información y se escribe sobre esta el texto de forma directa.

La Figura 4.12 muestra la tabla correspondiente al primer menú, mientras que en la Figura 4.13 se muestra la tabla para el segundo menú. En ambas tablas la información ha sido ingresada de forma manual.

4.3.3.2 Ingreso de información empleando el bot de búsqueda

Para ingresar la información de cada una de las universidades, es aconsejable hacer uso del *bot* de búsqueda para encontrar información como la misión, la visión y los nombres de las autoridades de cada universidad. Otros datos como los sitios web y la ubicación de las universidades pueden ser ingresados de forma manual.

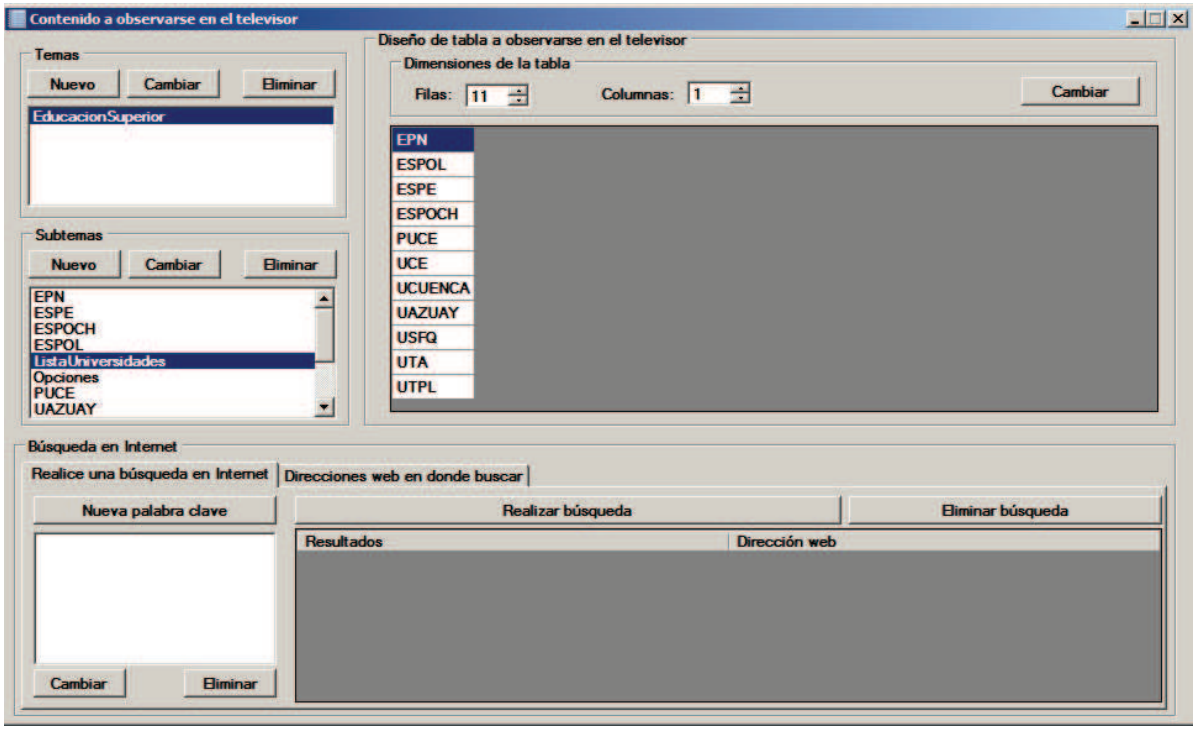


Figura 4.12 Ingreso de datos de forma manual en la tabla para el primer menú

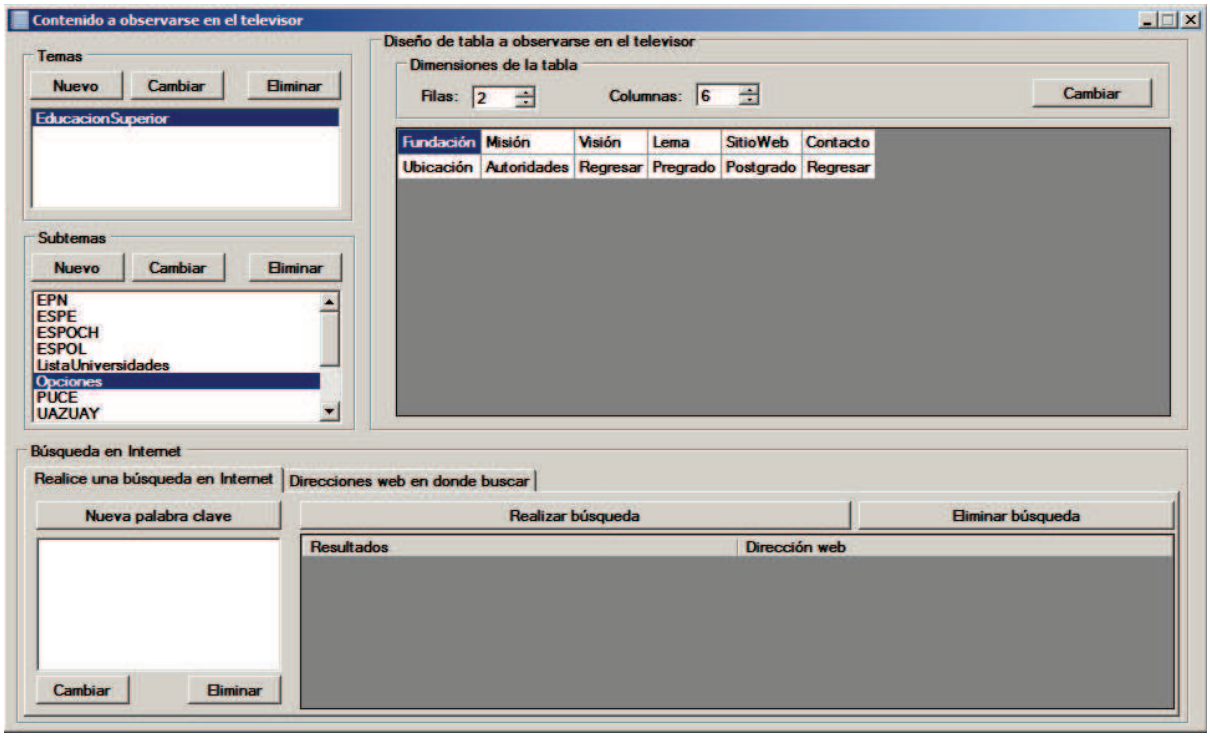


Figura 4.13 Ingreso de datos de forma manual en la tabla para el segundo menú

Para poder hacer uso del *bot* de búsqueda, se ha especificado como sitios web de confianza[74] los sitios web oficiales correspondientes a las once universidades, como se puede observar en la Figura 4.14.

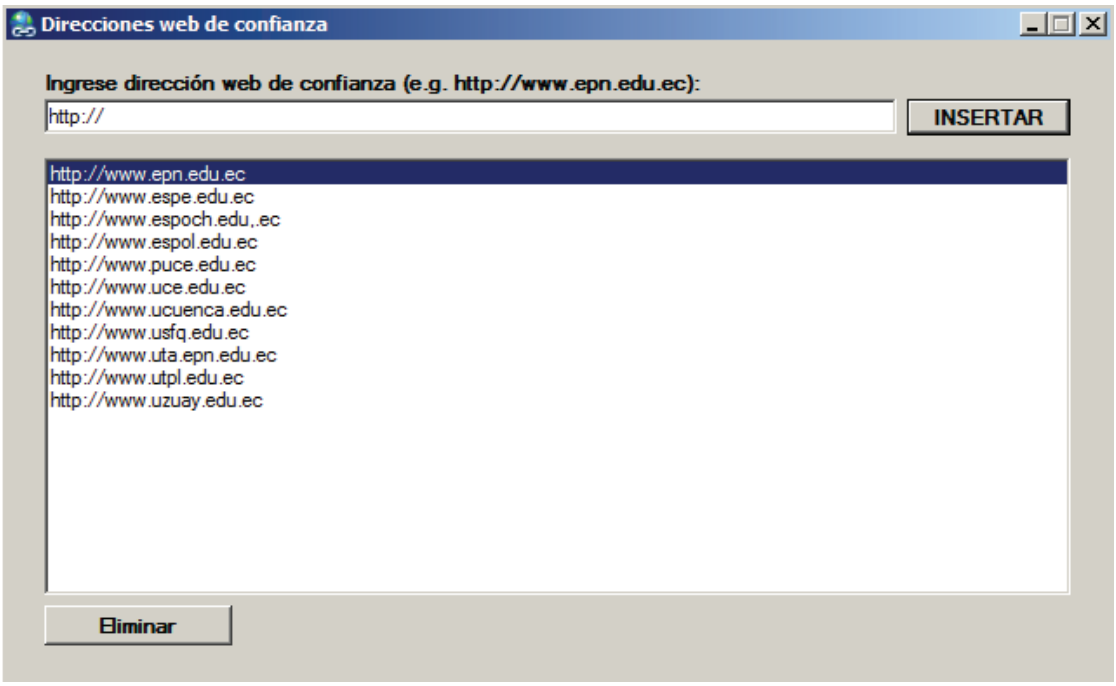


Figura 4.14 Ingreso de los sitios web correspondientes a las once universidades

Para obtener la información de cada universidad se emplea su sitio web oficial correspondiente.

Haciendo uso del *bot* de búsqueda se han realizado varias consultas (búsquedas) con las cuales se ha podido determinar la información a almacenarse en las celdas específicas.

La Figura 4.15 presenta los resultados obtenidos al realizar la búsqueda para obtener los nombres de rector y vicerrector de la Pontificia Universidad Católica de Quito (PUCE). Las palabras clave ingresadas para esta búsqueda son: `autoridades`, `rector` y `vicerrector`. La Figura 4.16 muestra la página web de la cual el *bot* de búsqueda obtuvo la información.

[74] Un sitio web de confianza es un sitio con información fidedigna. Lo recomendable es emplear el *bot* de búsqueda con sitios web oficiales, en este caso por ejemplo, se pueden emplear los sitios web oficiales de las universidades.

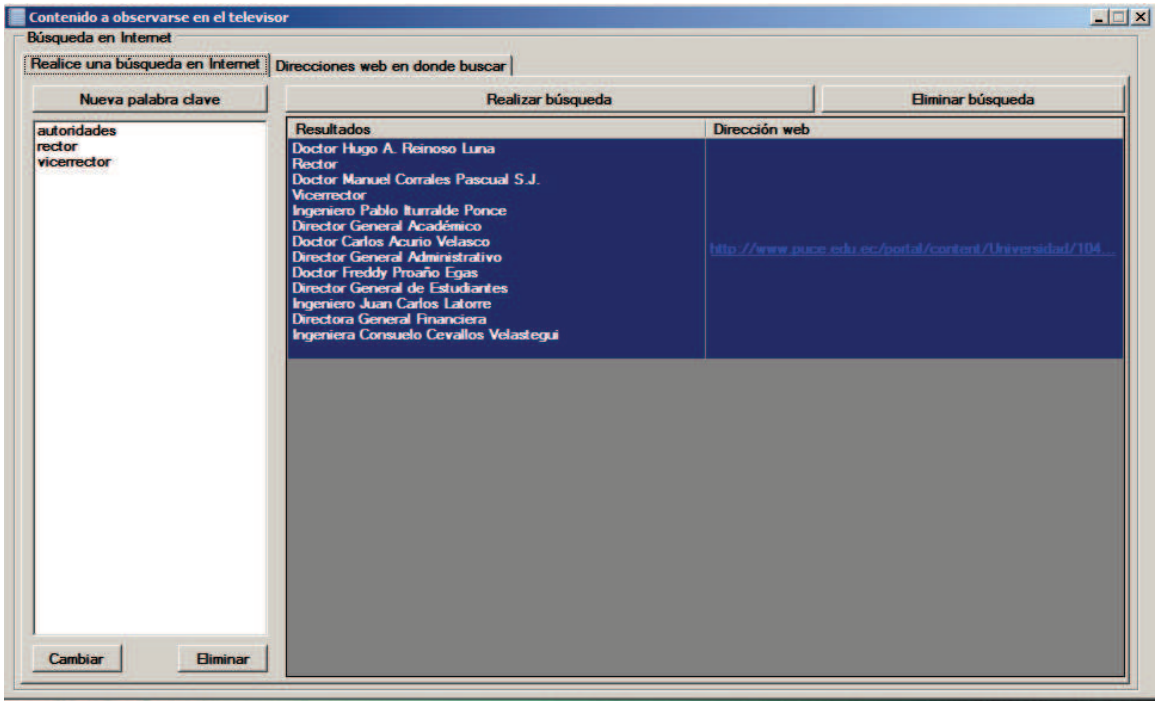


Figura 4.15 Búsqueda realizada para obtener los nombres de las autoridades de la PUCE

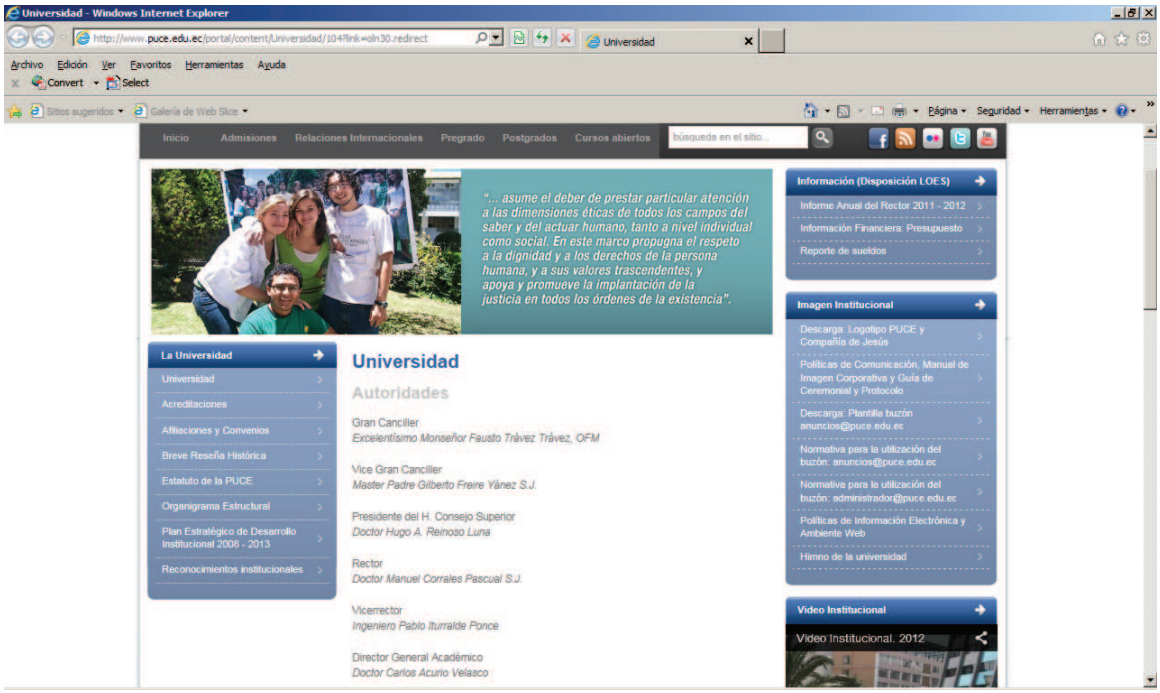


Figura 4.16 Página web de la cual el *bot* de búsqueda obtuvo los nombres de las autoridades de la PUCE

La Figura 4.17 presenta la búsqueda realizada para encontrar la misión de la Escuela Politécnica del Ejército (ESPE). Las palabras clave ingresadas para esta búsqueda son: `misión`, `principios` y `valores`. La Figura 4.18 muestra la página web de la cual el *bot* de búsqueda obtuvo la información.

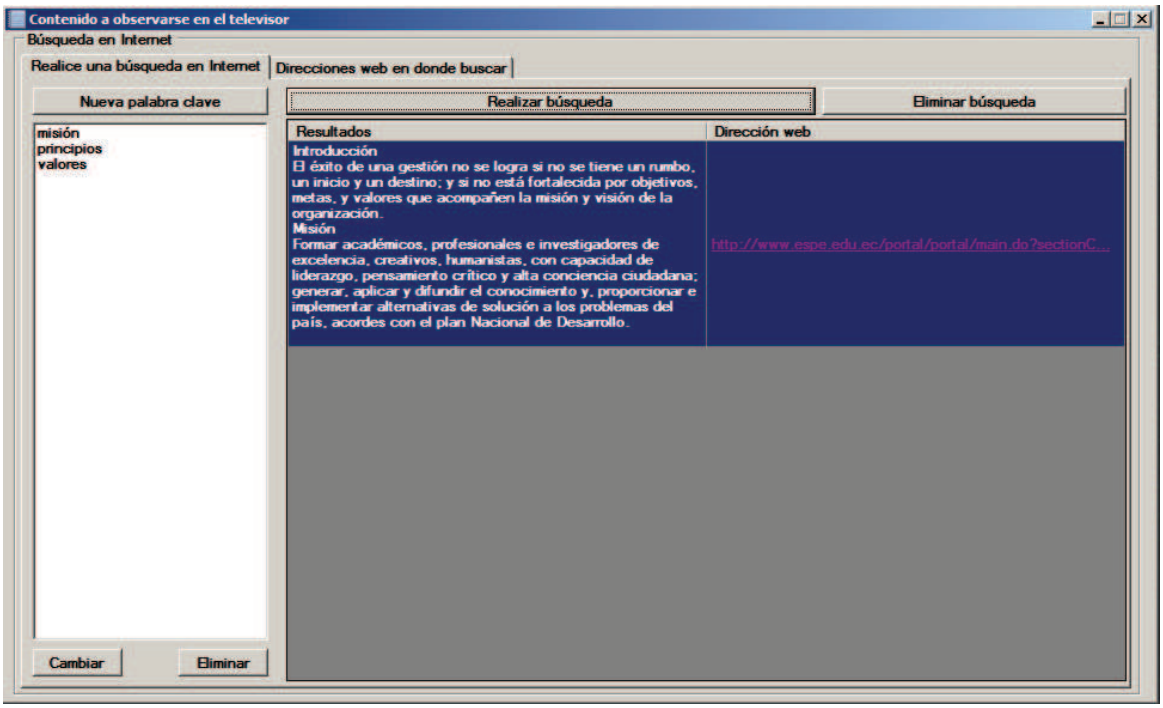


Figura 4.17 Búsqueda realizada para obtener la misión de la ESPE

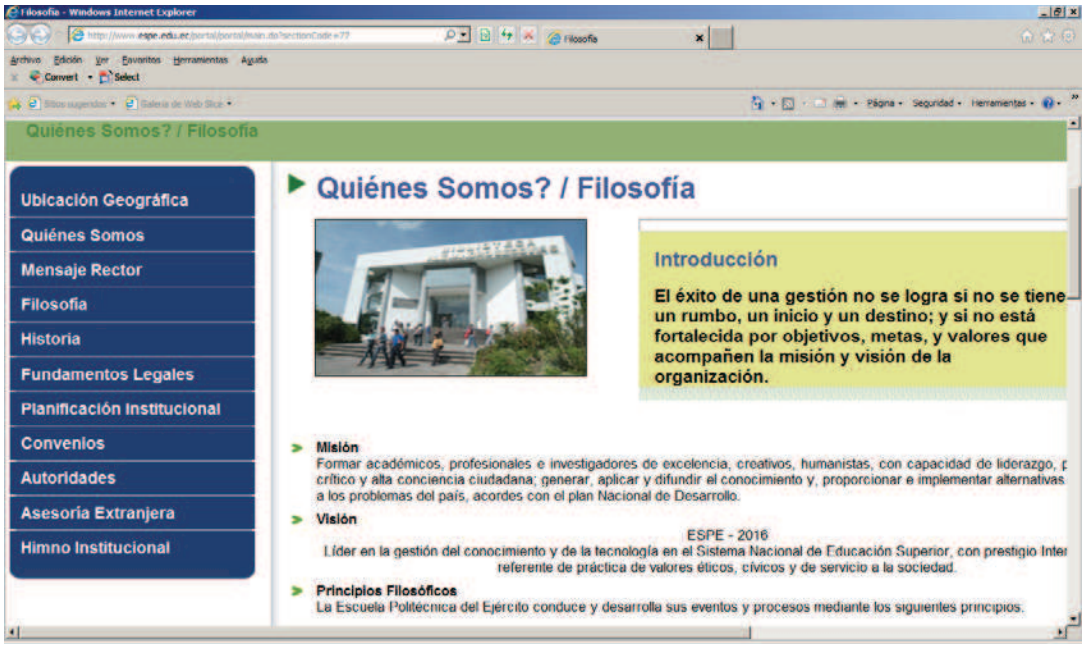


Figura 4.18 Página web de la cual el *bot* de búsqueda obtuvo la misión de la ESPE

Para información más compleja de encontrar, como por ejemplo la oferta de pregrado y/o de postgrado de una universidad, el *bot* de búsqueda podría presentar algunos resultados a partir de los cuales se puede construir la información a almacenarse en la base de datos.

Así por ejemplo, la Figura 4.19 muestra la búsqueda realizada para encontrar la información referente a la oferta de pregrado de la Universidad San Francisco de Quito (USFQ). La única palabra clave ingresada para esta búsqueda es `pregrado`. La Figura 4.20 muestra la primera página web encontrada por el *bot* de búsqueda que posee información acerca de la oferta de pregrado de esta universidad.

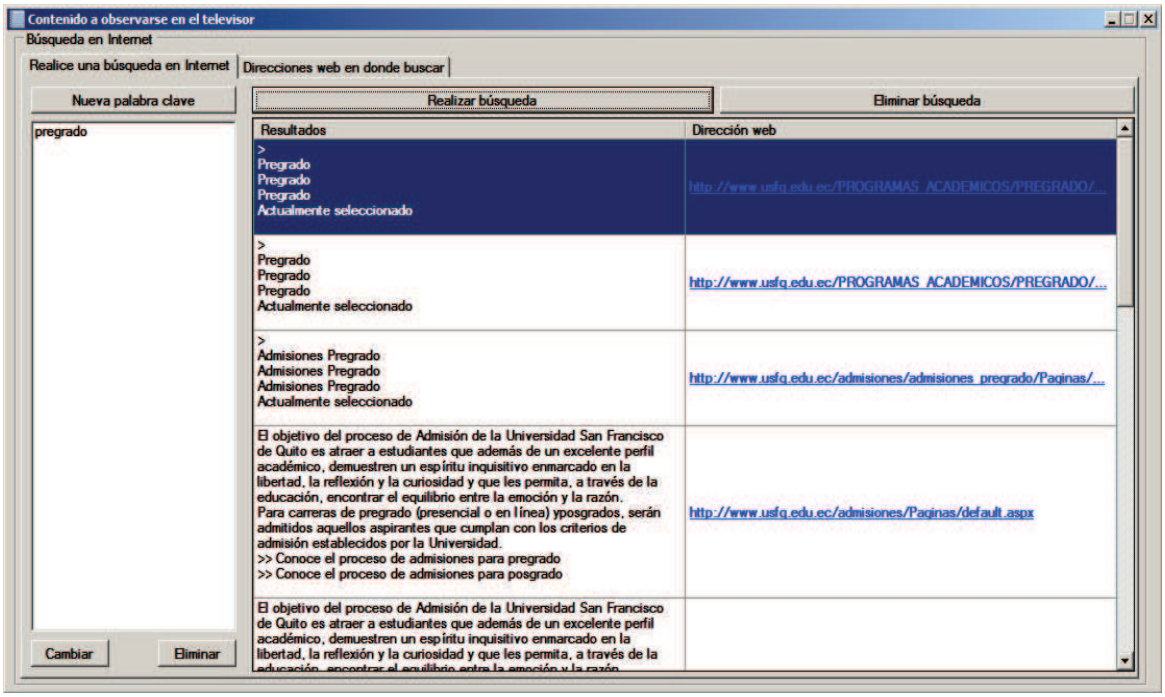


Figura 4.19 Búsqueda realizada para obtener la oferta de pregrado de la USFQ

4.3.4 RESULTADOS OBTENIDOS

Una vez generada la base de datos y sus tablas, y finalizado el ingreso de la información correspondiente de cada universidad, se obtiene una base de datos completa con la información a presentarse al televidente cuando se ejecute la aplicación interactiva, como se muestra en la Figura 4.21. Esta información será empleada posteriormente por la aplicación Mixer para la generación de los elementos media (imágenes PNG con transparencia y *scripts* Lua).

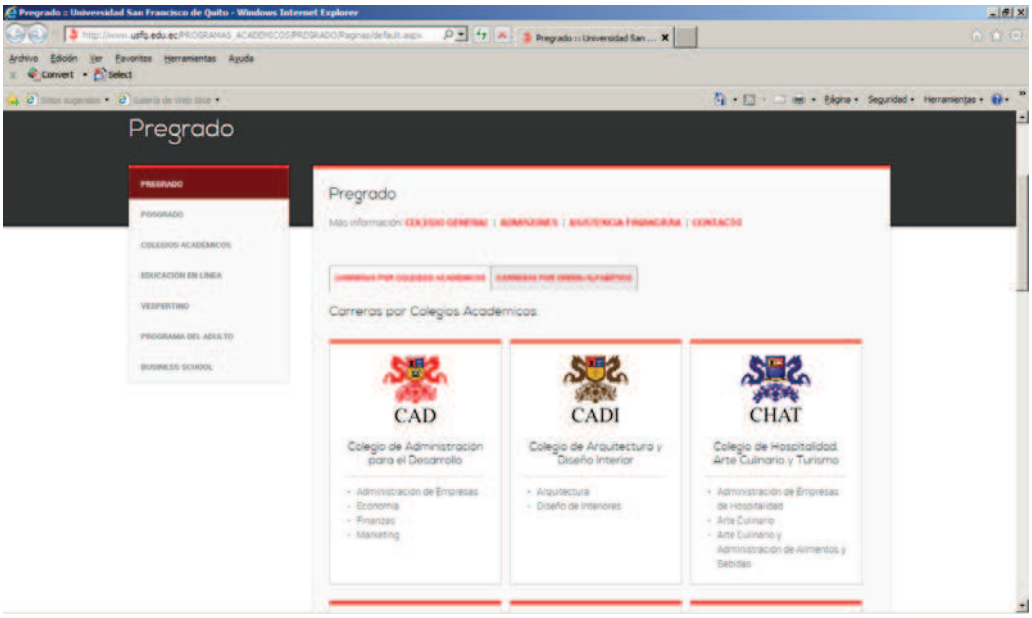


Figura 4.20 Página web correspondiente al primer resultado obtenido por el *bot* de búsqueda para la oferta de pregrado de la USFQ

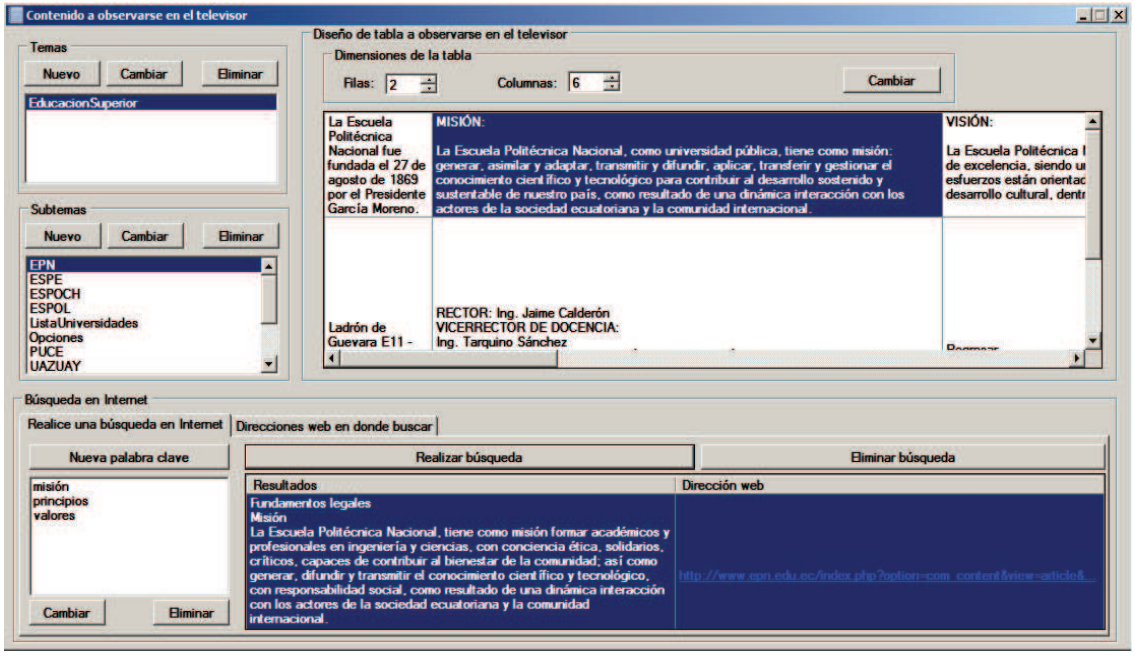


Figura 4.21 Base de datos y tablas con información ingresada

4.4 GENERACIÓN DE UNA APLICACIÓN INTERACTIVA SIN DATOS EMPLEANDO EL PLUG-IN MENU CREATOR

Empleando el IDE NCL Composer y el *plug-in* de generación automática de menús Menu Creator, se genera la aplicación interactiva sin datos, es decir, sin los elementos media con la información a presentarse al televidente.

4.4.1 GESTIÓN DE NOMBRES DE ELEMENTOS MEDIA

Dado que a partir de esta aplicación interactiva se desarrollará una segunda aplicación, para la gestión de los nombres de los elementos media dentro del código NCL se ha optado por hacer uso de un nombre genérico para cada una de las universidades. De esta manera, se evitará la tarea de renombrar a los elementos media al desarrollar la segunda aplicación interactiva. Así, cada una de las once universidades poseerá un identificador único: U1, U2, U3, …, U11.

Cada una de las universidades posee una imagen correspondiente a su escudo, un título con su nombre y una imagen de su campus universitario. Estas imágenes se almacenan en tres carpetas distintas dentro del proyecto Composer de la aplicación, haciendo uso del identificativo único para nombrar a cada imagen. La Tabla 4.8 presenta el identificativo empleado para cada universidad de la aplicación interactiva.

Tabla 4.8 Identificadores empleados para cada universidad de la primera aplicación interactiva

Identificador	Universidad
U1	Escuela Politécnica Nacional
U2	Escuela Superior Politécnica del Litoral
U3	Escuela Politécnica del Ejército
U4	Escuela Superior Politécnica del Chimborazo
U5	Pontificia Universidad Católica de Quito
U6	Universidad Central del Ecuador
U7	Universidad de Cuenca
U8	Universidad del Azuay
U9	Universidad San Francisco de Quito
U10	Universidad Técnica de Ambato
U11	Universidad Técnica Particular de Loja

4.4.2 CREACIÓN DE MENÚS Y OBTENCIÓN DE NODOS DE CONTEXTO

Mediante el *plug-in* Menu Creator, se crean los dos menús de la aplicación interactiva. La Figura 4.22 detalla las propiedades de diseño del primer menú, mientras que la Figura 4.23 presenta las del segundo menú.

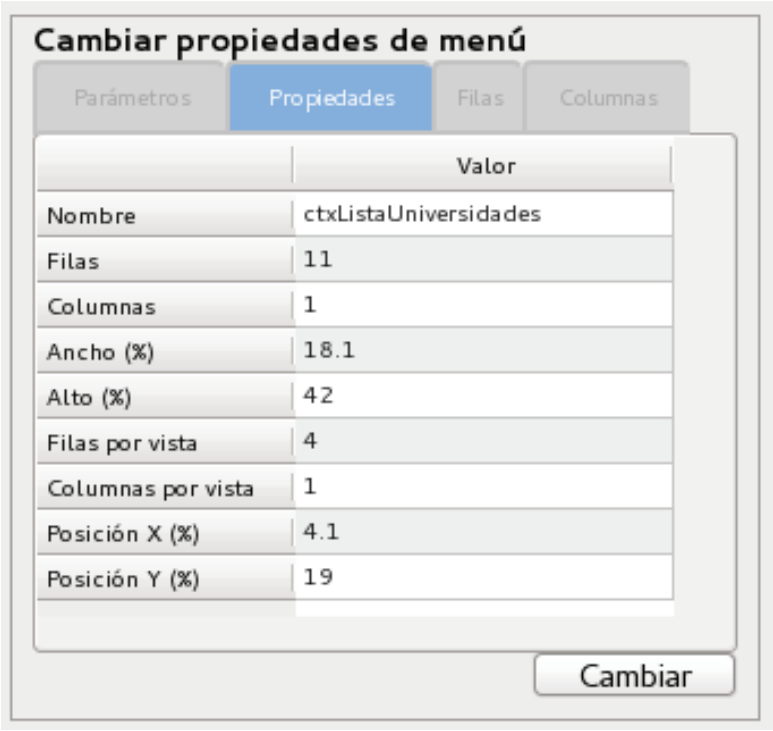


Figura 4.22 Propiedades de diseño del primer menú

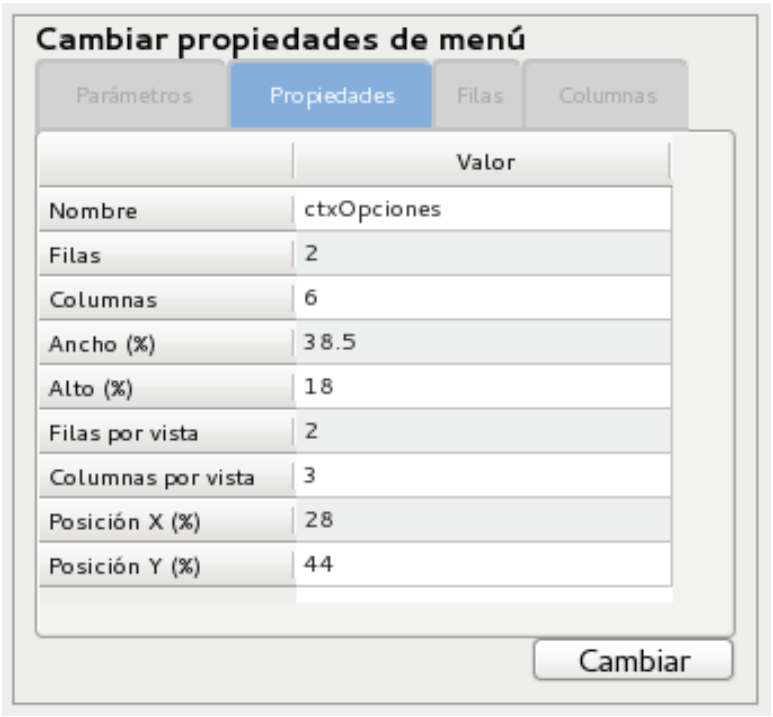


Figura 4.23 Propiedades de diseño del segundo menú

La Figura 4.24 presenta los dos menús diseñados de la aplicación interactiva graficados en el navegador interactivo de menús del *plug-in* Menu Creator.

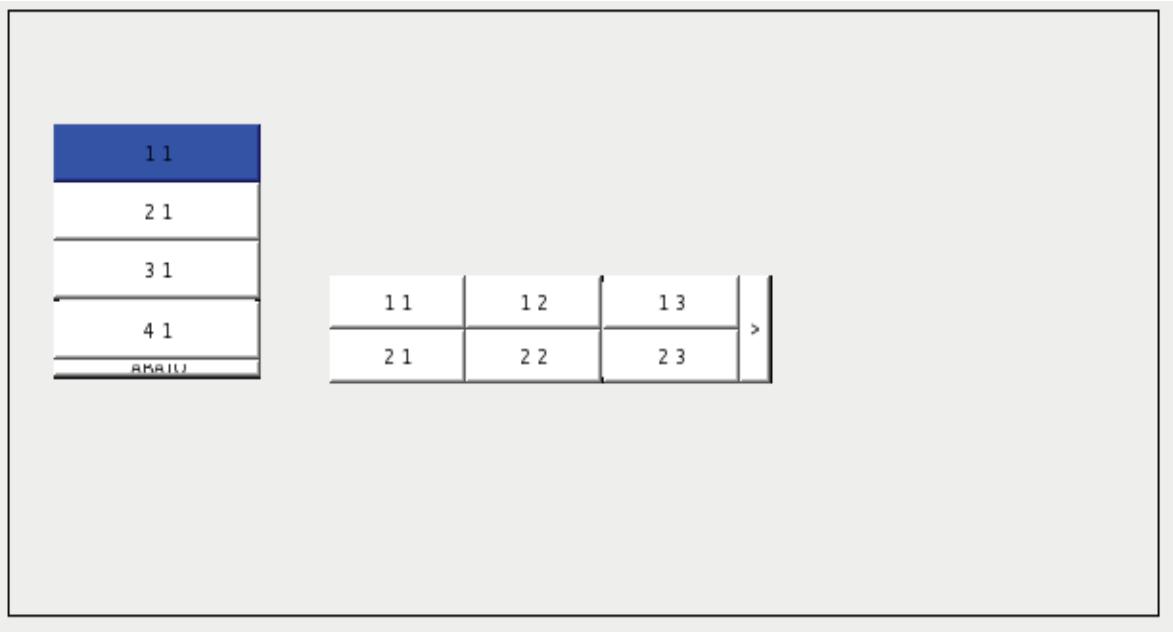


Figura 4.24 Menús diseñados en el navegador interactivo de menús del *plug-in* Menu Creator

Al generar el código NCL mediante el *plug-in* se obtienen dos nodos de contexto, cada uno de los cuales representa a un menú, como se puede apreciar en la Figura 4.25.

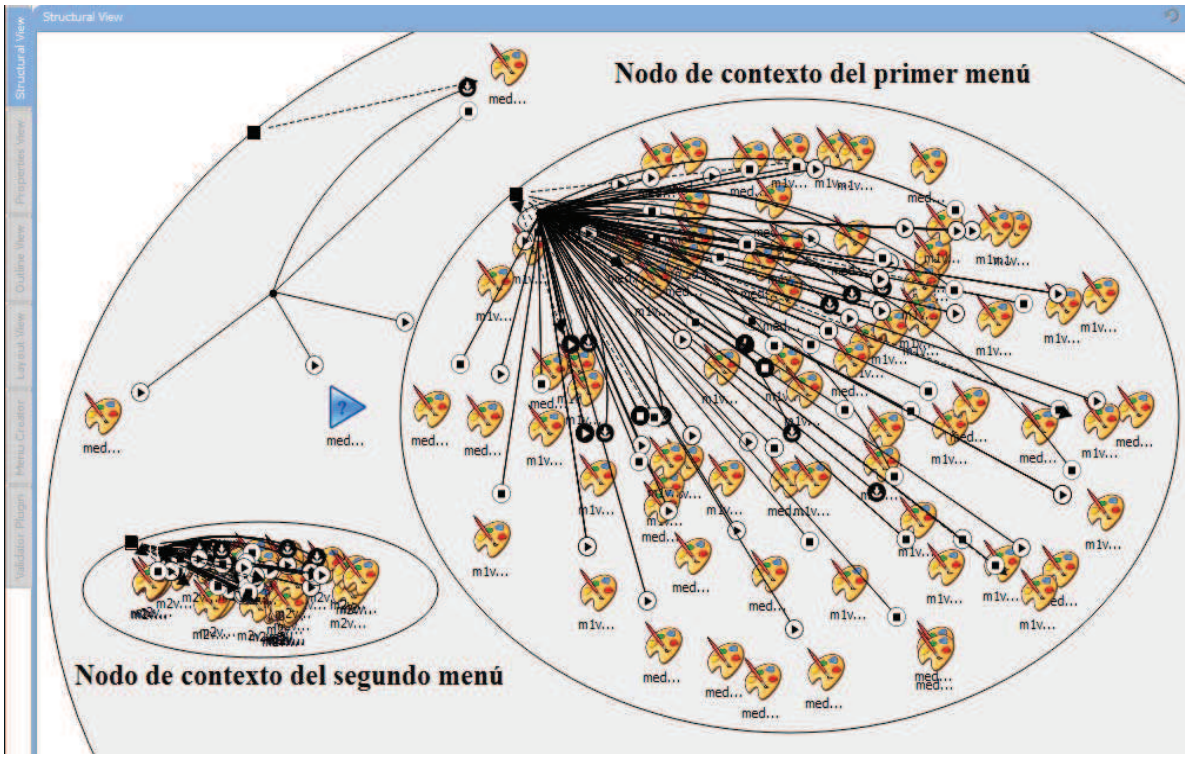


Figura 4.25 Nodos de contexto generados con el *plug-in* Menu Creator

4.4.3 DECLARACIÓN DE NODOS DE CONTENIDO

Dado que los elementos media serán posteriormente generados mediante la aplicación Mixer, en el desarrollo de la aplicación interactiva únicamente es necesario declarar los nodos de contenido referentes a dichos elementos media.

Para cada universidad, es necesario declarar diez nodos de contenido porque son diez los aspectos de cada universidad que se van a presentar al televidente. De estos diez aspectos, seis corresponden a imágenes PNG con transparencia mientras que cuatro corresponden a *scripts* Lua, como se detalla en la Tabla 4.9. En la propiedad `src` de cada nodo de contenido declarado se especifica el nombre del archivo y la ruta en donde la aplicación Mixer almacenará el elemento media a generarse.

Tabla 4.9 Presentación/Obtención de la información para cada aspecto

Aspecto	Presentación/Obtención de la información
fundación	imagen PNG
misión	*script* Lua
visión	*script* Lua
lema	imagen PNG
sitio web	imagen PNG
contacto	imagen PNG
ubicación	imagen PNG
autoridades	imagen PNG
pregrado	*script* Lua
postgrado	*script* Lua

El Código 4.1 presenta un ejemplo de los nodos de contenido declarados para presentar las imágenes PNG con transparencia correspondientes a la fundación de las once universidades.

Al igual que en el Código 4.1, se realiza el mismo procedimiento con el resto de los nueve aspectos. En el caso de los aspectos correspondientes a la misión, visión y oferta de pregrado y postgrado, la propiedad `src` del nodo se refiere a un *script* Lua en lugar de una imagen PNG. El Código 4.2 presenta un ejemplo con los nodos de contenido declarados para ejecutar los *scripts* Lua correspondientes a la misión de las once universidades.

```
<media id="medFundacionU1" src="Imagenes/Texto/Opciones/Fundacion/U1.png" descriptor="desRespuesta"/>
<media id="medFundacionU2" src="Imagenes/Texto/Opciones/Fundacion/U2.png" descriptor="desRespuesta"/>
<media id="medFundacionU3" src="Imagenes/Texto/Opciones/Fundacion/U3.png" descriptor="desRespuesta"/>
<media id="medFundacionU4" src="Imagenes/Texto/Opciones/Fundacion/U4.png" descriptor="desRespuesta"/>
<media id="medFundacionU5" src="Imagenes/Texto/Opciones/Fundacion/U5.png" descriptor="desRespuesta"/>
<media id="medFundacionU6" src="Imagenes/Texto/Opciones/Fundacion/U6.png" descriptor="desRespuesta"/>
<media id="medFundacionU7" src="Imagenes/Texto/Opciones/Fundacion/U7.png" descriptor="desRespuesta"/>
<media id="medFundacionU8" src="Imagenes/Texto/Opciones/Fundacion/U8.png" descriptor="desRespuesta"/>
<media id="medFundacionU9" src="Imagenes/Texto/Opciones/Fundacion/U9.png" descriptor="desRespuesta"/>
<media id="medFundacionU10" src="Imagenes/Texto/Opciones/Fundacion/U10.png" descriptor="desRespuesta"/>
<media id="medFundacionU11" src="Imagenes/Texto/Opciones/Fundacion/U11.png" descriptor="desRespuesta"/>
```

Código 4.1 Nodos de contenido declarados para presentar la información de la fundación de las once universidades

```
<media id="medMisionU1" src="Imagenes/Texto/Opciones/Mision/U1.lua" descriptor="desRetorno"/>
<media id="medMisionU2" src="Imagenes/Texto/Opciones/Mision/U2.lua" descriptor="desRetorno"/>
<media id="medMisionU3" src="Imagenes/Texto/Opciones/Mision/U3.lua" descriptor="desRetorno"/>
<media id="medMisionU4" src="Imagenes/Texto/Opciones/Mision/U4.lua" descriptor="desRetorno"/>
<media id="medMisionU5" src="Imagenes/Texto/Opciones/Mision/U5.lua" descriptor="desRetorno"/>
<media id="medMisionU6" src="Imagenes/Texto/Opciones/Mision/U6.lua" descriptor="desRetorno"/>
<media id="medMisionU7" src="Imagenes/Texto/Opciones/Mision/U7.lua" descriptor="desRetorno"/>
<media id="medMisionU8" src="Imagenes/Texto/Opciones/Mision/U8.lua" descriptor="desRetorno"/>
<media id="medMisionU9" src="Imagenes/Texto/Opciones/Mision/U9.lua" descriptor="desRetorno"/>
<media id="medMisionU10" src="Imagenes/Texto/Opciones/Mision/U10.lua" descriptor="desRetorno"/>
<media id="medMisionU11" src="Imagenes/Texto/Opciones/Mision/U11.lua" descriptor="desRetorno"/>
```

Código 4.2 Nodos de contenido declarados para presentar la información de la misión de las once universidades

4.4.4 CREACIÓN DE NODOS SWITCH

Para poder determinar la información que debe presentarse cuando el televidente seleccione una determinada opción, sea del primer menú o del segundo menú, se recurre al uso de dos variables de control que almacenarán un valor numérico correspondiente a la universidad seleccionada del primer menú y el aspecto seleccionado del segundo menú.

Cada vez que el televidente seleccione una universidad, en una variable se almacena el número de la universidad seleccionada (comprendido entre 1 y 11).

Cuando se seleccione una opción del segundo menú, con los aspectos de la universidad elegida, al igual que en el caso anterior, se almacena en una variable un número que representa la opción seleccionada por el televidente, como

sugiere la Figura 4.26. Los botones “Regresar” no requieren almacenar ningún valor ya que no aportan información.

Fundación (1)	**Misión (2)**	**Visión (3)**	**Lema (4)**	**Sitio web (5)**	**Contacto (6)**
Ubicación (7)	**Autoridades (8)**	**Regresar**	**Pregrado (9)**	**Postgrado (10)**	**Regresar**

Figura 4.26 Valor numérico a almacenarse en una variable por cada opción del segundo menú

Para la evaluación de las dos variables que determinan la universidad seleccionada por el televidente y el aspecto de dicha universidad a presentarse, se ha creado un nodo *switch* principal que determina cuál fue el aspecto seleccionado, y una serie de nodos *switch* anidados dentro del nodo *switch* principal, los cuales determinan la universidad escogida para presentar la información correspondiente al aspecto seleccionado por el televidente.

El Código 4.3 muestra parte del código NCL del nodo *switch* principal (`swtSelector`) y del nodo *switch* secundario (`swtFundacion`) empleado para presentar la información de la fundación de cada universidad. Para el caso de la información que se obtiene mediante el canal de retorno, como por ejemplo la misión de cada universidad, el nodo *switch* secundario correspondiente a cada aspecto (por ejemplo `swtMision`) se anida dentro de un nodo de contexto (por ejemplo `ctxMision`) para poder abrir, simultáneamente con la información a presentarse, una segunda imagen de fondo que permita visualizar de mejor manera el texto que se reciba.

Por otro lado, en la aplicación interactiva es necesario otro nodo *switch* que permita evaluar la universidad seleccionada para así poder presentar la imagen de su campus universitario correspondiente. Este nodo *switch* se ejecuta al presentarse el segundo menú, mediante una puerta (*port*) creada en el nodo de contexto del menú y asociada con el nodo *switch*. El Código 4.4 presenta el código NCL de este nodo *switch*.

```
<!-- Evalúa la opción escogida -->
<switch id="swtSelector">
    <!-- Evalúa la universidad escogida y abre su fundación -->
    <switch id="swtFundacion">
        <media id="medFundacionU1" src="Imagenes/Texto/Opciones/Fundacion/U1.png" descriptor="desRespuesta"/>
        .....
        .....
        <media id="medFundacionU11" src="Imagenes/Texto/Opciones/Fundacion/U11.png" descriptor="desRespuesta"/>

        <bindRule rule="rul1" constituent="medFundacionU1"/>
        .....
        .....
        <bindRule rule="rul11" constituent="medFundacionU11"/>
    </switch>

    <!-- Nodo de contexto para información sobre la misión de las universidades -->
    <context id="ctxMision">
        <media id="medFRMision" src="Imagenes/Fondos/retorno.png" descriptor="desFondoRetorno"/>
        <port id="porFRMision" component="medFRMision"/>
        <port id="porMision" component="swtMision"/>
        <!-- Evalúa la universidad escogida y abre su misión -->
        <switch id="swtMision">
            <media id="medMisionU1" src="Imagenes/Texto/Opciones/Mision/U1.lua" descriptor="desRetorno"/>
            .....
            .....
            <media id="medMisionU11" src="Imagenes/Texto/Opciones/Mision/U11.lua" descriptor="desRetorno"/>

            <bindRule rule="rul1" constituent="medMisionU1"/>
            .....
            .....
            <bindRule rule="rul11" constituent="medMisionU11"/>
        </switch>
    </context>
    .....
    .....
    .....
    <bindRule rule="rulOpcion1" constituent="swtFundacion"/>
    .....
    .....
    <bindRule rule="rulOpcion10" constituent="ctxPostgrado"/>
</switch>
```

Código 4.3 Nodo *switch* principal y nodos *switch* secundarios anidados para presentar la información de la fundación y misión de las universidades

```
<!-- Abre la imagen del campus universitario -->
<switch id="swtCampus">
    <media id="medCampusU1" src="Imagenes/Campus/U1.jpg" descriptor="desCampus"/>
    <media id="medCampusU2" src="Imagenes/Campus/U2.jpg" descriptor="desCampus"/>
    <media id="medCampusU3" src="Imagenes/Campus/U3.jpg" descriptor="desCampus"/>
    <media id="medCampusU4" src="Imagenes/Campus/U4.jpg" descriptor="desCampus"/>
    <media id="medCampusU5" src="Imagenes/Campus/U5.jpg" descriptor="desCampus"/>
    <media id="medCampusU6" src="Imagenes/Campus/U6.jpg" descriptor="desCampus"/>
    <media id="medCampusU7" src="Imagenes/Campus/U7.jpg" descriptor="desCampus"/>
    <media id="medCampusU8" src="Imagenes/Campus/U8.jpg" descriptor="desCampus"/>
    <media id="medCampusU9" src="Imagenes/Campus/U9.jpg" descriptor="desCampus"/>
    <media id="medCampusU10" src="Imagenes/Campus/U10.jpg" descriptor="desCampus"/>
    <media id="medCampusU11" src="Imagenes/Campus/U11.jpg" descriptor="desCampus"/>
    <bindRule rule="rul1" constituent="medCampusU1"/>
    <bindRule rule="rul2" constituent="medCampusU2"/>
    <bindRule rule="rul3" constituent="medCampusU3"/>
    <bindRule rule="rul4" constituent="medCampusU4"/>
    <bindRule rule="rul5" constituent="medCampusU5"/>
    <bindRule rule="rul6" constituent="medCampusU6"/>
    <bindRule rule="rul7" constituent="medCampusU7"/>
    <bindRule rule="rul8" constituent="medCampusU8"/>
    <bindRule rule="rul9" constituent="medCampusU9"/>
    <bindRule rule="rul10" constituent="medCampusU10"/>
    <bindRule rule="rul11" constituent="medCampusU11"/>
</switch>
```

Código 4.4 Nodo *switch* empleado para abrir la imagen del campus universitario de una determinada universidad seleccionada

4.4.5 INSERCIÓN DE ENLACES NCL

Para presentar el escudo y el título con el nombre de cada universidad mientras el televidente navega por cada uno de los elementos del primer menú, así como para detener la presentación de estos elementos, se han insertado, haciendo uso del módulo de enlaces del *plug-in* Menu Creator, dos enlaces NCL por cada opción del primer menú.

El Código 4.5 presenta los dos enlaces NCL creados para la primera opción del primer menú. El primer enlace es del tipo `onBeginSet_varStart` y permite, al activarse el foco del elemento, presentar al televidente el escudo y el nombre de la universidad correspondiente al primer elemento del menú. Este enlace NCL a la vez permite establecer el valor numérico de la variable de control para la universidad seleccionada. El segundo enlace NCL es del tipo `onEndStop` y permite detener la presentación del escudo y del título con el nombre de la universidad cuando el foco del primer elemento se detiene.

```
<!-- Abrir el escudo y título de la univesidad al activarse el foco -->
<link id="lnkLogoU1Ap" xconnector="conn#onBeginSet_varStart">
    <bind role="onBegin" component="m1f1c1k1m"/>
    <bind role="start" component="medLogoU1"/>
    <bind role="start" component="medTituloU1"/>
    <bind role="set" component="rvarUniversidad" interface="universidad">
        <bindParam name="var" value="1"/>
    </bind>
</link>

<!-- Cerrar el escudo y título de la univesidad al desactivarse el foco -->
<link id="lnkLogoU1Des" xconnector="conn#onEndStop">
    <bind role="onEnd" component="m1f1c1k1m"/>
    <bind role="stop" component="medLogoU1"/>
    <bind role="stop" component="medTituloU1"/>
</link>
```

Código 4.5 Enlaces NCL para presentar el escudo y título del nombre de la universidad de la primera opción del primer menú

Con el fin de controlar la interaccción entre ambos menús, a nivel del nodo de contexto principal (`body`) se crean dos enlaces NCL. El primer enlace NCL es de tipo `onPauseStart` y permite presentar el segundo menú cuando se pausa la presentación del primero. El segundo enlace NCL es de tipo `onEndResume` y

permite reanudar la presentación del primer menú cuando se detiene la presentación del segundo. El Código 4.6 presenta ambos enlaces NCL.

```
<!-- Arranca el menú Opciones al pausarse el menú Lista Universidades -->
<link id="lnkArrancarOpciones" xconnector="conn#onPauseStart">
    <bind role="onPause" component="ctxListaUniversidades"/>
    <bind role="start" component="ctxOpciones"/>
</link>

<!-- Resume el menú Lista de Universidades al terminar el menú Opciones -->
<link id="lnkArrancarListaU" xconnector="conn#onEndResume">
    <bind role="onEnd" component="ctxOpciones"/>
    <bind role="resume" component="ctxListaUniversidades"/>
</link>
```

Código 4.6 Enlaces NCL para gestionar la interacción entre los dos menús

Para cada uno de los elementos del primer menú es necesario añadir, mediante el módulo de inserción de enlaces NCL del *plug-in* Menu Creator, un enlace NCL de tipo `onKeySelectionPause` que permita pausar la presentación del primer menú cuando el televidente selecciona el elemento. En el Código 4.7 se detalla este enlace NCL para el primer elemento del primer menú.

```
<!-- Pausa el menú Lista Universidades lo que produce que se muestren las opciones a seleccionarse -->
<link id="lnkU1Entrar" xconnector="conn#onKeySelectionPause">
    <bind role="onSelection" component="m1f1c1k1m">
        <bindParam name="keyCode" value="ENTER"/>
    </bind>
    <bind role="pause" component="ctxListaUniversidades"/>
</link>
```

Código 4.7 Ejemplo de enlace NCL para pausar el primer menú cuando el televidente selecciona el primer elemento

Para cada uno de los elementos del segundo menú, se crea mediante el *plug-in* Menu Creator, un enlace NCL de tipo `onKeySelectionStopSet_varStart` que ejecuta el nodo *switch* principal para presentar la información referente al aspecto seleccionado por el televidente. Este enlace NCL a la vez, permite establecer el valor numérico de la variable de control para el aspecto seleccionado. Por otro lado, es necesario un enlace NCL de tipo `onKeySelectionStop` para que, cuando el televidente presione el botón ROJO

del control remoto, se detenga la presentación de la información que está actualmente visualizando mediante la detención de la ejecución del nodo *switch* principal. El Código 4.8 muestra el enlace NCL creado para presentar la información sobre la fundación de una universidad seleccionada por el televidente, y el enlace NCL que detiene la ejecución del nodo *switch* principal.

```
<!-- Abre la información seleccionada -->
<link id="lnkFundacion" xconnector="conn#onKeySelectionStopSet_varStart">
    <bind role="onSelection" component="m2f1c1k1m">
        <bindParam name="keyCode" value="ENTER"/>
    </bind>
    <bind role="stop" component="rswtSelector"/>
    <bind role="set" component="varOpciones" interface="opcion">
        <bindParam name="var" value="1"/>
    </bind>
    <bind role="start" component="rswtSelector"/>
</link>

<!-- Cierra la información actual -->
<link id="lnkCerrarInfo" xconnector="conn#onKeySelectionStop">
    <bind role="onSelection" component="medIndicacionesOpciones">
        <bindParam name="keyCode" value="RED"/>
    </bind>
    <bind role="stop" component="rswtSelector"/>
</link>
```

Código 4.8 Enlaces NCL para la gestión del segundo menú

Finalmente, mediante el *plug-in* Menu Creator se crea para cada uno de los elementos del segundo menú con la opción “Regresar”, un enlace NCL de tipo `onKeySelectionStop` para detener la presentación del segundo menú al ser seleccionado el elemento. El Código 4.9 muestra el enlace NCL creado para uno de estos elementos.

```
<!-- Regresa al menú Lista Universidades -->
<link id="lnkRegresar1" xconnector="conn#onKeySelectionStop">
    <bind role="onSelection" component="m2f2c3k1m">
        <bindParam name="keyCode" value="ENTER"/>
    </bind>
    <bind role="stop" component="ctxOpciones"/>
</link>
```

Código 4.9 Enlace NCL para detener la presentación del segundo menú al seleccionar el elemento “Regresar”

4.4.6 DECLARACIÓN DE METADATOS

La inserción de metadatos a través de la etiqueta `<meta>` permite asociar la información ingresada en la base de datos con la estructura estática (menús sin datos) creada en NCL Composer.

Para poder determinar la información a presentarse en cada uno de los menús, se han declarado dos metadatos de tipo uno, como se puede apreciar en el Código 4.10. La propiedad `name` de cada metadato indica el `id` del nodo de contexto de cada menú, y la propiedad `content` indica el nombre de la tabla de la base de datos con la que la aplicación Mixer realizará el proceso de asociación.

```
<meta name="ctxListaUniversidades" content="ListaUniversidades"/>
<meta name="ctxOpciones" content="Opciones"/>
```

Código 4.10 Declaración de metadatos de tipo uno

Para el caso de la información a ser transmitida mediante la interfaz de aire, es decir, junto con la aplicación interactiva, es necesario declarar por cada universidad los metadatos de tipo dos correspondientes a los aspectos de la fundación, ubicación, autoridades, lema, sitio web e información de contacto.

El Código 4.11 presenta la declaración de los metadatos de tipo dos para la información referente a la fundación de cada una de las once universidades. La propiedad `name` indica el `id` del nodo de contenido anteriormente declarado y la propiedad `content` indica la celda de una determinada tabla en la que está almacenada la información a presentarse.

```
<meta name="medFundacionU1" content="EPN/1/1"/>
<meta name="medFundacionU2" content="ESPOL/1/1"/>
<meta name="medFundacionU3" content="ESPE/1/1"/>
<meta name="medFundacionU4" content="ESPOCH/1/1"/>
<meta name="medFundacionU5" content="PUCE/1/1"/>
<meta name="medFundacionU6" content="UCE/1/1"/>
<meta name="medFundacionU7" content="UCuenca/1/1"/>
<meta name="medFundacionU8" content="UAzuay/1/1"/>
<meta name="medFundacionU9" content="USFQ/1/1"/>
<meta name="medFundacionU10" content="UTA/1/1"/>
<meta name="medFundacionU11" content="UTPL/1/1"/>
```

Código 4.11 Declaración de metadatos de tipo dos para la fundación de las once universidades

Para el caso de la información que se obtiene a través del canal de retorno, es necesario declarar por cada universidad los metadatos de tipo tres correspondientes a los aspectos de la misión, visión y oferta de pregrado y postgrado. La declaración de estos metadatos es similar a la de los metadatos de tipo dos, pero el valor de la propiedad `content` culmina con la letra r (referente a retorno). El Código 4.12 presenta la declaración de los metadatos de tipo tres para la información referente a la misión de cada una de las once universidades.

```
<meta name="medMisionU1" content="EPN/1/2/r"/>
<meta name="medMisionU2" content="ESPOL/1/2/r"/>
<meta name="medMisionU3" content="ESPE/1/2/r"/>
<meta name="medMisionU4" content="ESPOCH/1/2/r"/>
<meta name="medMisionU5" content="PUCE/1/2/r"/>
<meta name="medMisionU6" content="UCE/1/2/r"/>
<meta name="medMisionU7" content="UCuenca/1/2/r"/>
<meta name="medMisionU8" content="UAzuay/1/2/r"/>
<meta name="medMisionU9" content="USFQ/1/2/r"/>
<meta name="medMisionU10" content="UTA/1/2/r"/>
<meta name="medMisionU11" content="UTPL/1/2/r"/>
```

Código 4.12 Declaración de metadatos de tipo tres para la misión de las once universidades

4.4.7 RESULTADOS OBTENIDOS

Al culminar el trabajo con el IDE NCL Composer y el *plug-in* Menu Creator, se obtiene como resultado la aplicación interactiva a ser presentada al televidente pero aún sin datos, es decir, presentando una estructura estática para posteriormente integrarse con la información que se desee. La Figura 4.27 presenta el diagrama NCM obtenido, correspondiente a la aplicación interactiva.

El *plug-in* Menu Creator genera, por cada menú desarrollado, su correspondiente nodo de contexto para que pueda ser integrado con el resto de elementos media de la aplicación interactiva haciendo uso del resto de *plug-ins* del IDE NCL Composer. Para ello, únicamente es necesario crear una puerta que permita alcanzar al nodo de contexto cuya presentación se desee iniciar, detener, pausar, reiniciar, entre otras acciones.

En la Figura 4.28 se puede visualizar el primer menú de la aplicación interactiva, aún sin datos, al ser ejecutada en el STB.

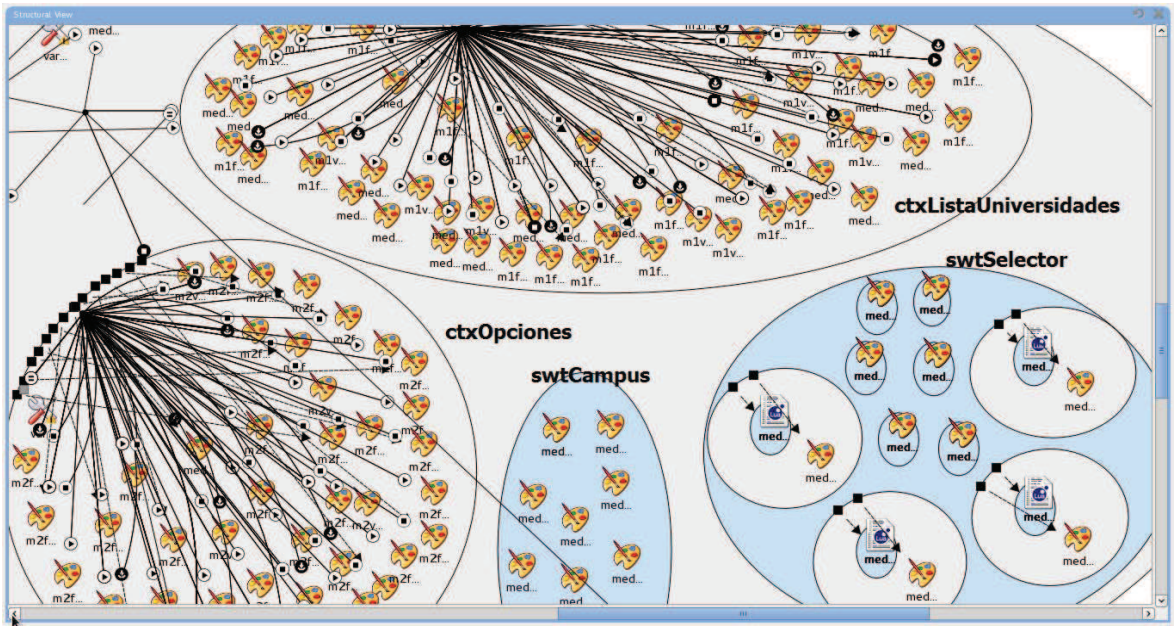


Figura 4.27 Diagrama NCM de la primera aplicación interactiva

Figura 4.28 Primer menú aún sin datos de la primera aplicación interactiva

La Figura 4.29 presenta el segundo menú, aún sin datos, el cual se presenta al seleccionar un determinado elemento del primer menú.

Figura 4.29 Menús aún sin datos de la primera aplicación interactiva

4.5 GENERACIÓN DE ELEMENTOS MEDIA EMPLEANDO LA APLICACIÓN MIXER (NCL-TEXTUAL DATA MIXER)

La aplicación Mixer permite generar los elementos media, en base a los metadatos declarados, correspondientes a las imágenes PNG con transparencia que contienen el texto y *scripts* Lua necesarios para completar la aplicación interactiva desarrollada.

4.5.1 CREACIÓN DE UNA INSTANCIA E INGRESO DE DATOS PARA EL PROCESO DE ASOCIACIÓN

Para realizar el proceso de asociación en base a los metadatos declarados dentro del documento NCL de la aplicación interactiva, es necesario crear una única instancia[75]. La Figura 4.30 muestra los tres datos para la creación de la instancia que se han ingresado. El documento NCL corresponde al archivo de extensión .ncl obtenido al desarrollar la aplicación interactiva sin datos, y la base de datos corresponde a la base de datos anteriormente creada con la aplicación cliente Textual Data Creator.

[75] Una instancia es el proceso de asociación de metadatos que se lleva a cabo para generar los elementos media de una aplicación interactiva. La aplicación Mixer es una aplicación de varias instancias, pues permite llevar a cabo el proceso de asociación de metadatos de varias aplicaciones interactivas que se estén desarrollando.

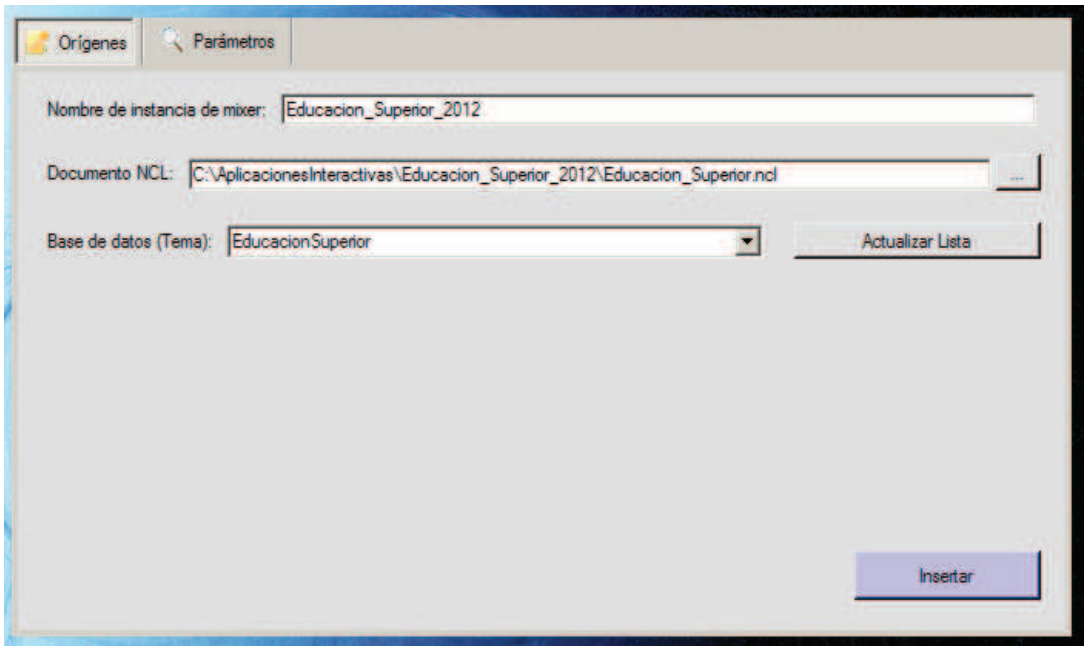


Figura 4.30 Creación de una instancia para el proceso de asociación

Al hacer clic en el botón "Insertar", la aplicación Mixer reconoce y valida automáticamente los metadatos declarados dentro del documento NCL y presenta un listado en la pestaña "Parámetros", como se puede observar en la Figura 4.31.

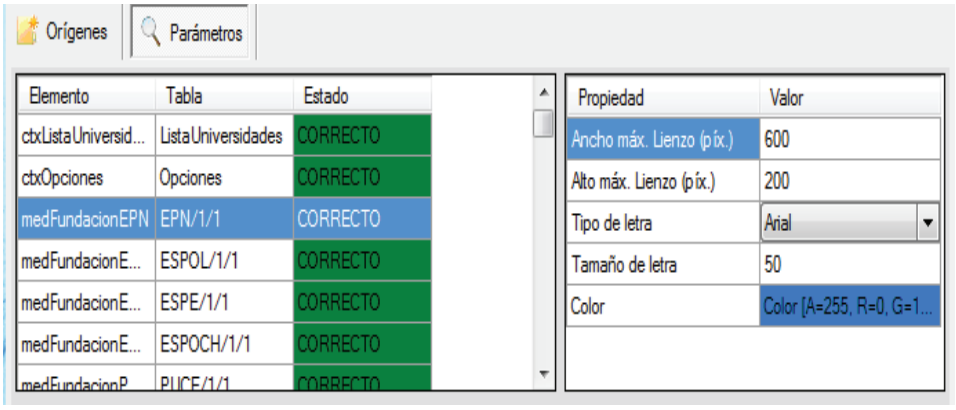


Figura 4.31 Metadatos encontrados en el documento NCL de la aplicación interactiva

4.5.2 CONFIGURACIÓN DE LAS PROPIEDADES DEL TEXTO DE LOS METADATOS

Por defecto, la aplicación Mixer genera el texto que presentarán los elementos media que describen cada tipo de metadato con determinadas propiedades. Para el desarrollo de la aplicación interactiva se han modificado estas propiedades con el fin de mejorar la presentación del texto que visualizará el televidente.

Una vez configuradas las propiedades que se desea para el texto en cada tipo de metadato, la aplicación Mixer genera de forma automática los elementos media (imágenes PNG con transparencia o *scripts* Lua) con las propiedades del texto establecidas, obteniéndose de esta manera la aplicación interactiva final a ser presentada al televidente.

4.5.2.1 Metadatos de tipo uno

Para el texto de cada uno de los elementos, tanto del primer menú como del segundo menú desarrollado, se han establecido las siguientes propiedades:

- **Ancho máximo del lienzo (píxeles):** 600.
- **Alto máximo del lienzo (píxeles):** 200.
- **Tipo de letra:** Arial.
- **Tamaño de letra:** 50.
- **Color:** Azul (A=255, R=0, G=128, B=255).

La Figura 4.32 presenta la configuración de las propiedades del texto realizadas para los metadatos de tipo uno correspondientes a los menús desarrollados.

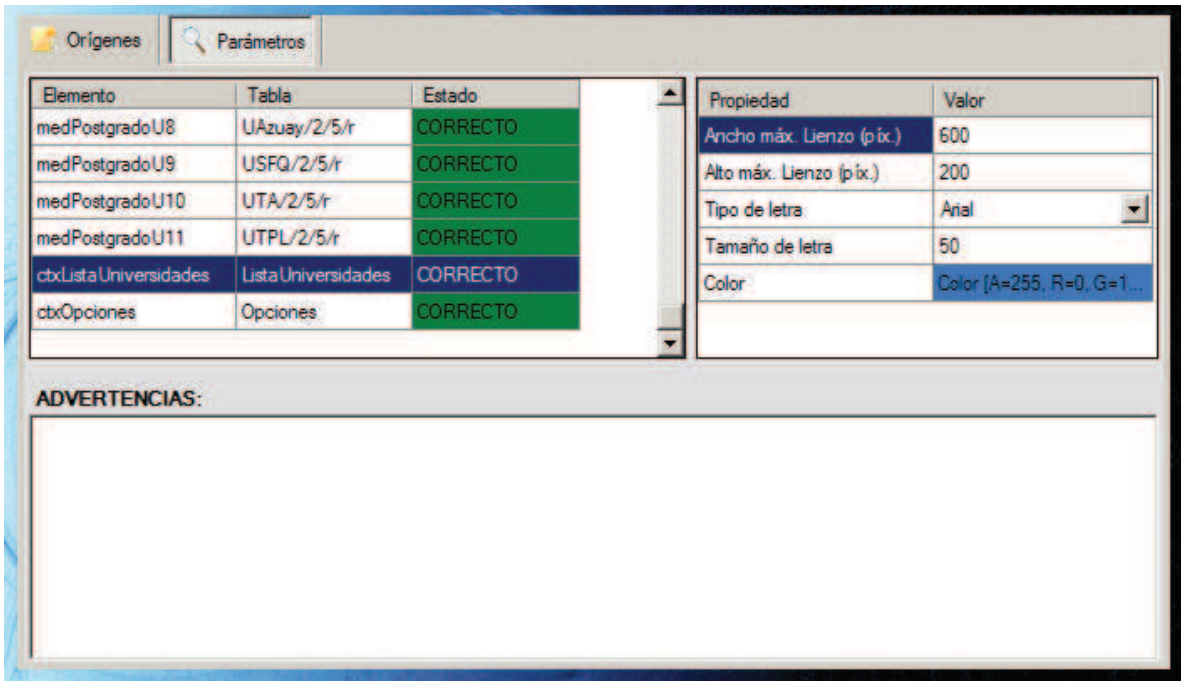


Figura 4.32 Configuración de las propiedades del texto de los metadatos de tipo uno

4.5.2.2 Metadatos de tipo dos

Los metadatos de tipo dos presentan la información correspondiente a la fundación, ubicación, autoridades, lema, sitio web e información de contacto de cada universidad. Para presentar el texto de estos aspectos, se han establecido las siguientes propiedades:

- **Ancho máximo del lienzo (píxeles):** Variable, depende de la cantidad de texto a presentarse para cada aspecto.
- **Alto máximo del lienzo (píxeles):** Variable, depende de la cantidad de texto a presentarse para cada aspecto.
- **Tipo de letra:** Tahoma.
- **Tamaño de letra:** Variable, depende de la cantidad de texto a presentarse en el tamaño de lienzo de cada aspecto.
- **Color:** Negro.

La Figura 4.33 presenta, como ejemplo, la configuración de las propiedades del texto correspondientes al metadato de tipo dos de la información de las autoridades de la Universidad Técnica Particular de Loja (UTPL).

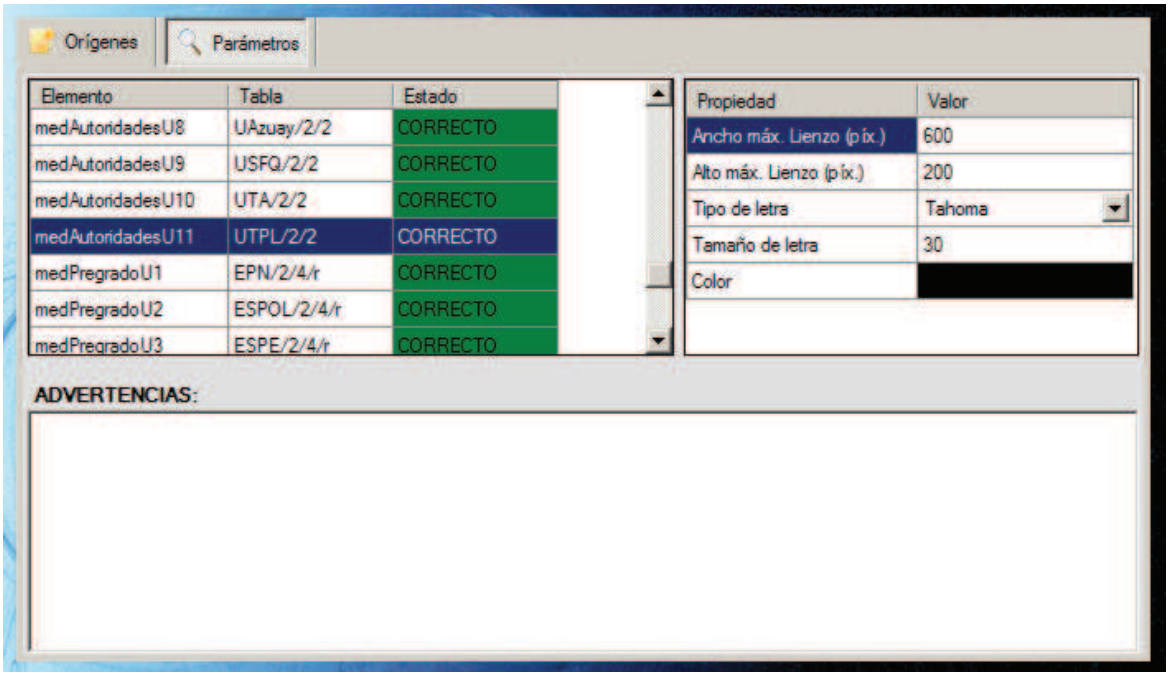


Figura 4.33 Ejemplo de configuración de las propiedades del texto de un metadato de tipo dos

4.5.2.3 Metadatos de tipo tres

Los metadatos de tipo tres presentan la información correspondiente a la misión, visión y oferta de pregrado y postgrado de cada universidad. Como la información para estos aspectos se obtiene a través del canal de retorno, únicamente se puede configurar el tamaño de letra y el color.

Se han establecido las siguientes propiedades para presentar el texto de estos aspectos:

- **Tamaño de letra:** Variable, depende de la cantidad de texto a presentarse en la región NCL de cada aspecto.
- **Color:** Negro.

La Figura 4.34 presenta como ejemplo la configuración de las propiedades del texto correspondientes al metadato de tipo tres de la oferta de pregrado de la Escuela Politécnica Nacional (EPN).

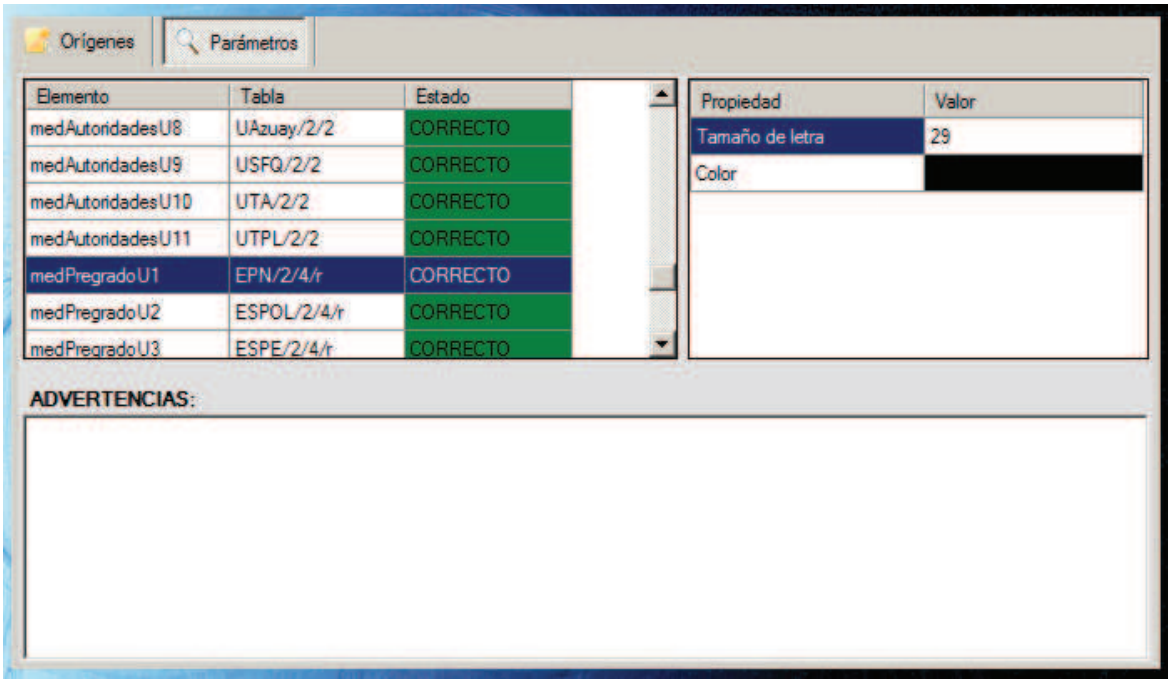


Figura 4.34 Ejemplo de configuración de las propiedades del texto de un metadato de tipo tres

4.5.3 RESULTADOS OBTENIDOS

Cuando la aplicación Mixer culmina de realizar el proceso de asociación y genera los elementos media dentro del documento NCL, se obtiene la aplicación interactiva con datos, es decir, la aplicación interactiva final a presentarse al televidente.

La Figura 4.35 muestra la primera aplicación interactiva con el primer menú desarrollado presentando los acrónimos de cada universidad con las propiedades del texto que se establecieron para el metadato de tipo uno correspondiente a este menú.

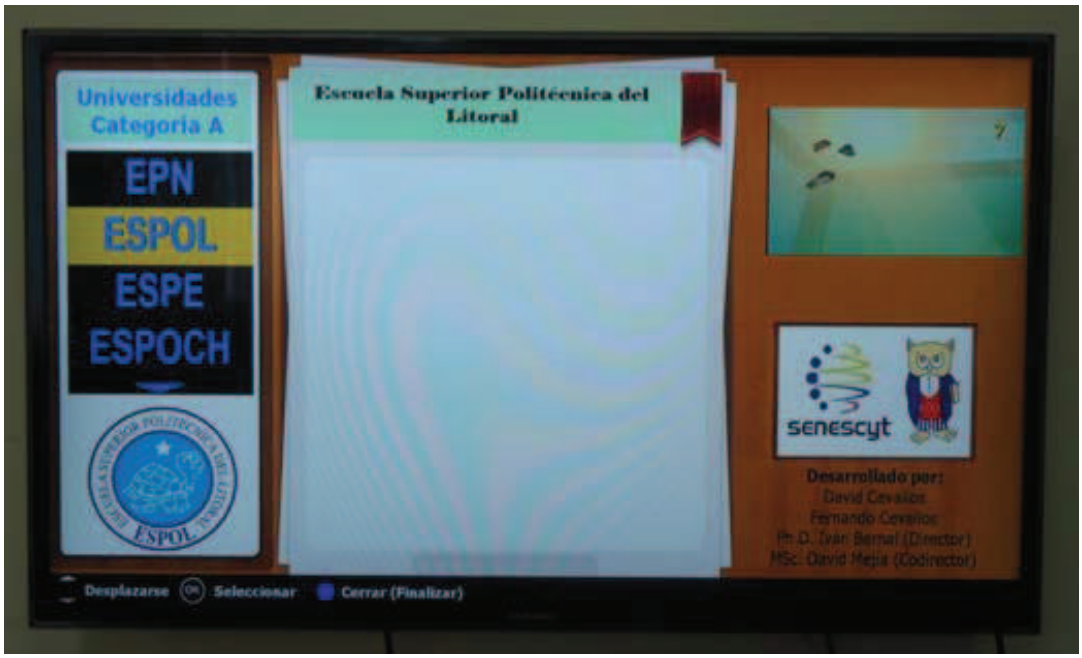


Figura 4.35 Primer menú de la primera aplicación interactiva

La Figura 4.36 muestra la aplicación interactiva con el segundo menú desarrollado. Este menú presenta, con las propiedades del texto que se establecieron para el metadato de tipo uno correspondiente a este menú, los aspectos que puede visualizar el televidente para la universidad seleccionada.

En la sección 4.7.1 se detallan las pruebas de funcionamiento y los resultados obtenidos de esta primera aplicación interactiva.

Figura 4.36 Menús de la primera aplicación interactiva

4.6 EJEMPLO DE FLEXIBILIDAD QUE OFRECE EL SISTEMA: DESARROLLO DE LA SEGUNDA APLICACIÓN INTERACTIVA A PARTIR DE LA PRIMERA [2]

El sistema de búsqueda, almacenamiento y procesamiento de información brinda gran flexibilidad en la generación de contenido interactivo de televisión digital, permitiendo que al realizar pequeñas modificaciones, una aplicación interactiva anteriormente desarrollada pueda ser empleada para presentar otra información o información actualizada (otros datos).

A manera de ejemplo se ha desarrollado una segunda aplicación interactiva, a partir de la primera aplicación desarrollada, para presentar información general de las cinco universidades que en la actualidad son parte de la categoría A. Estas universidades son [2]:

- Escuela Politécnica Nacional (EPN)
- Escuela Superior Politécnica del Litoral (ESPOL)
- Universidad San Francisco de Quito (USFQ)
- Universidad Andina Simón Bolívar (UASB)
- Facultad Latinoamericana de Ciencias Sociales (FLACSO)

De estas universidades, las tres primeras tienen oferta de pregrado y de postgrado, y ya fueron incluidas en la primera aplicación interactiva, mientras que las dos últimas únicamente tienen oferta de postgrado y serán incluidas en la segunda aplicación.

Esta segunda aplicación interactiva es igual a la primera, pero en lugar del menú de once filas y una columna contará con un menú de cinco filas y una columna que se presenta al televidente en una única vista.

4.6.1 GENERACIÓN DE NUEVAS TABLAS PARA LA BASE DE DATOS

Para el caso de la información ingresada haciendo uso de Textual Data Creator, es necesario añadir dos tablas más a la base de datos, correspondientes a la Universidad Andina Simón Bolívar (UASB) y a la Facultad Latinoamericana de Ciencias Sociales (FLACSO). Además, dado que la lista de universidades cambia, se ha incluido una nueva tabla con la nueva lista de universidades.

La Tabla 4.10 presenta las tablas que se han añadido a la base de datos, así como la información que almacenan y sus dimensiones.

Tabla 4.10 Tablas añadidas a la base de datos

Nombre de la tabla	Información que almacena cada tabla	Dimensiones (filas x columnas)
ListaUniversidades1	Acrónimo de cada universidad	5 x 1
USAB	Información de la Universidad Andina Simón Bolívar	2 x 6
FLACSO	Información de la Facultad Latinoamericana de Ciencias Sociales	2 x 6

El resultado final obtenido es una base de datos que puede ser empleada por ambas aplicaciones interactivas. La Figura 4.37 presenta, como ejemplo, la información que ha sido ingresada para la Facultad Latinoamericana de Ciencias Sociales (FLACSO).

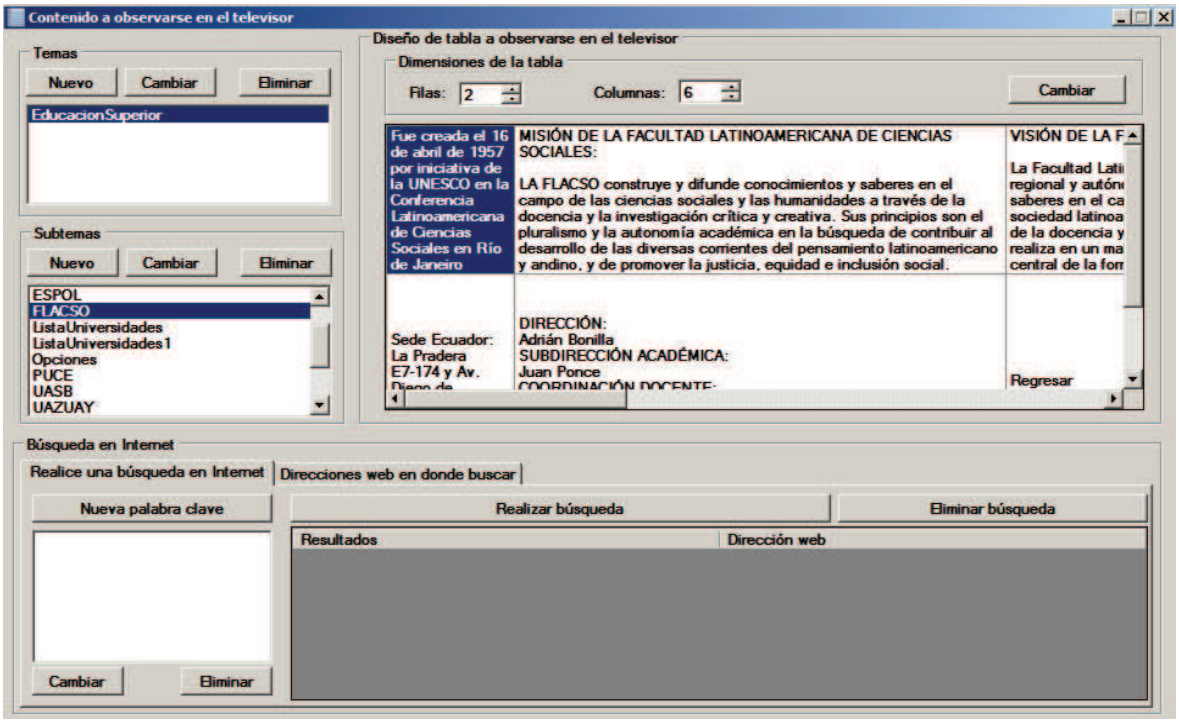


Figura 4.37 Información ingresada para la Facultad Latinoamericana de Ciencias Sociales (FLACSO)

4.6.2 MODIFICACIÓN DE LA APLICACIÓN INTERACTIVA

4.6.2.1 Modificación del primer menú

Haciendo uso del *plug-in* Menu Creator, se puede modificar fácilmente las propiedades del primer menú de la aplicación interactiva. La Figura 4.38 detalla las propiedades del menú de cinco filas y una columna que ahora presentará el listado con los acrónimos de las cinco universidades.

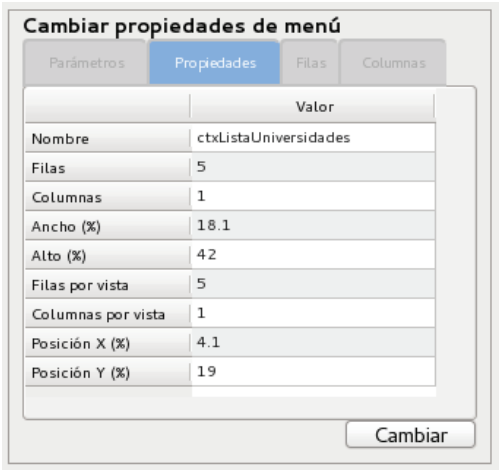


Figura 4.38 Propiedades de diseño para el nuevo menú

Al realizar los cambios, el *plug-in* Menu Creator modifica automáticamente el menú anterior dentro del mismo nodo de contexto. La Figura 4.39 presenta el primer menú, con las nuevas propiedades de diseño, graficado en el navegador interactivo de menús del *plug-in* Menu Creator, junto con el segundo menú.

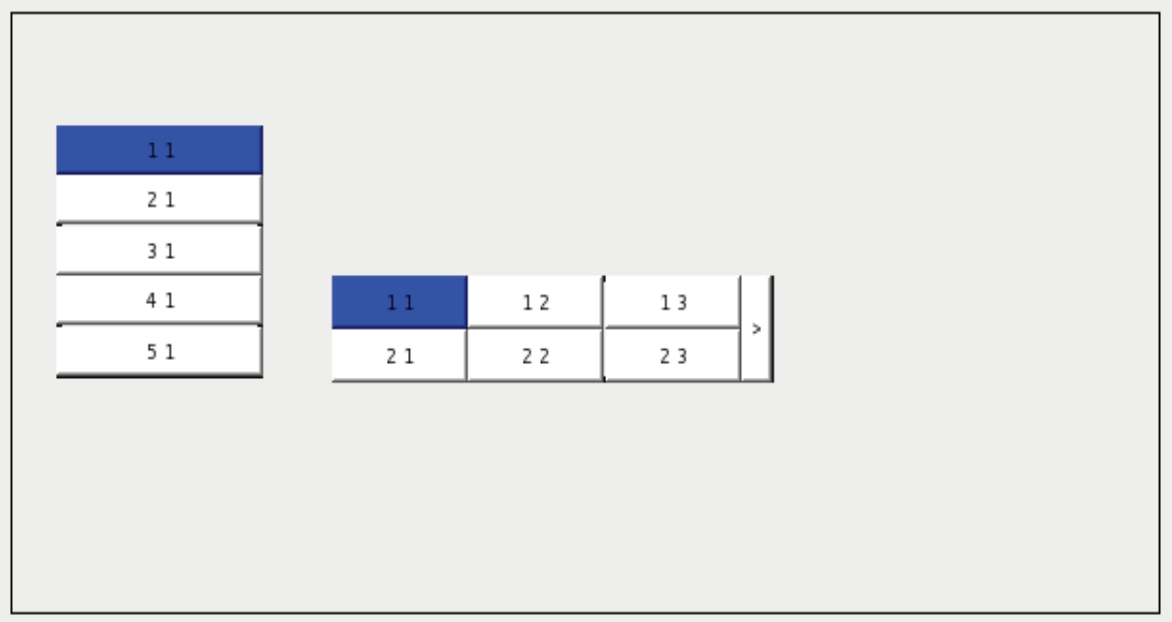


Figura 4.39 Menús presentados en el navegador interactivo de menús

4.6.2.2 Modificación de metadatos

Con el fin de determinar la información que se presentará en el primer menú, es necesario modificar el valor de la propiedad `content` del metadato de tipo uno correspondiente a este menú, ingresando el nombre de la tabla que almacena el listado con los acrónimos de las cinco universidades, como se puede apreciar en el Código 4.13.

```
<meta  name="ctxListaUniversidades" content="ListaUniversidades1"/>
```

Código 4.13 Declaración del metadato de tipo uno correspondiente al primer menú

En los metadatos de tipo dos, de igual manera, es necesario modificar el valor de la propiedad `content` de cada metadato, de manera que ahora especifique la tabla correspondiente a cada universidad. Haciendo uso del *plug-in* NCL Outline View, los cinco primeros metadatos de tipo dos de cada aspecto son modificados,

y los metadatos restantes son eliminados manualmente. El Código 4.14 muestra la declaración de los metadatos de tipo dos correspondientes a la fundación de las cinco universidades.

```
<meta name="medFundacionU1" content="EPN/1/1"/>
<meta name="medFundacionU2" content="ESPOL/1/1"/>
<meta name="medFundacionU3" content="USFQ/1/1"/>
<meta name="medFundacionU4" content="UASB/1/1"/>
<meta name="medFundacionU5" content="FLACSO/1/1"/>
```

Código 4.14 Declaración de metadatos de tipo dos para la fundación de las cinco universidades

De manera similar se procede con los metadatos de tipo tres. Los cinco primeros metadatos de tipo tres correspondientes a los aspectos cuya información es obtenida a través del canal de retorno son modificados, y los metadatos restantes son eliminados manualmente. El Código 4.15 muestra la declaración de los metadatos de tipo tres correspondientes a la misión de las cinco universidades.

```
<meta name="medMisionU1" content="EPN/1/2/r"/>
<meta name="medMisionU2" content="ESPOL/1/2/r"/>
<meta name="medMisionU3" content="USFQ/1/2/r"/>
<meta name="medMisionU4" content="UASB/1/2/r"/>
<meta name="medMisionU5" content="FLACSO/1/2/r"/>
```

Código 4.15 Declaración de metadatos de tipo tres para la misión de las cinco universidades

4.6.2.3 Nombres de elementos media

Los nombres de los elementos media corrrespondientes al escudo, el campus y el título con el nombre de cada universidad a presentarse al televidente, deben ser asociados con las universidades correspondientes como sugiere la Tabla 4.11.

Tabla 4.11 Identificadores empleados para cada universidad de la segunda aplicación interactiva

Identificador	Universidad
U1	Escuela Politécnica Nacional (EPN)
U2	Escuela Politécnica de Litoral (ESPOL)
U3	Universidad San Francisco de Quito (USFQ)
U4	Universidad Andina Simón Bolívar (UASB)
U5	Facultad Latinoamericana de Ciencias Sociales (FLACSO)

4.6.2.4 Eliminación de enlaces NCL y de opciones de los nodos switch

Como el primer menú anteriormente estaba conformado por once elementos y ahora por cinco, los enlaces NCL insertados para el sexto elemento del menú hasta el décimo primer elemento deben ser eliminados manualmente. Este procedimiento es realizado en el IDE NCL Composer mediante el *plug-in* NCL Outline View. De esta manera, se conservan únicamente los enlaces NCL que han sido insertados para los cinco primeros elementos del menú.

De igual manera, en los nodos *switch* que toman una decisión en función de la universidad seleccionada, se conservan únicamente las cinco primeras opciones y se eliminan manualmente las opciones restantes.

4.6.2.5 Resultados obtenidos

Al terminar de realizar los cambios con el IDE NCL Composer en la aplicación interactiva, se obtiene una segunda aplicación similar a la primera pero con el diseño del primer menú modificado para presentar información de las cinco universidades. La aplicación aún no tiene datos. La Figura 4.40 muestra el nuevo diagrama NCM obtenido.

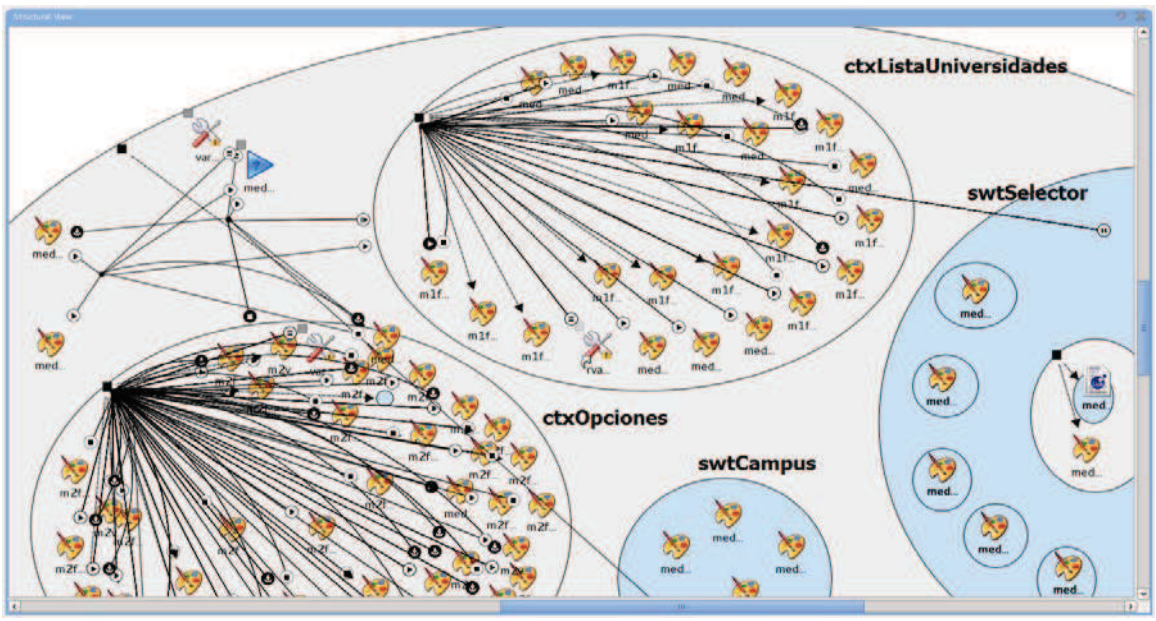


Figura 4.40 Diagrama NCM de la segunda aplicación interactiva

La Figura 4.41 muestra el primer menú de la aplicación interactiva con las propiedades establecidas, aún sin datos, listo para posteriormente poder visualizar los acrónimos de las cinco universidades.

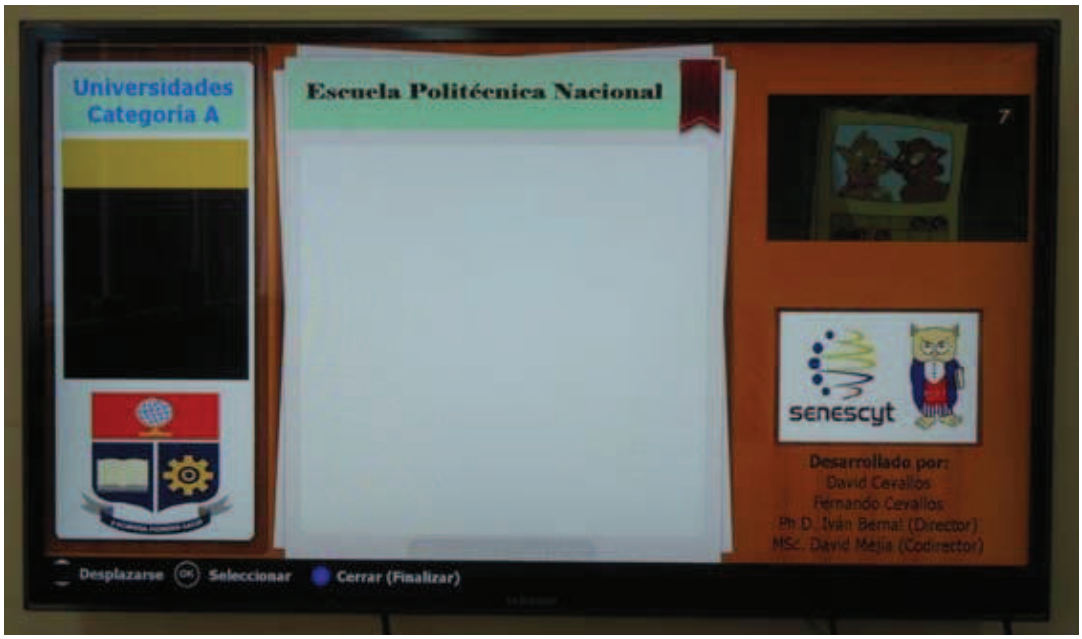


Figura 4.41 Primer menú aún sin datos de la segunda aplicación interactiva

La Figura 4.42 muestra los dos menús de la aplicación interactiva, aún sin datos, que posteriormente presentarán información al televidente.

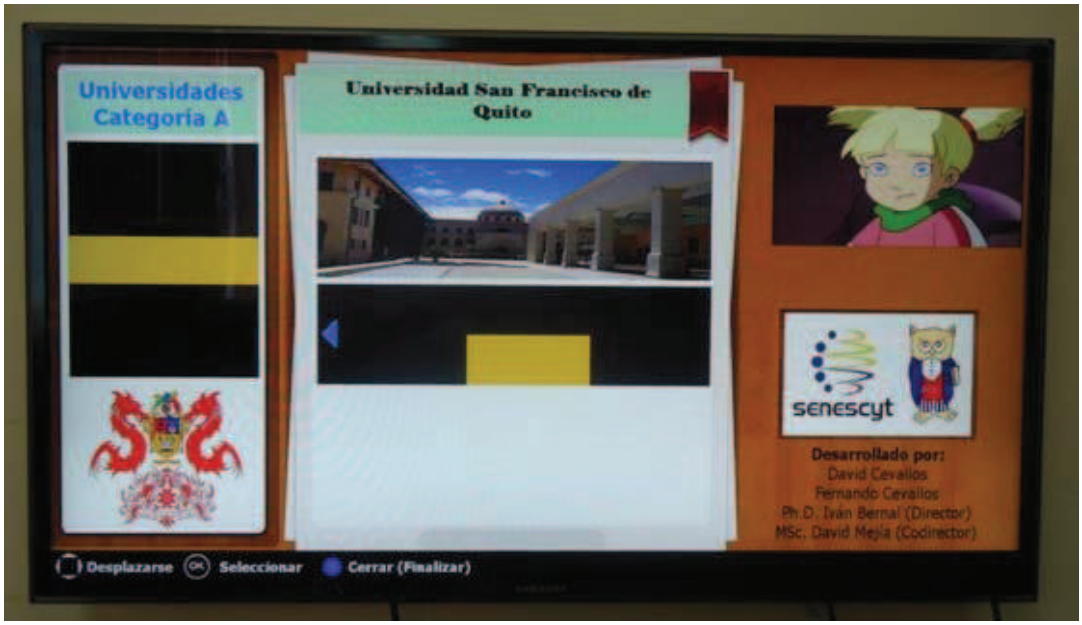


Figura 4.42 Menús aún sin datos de la segunda aplicación interactiva

4.6.3 GENERACIÓN DE ELEMENTOS MEDIA

Para la generación de los elementos media de la nueva aplicación interactiva, haciendo uso de la aplicación Mixer se puede crear una nueva instancia o bien modificar el parámetro correspondiente al documento NCL de la anterior instancia creada para la primera aplicación interactiva. Luego, se realiza el mismo procedimiento que se detalló en la sección 4.5, estableciendo los parámetros del texto para cada tipo de metadato.

4.6.3.1 Resultados obtenidos

El resultado obtenido después de hacer uso de la aplicación Mixer es la aplicación interactiva final a presentarse al televidente, es decir, la aplicación interactiva con datos.

La Figura 4.43 muestra la aplicación interactiva obtenida con el primer menú desarrollado.

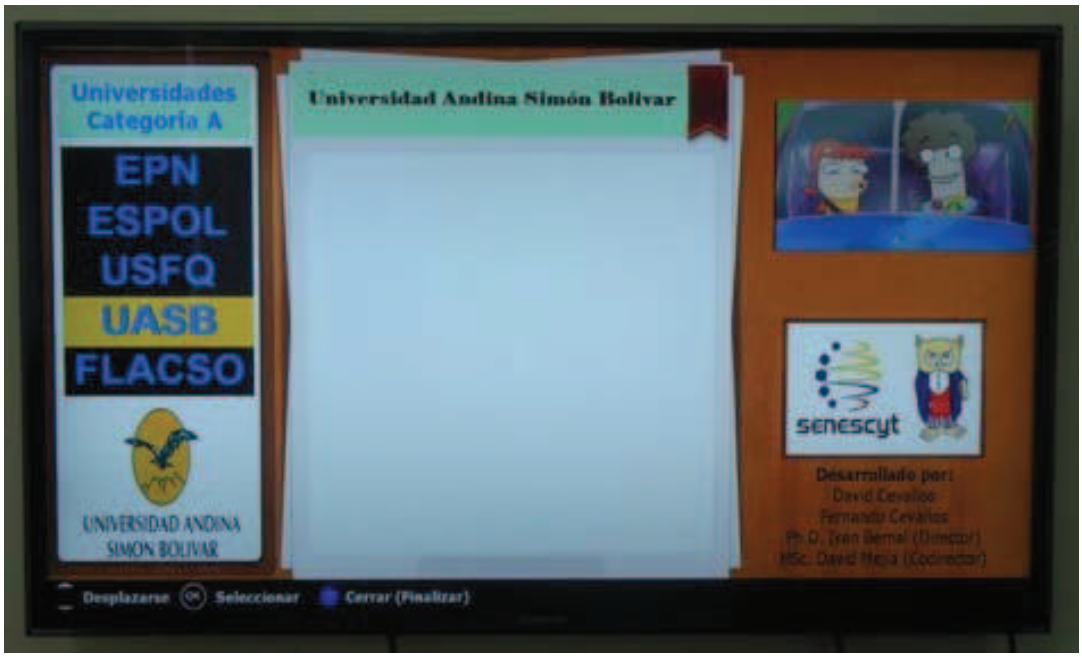


Figura 4.43 Primer menú de la segunda aplicación interactiva

La Figura 4.44 muestra la aplicación interactiva obtenida con los menús desarrollados.

Figura 4.44 Menús de la segunda aplicación interactiva

En la sección 4.7.2 se detallan las pruebas de funcionamiento y los resultados obtenidos de esta segunda aplicación interactiva.

4.7 PRUEBAS DE FUNCIONAMIENTO Y RESULTADOS OBTENIDOS

Para la realización de las pruebas de funcionamiento, las aplicaciones interactivas desarrolladas han sido cargadas directamente en el STB híbrido EiTV insertando una memoria en el puerto USB. Sin embargo, las aplicaciones podrían ser cargadas en el STB por medio de otros mecanismos, como por ejemplo, mediante su radiación a través de la interfaz de aire o mediante su transmisión a través del canal de retorno.

4.7.1 PRUEBAS DE FUNCIONAMIENTO Y RESULTADOS OBTENIDOS CON LA PRIMERA APLICACIÓN INTERACTIVA

Una vez que la aplicación interactiva se carga en el STB, se visualiza el ícono de interactividad en la parte superior derecha de la pantalla del televisor sugiriendo al televidente pulsar el botón OK del control remoto para poder ingresar a la aplicación, como se muestra en la Figura 4.45.

Figura 4.45 Ícono de interatividad

En la Figura 4.46 se muestra la pantalla inicial de la aplicación interactiva que se presenta al televidente luego de pulsar el botón OK, en la que se puede apreciar una imagen de fondo, el video de la programación redimensionado en un costado y el listado de universidades que conformaron la categoría A durante el período 2009-2013, en el primer menú desarrollado.

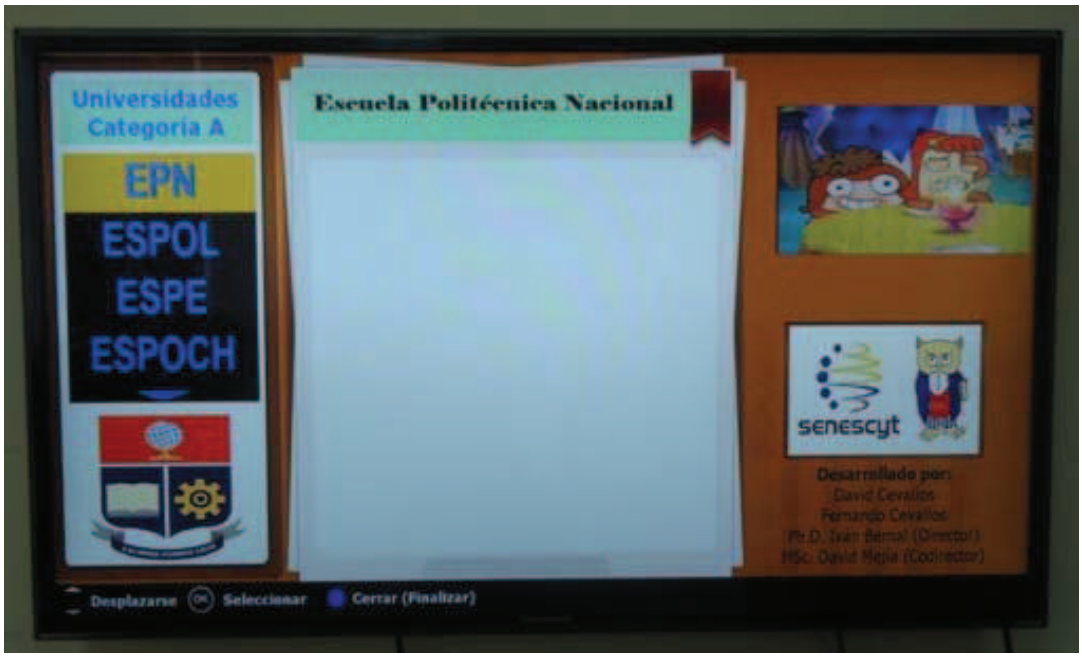


Figura 4.46 Primera vista del primer menú

El televidente puede desplazarse entre los elementos del menú con los botones ARRIBA y ABAJO del control remoto. El foco se mueve de acuerdo a las indicaciones del televidente señalando la universidad que puede elegir, mostrando su respectivo escudo en la parte inferior y el título con el nombre de la universidad en la parte superior. Al desplazarse el foco, la imagen del escudo y el título con el nombre cambian automáticamente, para mostrar el nuevo escudo y título de la universidad correspondiente al elemento del menú donde el foco se ha desplazado. Mientras se navega entre los elementos del menú no se aprecia *flickering*[76].

Las universidades presentadas en el primer menú se muestran en vistas de cuatro elementos. Al pulsar el botón OK sobre las viñetas de navegación del menú, se presenta la vista de cuatro elementos anterior o la vista de cuatro elementos siguiente según corresponda. La Figura 4.47 presenta la segunda vista del menú, mientras que la Figura 4.48 presenta la tercera vista. La aplicación presenta la vista correcta acorde a la viñeta de navegación seleccionada sin ningún problema. De igual manera, no se aprecia *flickering* al navegar entre los elementos de la segunda y tercera vista del primer menú.

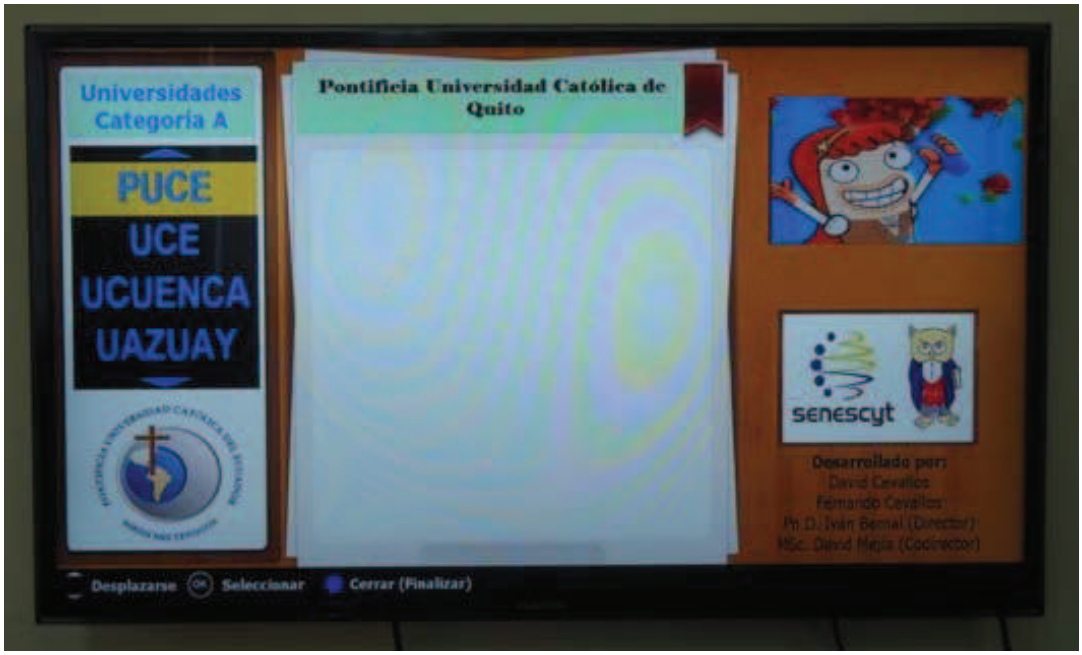


Figura 4.47 Segunda vista del primer menú

[76] *Flickering*: Parpadeo o titilación intermitente que se presenta durante unos segundos.

Figura 4.48 Tercera vista del primer menú

Cuando se selecciona una universidad pulsando el botón OK, se presenta el segundo menú desarrollado en el centro del televisor, como se muestra en la Figura 4.49, ofreciendo al televidente el conjunto de aspectos que puede visualizar sobre la universidad elegida. Además, aparece sobre el menú la imagen del campus universitario de la universidad seleccionada.

El segundo menú ofrece al televidente los diez aspectos que se determinaron en el diseño de la aplicación: fundación, ubicación, autoridades, lema, sitio web, información de contacto, misión, visión y oferta de pregrado y postgrado. La información de los seis primeros será presentada directamente mediante imágenes PNG y la de los cuatro últimos será obtenida mediante el canal de retorno.

El televidente se puede desplazar entre los elementos del segundo menú haciendo uso de los botones ARRIBA, ABAJO, IZQUIERDA y DERECHA del control remoto. El foco se desplaza sin problemas hacia el nuevo elemento del menú indicado por el televidente.

Figura 4.49 Primera vista del segundo menú

El segundo menú posee dos vistas. Al desplazarse hacia la segunda vista, seleccionando la viñeta de navegación correspondiente, se presenta sin ningún problema la segunda vista del menú, la cual se puede apreciar en la Figura 4.50. Al navegar entre los elementos de la primera vista y de la segunda vista del menú no se aprecia *flickering*.

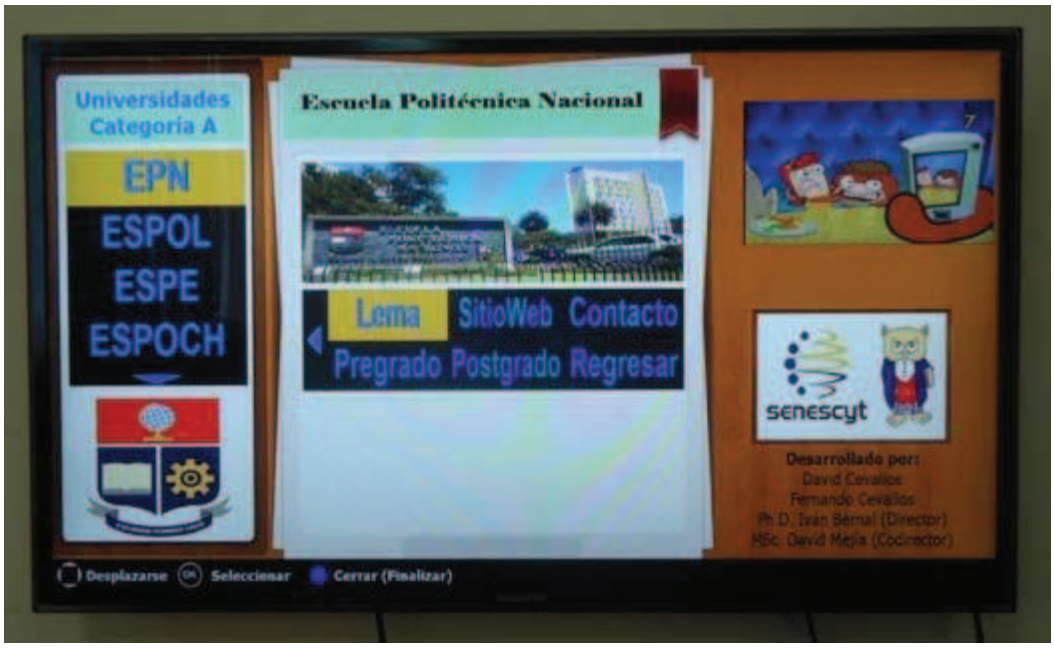


Figura 4.50 Segunda vista del segundo menú

Los datos correspondientes a la fundación, ubicación, autoridades, lema, sitio web e información de contacto de la universidad se presentan al televidente en la parte inferior del segundo menú, tan pronto son seleccionados con el botón OK del control remoto. La Figura 4.51 y la Figura 4.52 muestran ejemplos de los resultados obtenidos.

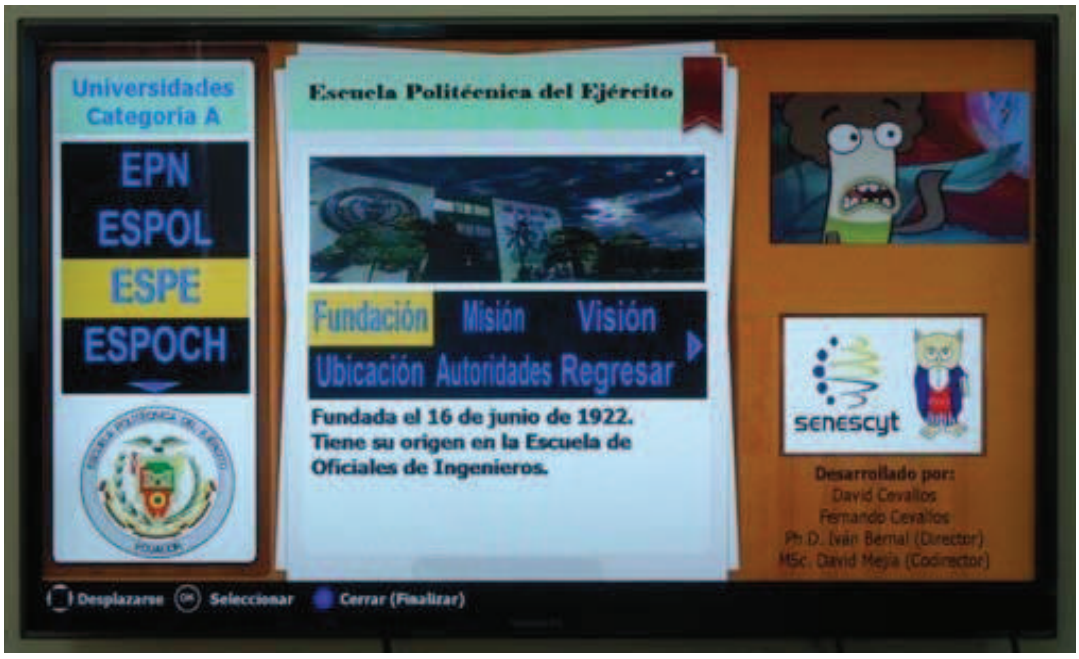


Figura 4.51 Presentación de información en la primera aplicación interactiva (Ejemplo 1)

Figura 4.52 Presentación de información en la primera aplicación interactiva (Ejemplo 2)

En cada una de las dos vistas del segundo menú se presenta la opción “Regresar”. Al seleccionar uno de estos elementos con el botón OK del control remoto, la aplicación detiene la presentación del segundo menú, así como la imagen del campus universitario e información que se esté visualizando en ese momento, para regresar nuevamente al primer menú y poder elegir una nueva universidad.

En el caso de los datos de la misión, visión y oferta de pregrado y postgrado, al presionar el botón OK se genera una petición HTTP al servidor web, que se envía a través del canal de retorno, para obtener la información del aspecto seleccionado. Cuando se obtiene la información se la presenta en una nueva pantalla con una segunda imagen de fondo y el texto debidamente justificado abarcando gran parte de la pantalla del televisor.

El tiempo que tarda la aplicación en presentar la información en la pantalla depende del aspecto seleccionado, así como también de la conexión de red disponible, pero de manera general dura apenas segundos.

La Figura 4.53 y la Figura 4.54 muestran ejemplos de los resultados al obtener información a través del canal de retorno.

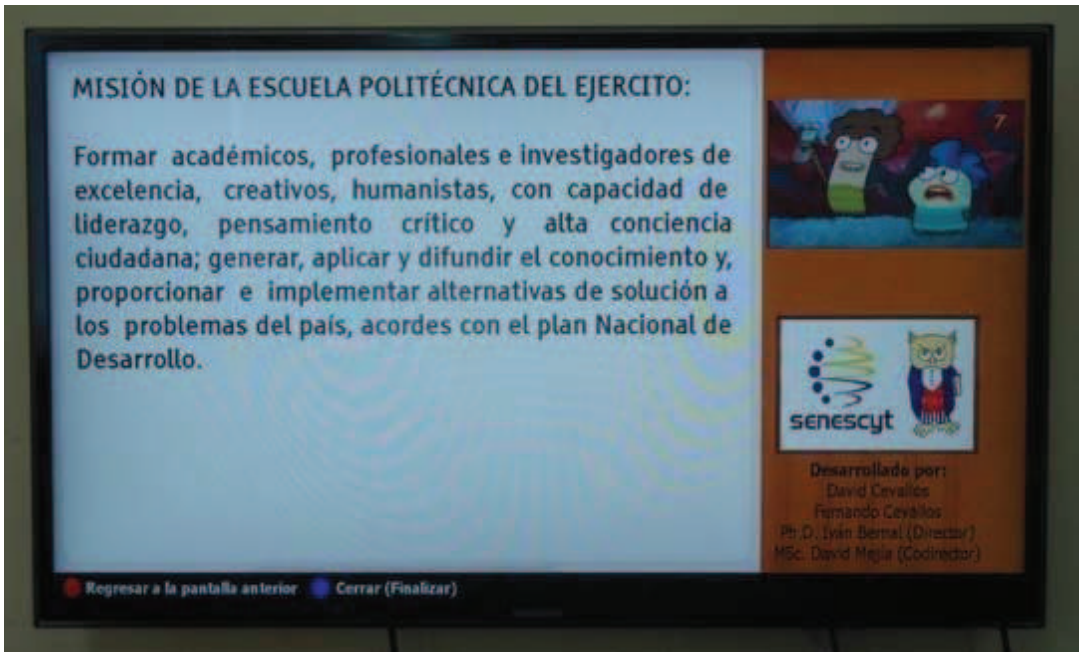


Figura 4.53 Presentación de información obtenida a través del canal de retorno en la primera aplicación interactiva (Ejemplo 1)

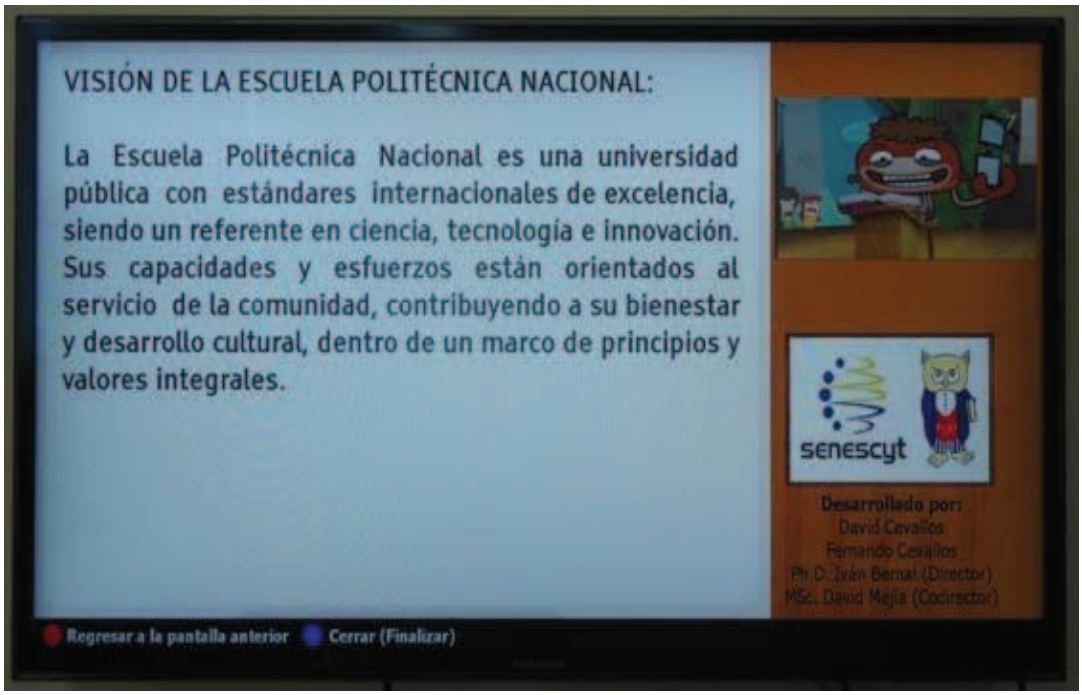


Figura 4.54 Presentación de información obtenida a través del canal de retorno en la primera aplicación interactiva (Ejemplo 2)

Cuando se presiona el botón ROJO se detiene la presentación de la información obtenida a través del canal de retorno, para regresar nuevamente a la presentación original del segundo menú, de manera que el televidente pueda seguir seleccionando otros aspectos sobre la universidad.

No se aprecia *flickering* al alternar de la pantalla principal a la segunda pantalla para presentar la información obtenida a través del canal de retorno. De igual manera, no se aprecia *flickering* al regresar a la pantalla principal cuando el botón ROJO es presionado.

Para las pruebas de funcionamiento, intencionalmente se ha retirado la conectividad entre el STB y el servidor web. Al solicitar un aspecto cuya información es obtenida por el canal de retorno, la aplicación interactiva no finaliza y se puede apreciar el mensaje de no disponibilidad, como muestra la Figura 4.55, lo cual demuestra que el manejo de excepciones del *script* Lua funciona correctamente.

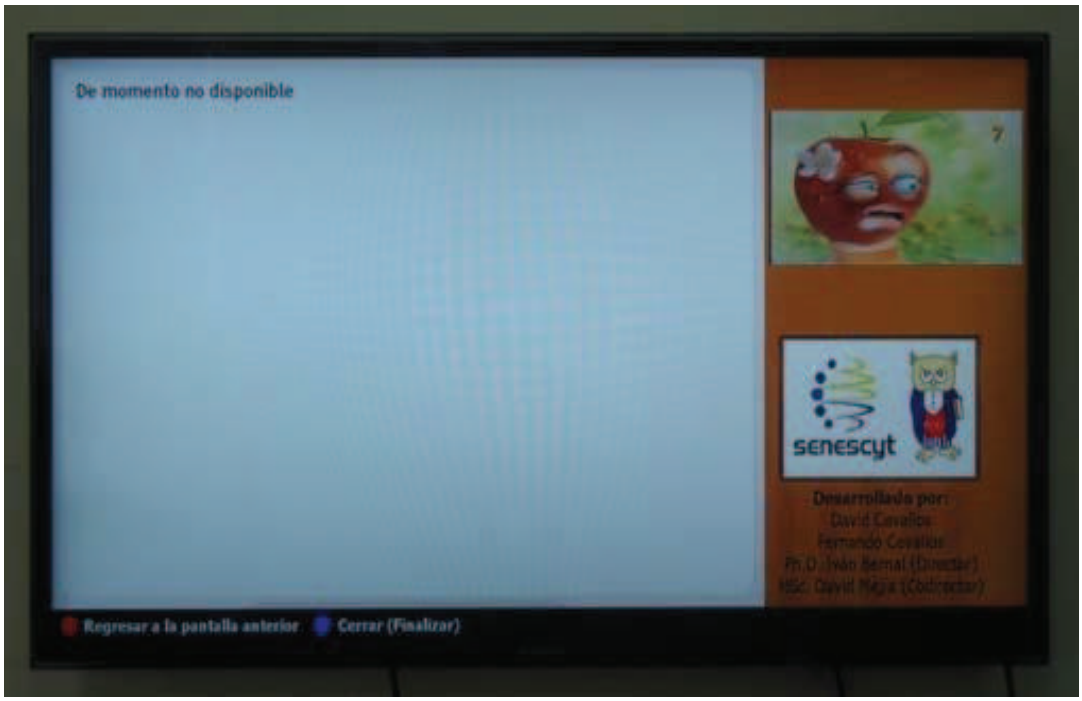


Figura 4.55 Mensaje de no disponibilidad en la primera aplicación interactiva

Si en cualquier momento de la presentación de la aplicación interactiva se presiona el botón AZUL, finaliza inmediatamente la presentación de la misma y el video de la programación se redimensiona a su tamaño original. El ícono de interactividad se vuelve a presentar automáticamente, para que el televidente pueda nuevamente hacer uso de la aplicación, si así lo desea, pulsando el botón OK del control remoto.

4.7.2 PRUEBAS DE FUNCIONAMIENTO Y RESULTADOS OBTENIDOS CON LA SEGUNDA APLICACIÓN INTERACTIVA

Las pruebas de funcionamiento realizadas para la segunda aplicación interactiva fueron las mismas que las de la primera aplicación. De igual manera, los resultados obtenidos fueron similares.

Una vez que la aplicación interactiva se carga en el STB, se visualiza el mismo ícono de interactividad que el de la primera aplicación sugiriendo al televidente pulsar el botón OK del control remoto.

Al presionar el botón OK se presenta la aplicación, de igual forma que en el caso de la primera aplicación interactiva, pero el primer menú muestra tan solo cinco

universidades en una sola vista, como se puede apreciar en la Figura 4.56. Al navegar entre los elementos del primer menú no se aprecia *flickering*.

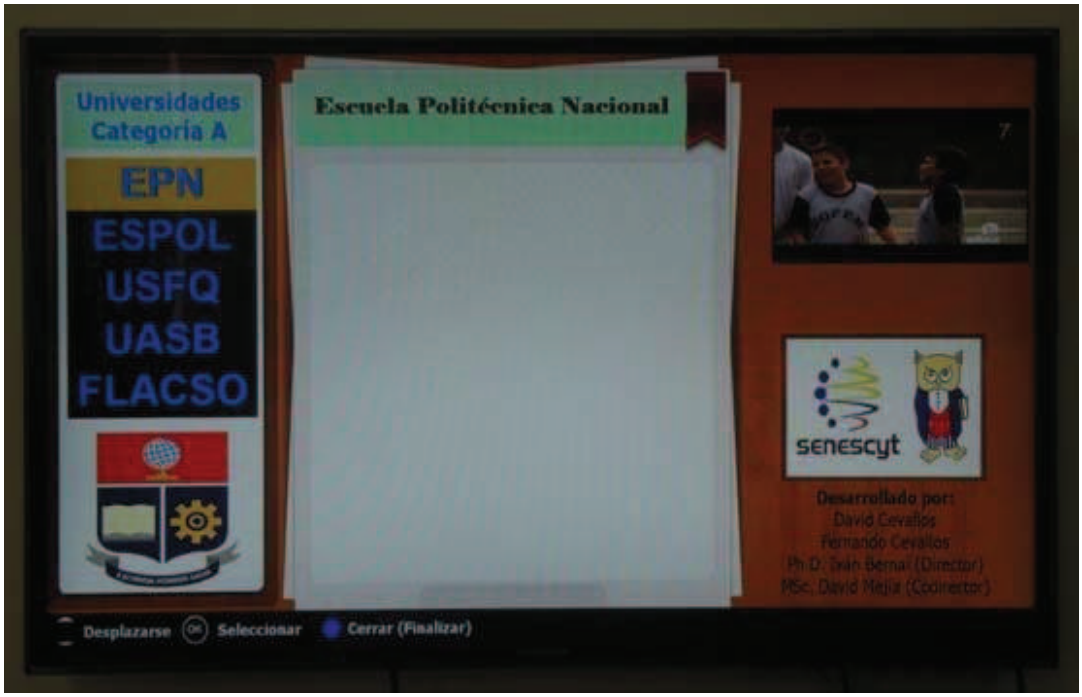


Figura 4.56 Primer menú de la segunda aplicación interactiva

Al desplazarse entre cada uno de los elementos del primer menú, en la parte inferior, se presenta el escudo correspondiente a la universidad que tiene el foco. Además, en la parte superior se presenta el título con el nombre de la universidad.

Al seleccionar una determinada universidad, se presenta el segundo menú y la imagen del campus universitario de la universidad elegida. El segundo menú es el mismo que se utilizó en la primera aplicación interactiva y se pueden visualizar cada una de las dos vistas mediante sus viñetas de navegación.

La Figura 4.57 presenta la primera vista del segundo menú mientras que la Figura 4.58 presenta la segunda vista. Al igual que en la primera aplicación interactiva, no se aprecia *flickering* al navegar entre los elementos de ambas vistas.

Figura 4.57 Primera vista del segundo menú en la segunda aplicación interactiva

Figura 4.58 Segunda vista del segundo menú en la segunda aplicación interactiva

Los aspectos correspondientes a la fundación, ubicación, autoridades, lema, sitio web e información de contacto de la universidad se presentan sin problemas debajo del segundo menú al ser seleccionados con el botón OK del control remoto. La Figura 4.59 y la Figura 4.60 muestran ejemplos de los resultados obtenidos.

Figura 4.59 Presentación de información en la segunda aplicación interactiva (Ejemplo 1)

Figura 4.60 Presentación de información en la segunda aplicación interactiva (Ejemplo 2)

Los aspectos correspondientes a la misión, visión y oferta de pregrado y postgrado de las universidades son obtenidos a través del canal de retorno y presentados, al igual que en la primera aplicación, en una nueva pantalla con una segunda imagen de fondo y el texto debidamente justificado abarcando gran parte

de la pantalla del televisor. La Figura 4.61 y la Figura 4.62 muestran ejemplos de los resultados obtenidos.

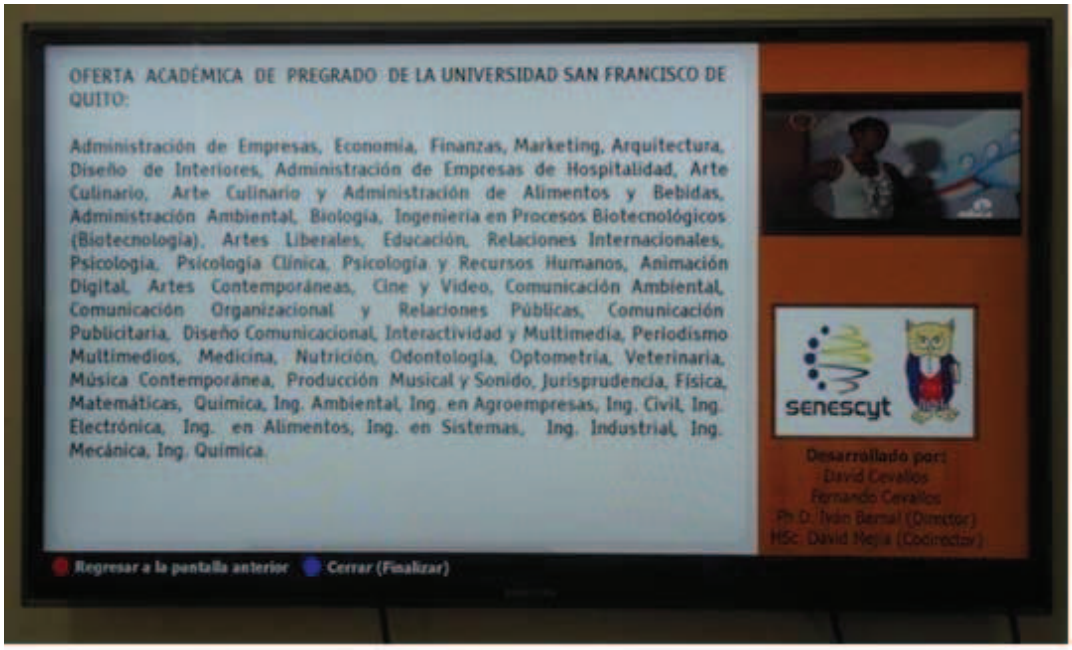


Figura 4.61 Presentación de información obtenida a través del canal de retorno en la segunda aplicación interactiva (Ejemplo 1)

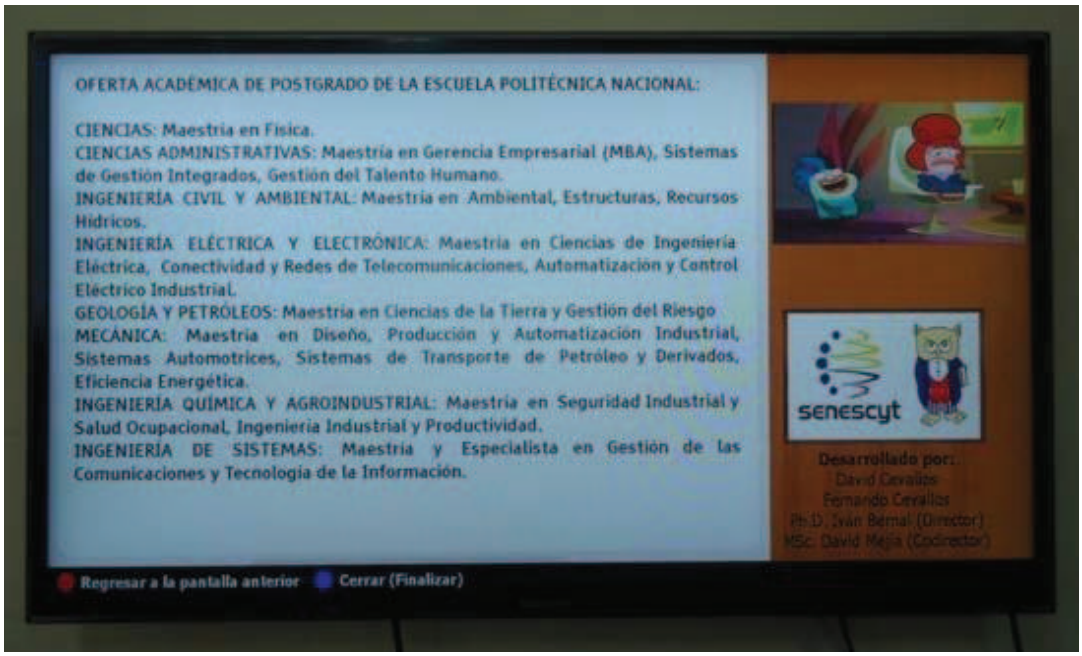


Figura 4.62 Presentación de información obtenida a través del canal de retorno en la segunda aplicación interactiva (Ejemplo 2)

De igual manera, el manejo de excepciones del *script* Lua funciona correctamente, como se puede apreciar en la Figura 4.63.

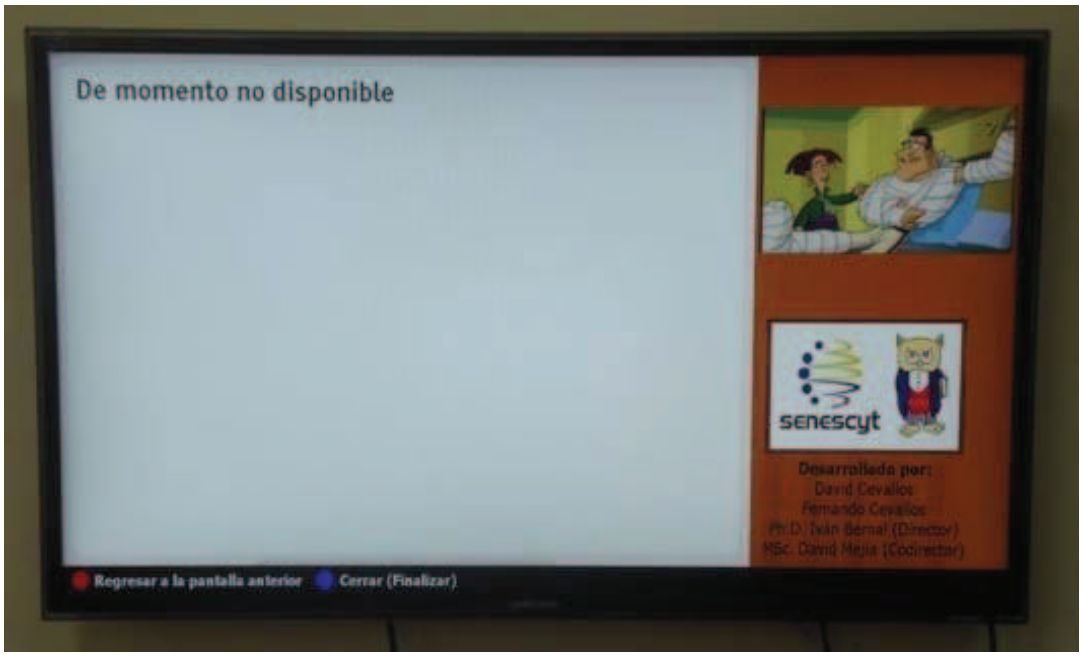


Figura 4.63 Mensaje de no disponibilidad en la segunda aplicación interactiva

Finalmente, las acciones de los botones ROJO y AZUL del control remoto funcionan correctamente, permitiendo detener la presentación de la información obtenida a través del canal de retorno y salir de la aplicación interactiva, respectivamente.

En el Anexo E.6 se ha incluido una guía de usuario de las aplicaciones interactivas desarrolladas.

REFERENCIAS

[1] CDMarket. Linksys WRT120N. [Online]. Disponible en: http://www.cdmarket.com.ar/View/3184/linksys-wrt120n-wireless-n-home-router.aspx (Consultado el 5 de febrero de 2014).

[2] Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEAACES). Categorización Universidades. [Online]. Disponible en: http://www.ceaaces.gob.ec/index.php?option=com_content&view=article&id=19&Itemid=22 (Consultado el 3 de febrero de 2014).

[3] Consejo Nacional de Evaluación y Acreditación de la Educación Superior del Ecuador (CONEA). Evaluación del Desempeño Institucional de las Universidades y Escuelas Politécnicas del Ecuador. [Online]. Disponible en: http://www.educacionsuperior.gob.ec/wp-content/uploads/downloads/2012/07/Extracto_informe_CONEA.pdf (Consultado el 3 de febrero de 2014).

[4] EiTV Entretenimiento e Interactividad para la TV digital. EiTV Developer Box. [Online]. Disponible en: http://www.eitv.com.br/devbox_es.php (Consultado el 5 de febrero de 2014).

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES

- Al analizar el código fuente de NCL Composer, se pudo concluir que la principal característica de este IDE es que basa su funcionamiento en un árbol jerárquico de entidades, el cual únicamente puede ser manipulado por el composer-core. Las entidades que conforman este árbol corresponden a regiones, descriptores, nodos de composición, nodos de contenido, enlaces NCL, etc; los cuales estructuran el documento NCL con el contenido interactivo a presentarse al televidente. Por lo tanto, todo *plug-in* que se desarrolle para este IDE necesariamente deberá emplear el mecanismo *signals and slots* para comunicarse con el composer-core y así poder manipular el árbol jerárquico de manera indirecta. El *plug-in* Menu Creator emite señales al composer-core con el fin de crear, modificar y eliminar entidades que permitan crear, modificar y eliminar menús con un alto nivel de interactividad.

- Al ser NCL un lenguaje declarativo, ciertas tareas como la creación de menús, pueden llegar a ser complicadas y a requerir mucho tiempo de desarrollo, por lo que los menús creados con Ginga-NCL, por lo general, son bastante simples y limitados al hacer uso únicamente de los botones de bifurcación del control remoto (botones numéricos o de colores que permiten al televidente seleccionar una opción de entre varias posibles). Aunque la creación de menús es mucho más sencilla con un lenguaje como Lua, no todos los STB tienen soporte para este lenguaje imperativo.

  Los menús que el *plug-in* Menu Creator es capaz de crear son de un alto nivel de interactividad, porque genera los menús con elementos media y

varias características y funcionalidades que dan lugar a aplicaciones llamativas para el televidente, a diferencia de un menú típico creado con Ginga-NCL en el que el televidente, usualmente, hace uso únicamente de los botones ROJO, VERDE, AMARILLO y AZUL del control remoto para seleccionar las opciones que el menú ofrece. De esta manera, un diseñador de aplicaciones es capaz crear aplicaciones interactivas que requieran menús, en menor tiempo de desarrollo y sin necesidad de tener conocimientos avanzados del lenguaje NCL, haciendo uso de la interfaz gráfica amigable que ofrece el *plug-in*.

- Se requiere que las aplicaciones interactivas tengan tamaños de archivo pequeños (para el código y los medios) para que sea factible su transmisión por la interfaz de aire en tiempos cortos y por la limitación de memoria que pueden tener los STB. Los fabricantes de STB recomiendan que el tamaño de una aplicación interactiva no sea mayor a 6 MB. El *plug-in* Menu Creator, mediante el reuso de la misma imagen de fondo y la de foco (que forman parte de una plantilla para la generación de los elementos de un menú), junto con la capacidad del sistema para generar *scripts* Lua para obtener información a través del canal de retorno (evitando la generación de imágenes), logra que el tamaño de la aplicación interactiva se reduzca considerablemente, permitiendo que una gran cantidad de información se pueda presentar al televidente y de forma individualizada.

  Esta característica del sistema permitió que se pueda presentar información de diez aspectos por cada universidad, que de otra manera no hubiese sido posible por la gran cantidad de imágenes que deberían haberse transmitido.

- La principal fortaleza de la tecnología WCF es que permite la creación de servicios web empleando el protocolo SOAP, la arquitectura REST o ambos (servicios web híbridos). SOAP facilitó la implementación de las aplicaciones cliente (Textual Data Creator) y Mixer (NCL-Textual Data Mixer) a través de un *proxy* que permitió la invocación de cada método del servicio web evitando hacer uso de URL, mientras que REST permitió

retornar datos en texto plano a través del canal de retorno para presentar información al televidente.

- Por defecto, un servicio RESTful creado empleando la tecnología WCF, es únicamente capaz de retornar información en formato XML y JSON. Sin embargo, se puede modificar el formato de la información que retorna el servicio mediante el manejo de *streams* y la configuración en tiempo de ejecución del contexto actual del servicio RESTful.

  Esta funcionalidad fue aprovechada para retornar desde el lado del servicio web la información que solicita el televidente a través del canal de retorno como texto plano. Por otro lado, REST permitió prescindir de la envoltura XML del mensaje SOAP, con lo cual se evitó la necesidad de implementar en el *script* Lua, generado por la aplicación Mixer, funciones recursivas para obtener la información a presentarse al televidente y mostrarla en pantalla, lo cual redujo significativamente la complejidad del *script* y permitió obtener mejores resultados.

- A diferencia de otros buscadores, Google codifica los URL correspondientes a las páginas web de sus resultados de búsqueda, por lo que los URL obtenidos en una búsqueda con Google no pueden ser empleados directamente para analizar la información que contienen. Sin embargo, un URL puede ser decodificado tomando la cadena de texto inicial del mismo, comprendida antes del primer símbolo `&`, y haciendo un tratamiento a la cadena de texto adecuado.

  El *bot* de búsqueda desarrollado decodifica los URL de los resultados de búsqueda de Google, para posteriormente analizar la información de las páginas web contenidas en los sitios web indicados por el usuario. De esta manera, se logra presentar los resultados en un tiempo significativamente menor que al necesario si se exploraría cada sitio web individualmente sin hacer uso de Google.

- El IDE NCL Composer permite, a través del *plug-in* NCL Outline View que viene incluido, la inserción de metadatos, los cuales son empleados para describir a otros tipos de datos y así poder asociar las aplicaciones interactivas desarrolladas con el IDE con otras herramientas externas. El uso de metadatos permitió que la aplicación Mixer pueda generar los elementos media de las aplicaciones desarrolladas con el IDE empleando el *plug-in* Menu Creator.

- La creación de los elementos media, correspondientes a los nodos de contenido declarados dentro del documento NCL de una aplicación interactiva, es uno de los aspectos que más tiempo lleva en el desarrollo de contenido interactivo. Al hacer uso del sistema de búsqueda, almacenamiento y procesamiento de información se reduce significativamente el tiempo necesario para este proceso, pues la aplicación Mixer, además de ser capaz de generar los elementos media (imágenes en este caso) con el texto a presentarse en los menús, también genera los elementos media a ser presentados cuando el televidente selecciona las opciones de los menús, ya sea mediante imágenes PNG o mediante *scripts* Lua que obtienen la información a través del canal de retorno.

- SQL Server realiza operaciones sobre las bases de datos a través de sentencias declarativas. Con el fin de que el servicio WCF pueda ejecutar las acciones que el usuario le indica sobre las bases de datos, este debió convertirse en un cliente de SQL Server para así indicar mediante sentencias las acciones a realizarse.

  Por lo tanto, es el motor de SQL Server el responsable de realizar todas las acciones sobre las bases de datos y no el servicio web WCF.

- Al no contar un servicio web con una interfaz gráfica, son los clientes los encargados de proveer esta interfaz gráfica al usuario para que pueda consumir el servicio. La tecnología WCF permitió que la interfaz gráfica de usuario sea proveída a través de aplicaciones de escritorio, como la

aplicación cliente (Textual Data Creator) y Mixer (NCL-Textual Data Mixer), y no por tan solo navegadores (*browsers*). De esta manera, se consiguió desarrollar un sistema que permite al usuario hacer uso de aplicaciones fáciles de emplear, por su interfaz gráfica, y mucho más llamativas que al emplear únicamente un navegador para consumir el servicio web.

- El desarrollo de aplicaciones interactivas para televisión digital demanda de una gran cantidad de tiempo, no tan solo en la parte de programación, sino también en la calidad de la información que se presenta al televidente. El sistema de búsqueda, almacenamiento y procesamiento de información permitió desarrollar las aplicaciones interactivas en un tiempo significativamente menor que el necesario al emplear únicamente las herramientas existentes, al hacer uso de las distintas aplicaciones que el sistema ofrece. De igual manera, el sistema permitió crear sin complicaciones una segunda aplicación a partir de la primera, lo cual demuestra la flexibilidad que el sistema brinda para crear nuevas aplicaciones interactivas a partir de aplicaciones anteriormente desarrolladas, con el fin de presentar al televidente información actualizada o información referente a otras temáticas.

5.2 RECOMENDACIONES

- La base del funcionamiento del modelo NCM es el uso de nodos de contexto. Al trabajar con el IDE NCL Composer, todos las entidades creadas se anidan dentro de un nodo de contexto. De hecho, el cuerpo del documento NCL es un nodo de contexto más. Al desarrollar un *plug-in* para el IDE NCL Composer, se recomienda hacer que la funcionalidad del mismo se base en crear entidades agrupadas en nodos de contexto y no entidades independientes, al igual como el *plug-in* Menu Creator genera un nodo de contexto por cada menú desarrollado por el usuario. De esta manera, el *plug-in* podrá ser empleado con el resto de *plug-ins* originales de manera sencilla y le será de mayor utilidad al usuario.

- Por defecto, el *plug-in* Menu Creator dispone de cuatro plantillas de diseño. Si se va a desarrollar una plantilla personalizada, se recomienda que las imágenes que conforman la plantilla estén en formato PNG, ya que este es un formato de compresión sin pérdidas que permite visualizar las imágenes en la pantalla del televisor con buena calidad. El tamaño de las imágenes con formato PNG es razonable; sin embargo, si se desea reducir estos tamaños se recomienda comprimir las imágenes teniendo cuidado de no perder calidad en las mismas. Para ello, se puede hacer uso de programas de software libre especializados en la creación y edición de imágenes, como por ejemplo Gimp (*GNU Image Manipulation Program*), el cual brinda varias opciones para comprimir y crear imágenes de alta calidad con tamaños de archivo adecuados.

- Para obtener mejores resultados al hacer uso del *bot* de búsqueda, se recomienda ingresar el mayor número posible de palabras clave que más se aproximen al dato exacto que desea encontrar, así como también indicar varios sitios web de confianza y no tan solo uno, de manera que el *bot* de búsqueda pueda obtener información relevante.

- Para la búsqueda en sitios web, el servicio SOAP hace uso de la biblioteca `HTMLAgilityPack` y de un algoritmo de puntuaciones desarrollado en este proyecto para obtener los mejores resultados de la búsqueda que más se aproximen al dato exacto que el usuario está buscando. El algoritmo funciona correctamente y es de gran ayuda en el ingreso de información en las bases de datos.

  En la actualidad se impulsa la investigación de las tecnologías involucradas en la web semántica, la cual nació años atrás para enriquecer la búsqueda y la explotación de los resultados en la web. Si se desea mejorar el servicio de búsqueda, se recomienda implementar las técnicas basadas en la web semántica con el fin de obtener resultados más precisos.

- Aunque el *bot* de búsqueda desarrollado analiza los resultados obtenidos con el buscador de Google, la misma técnica de programación

implementada en la función de búsqueda del servicio SOAP podría ser empleada para analizar resultados de búsqueda obtenidos con otros buscadores. Si así se requiere, se recomienda modificar en el código fuente del servicio SOAP el URL del buscador a emplearse, y eliminar el código empleado para la decoficación de los URL de Google.

Por otro lado, el *bot* de búsqueda con la misma técnica de programación implementada podría obtener información de otros sitios web y no tan solo empleando buscadores. Se recomienda, si así se requiere, hacer uso del *bot* de búsqueda implementado para obtener información de otros sitios web, como por ejemplo redes sociales.

- El sistema de búsqueda, almacenamiento y procesamiento de información permite al usuario especificar la información que desea obtener mediante el canal de retorno. Se recomienda que esta información corresponda a información dinámica (en vivo), es decir a datos que cambian con el tiempo, ya que al cambiar la información bastaría con tan solo modificarla en la base de datos sin la necesidad de tener que modificar la aplicación interactiva.

  Se recomienda destinar información que puede ser obtenida mediante el canal de retorno si se requiere reducir el tamaño de la aplicación interactiva, de manera que esta pueda ser radiada por la interfaz de aire y ejecutada en el STB, para que cada televidente obtenga la información que desee sin que se entere de este detalle, siempre y cuando tenga disponible el canal de retorno.

- Para reducir el tiempo que tarda en presentarse la información obtenida mediante el canal de retorno, se recomienda alojar el servicio RESTful junto con el servicio de bases de datos de SQL Server en un mismo servidor. De esta manera, se evitará el tiempo innecesario al tener que realizar consultas sobre las bases de datos en un servidor remoto. De igual manera, se recomienda configurar el servicio web para que emplee una dirección de red numérica en la generación de los *scripts* Lua en lugar de

un nombre de dominio, para así evitar el tiempo innecesario en la resolución del nombre de dominio debido al empleo de un servidor DNS. De esta manera, se obtienen mejores resultados en el STB del televidente.

- Si el sistema de búsqueda, almacenamiento y procesamiento de información va a ser empleado en un entorno de producción multimedia, se recomienda alojar los servicios haciendo uso de Windows Server versión 2008 o posterior, el cual es un sistema operativo de servidor apropiado para el servicio WCF desarrollado. De igual manera, se recomienda hacer uso de la funcionalidad de trabajo colaborativo que el sistema ofrece mediante la creación de usuarios que ayuden a ingresar información en las bases de datos.

- Aunque la aplicación Mixer genera automáticamente *scripts* Lua, el sistema puede ser empleado para crear una aplicación interactiva híbrida creando únicamente su parte declarativa y, posteriormente, incorporando *scripts* Lua programados por el propio usuario. Si se va a desarrollar un *script* Lua, se recomienda codificar el archivo del *script* usando el formato ASCII normal, de manera que pueda ser ejecutado sin problemas en el STB del televidente. De igual manera, se recomienda hacer uso de la función `pcall` de Lua para controlar posibles excepciones, así como también hacer uso de las funciones para el manejo de cadenas que Lua ofrece, para así manejar el caracter de salto de línea `\n` el cual no es capaz de reconocer el STB.

5.3 COMENTARIOS

- En la actualidad, el desarrollo de aplicaciones interactivas para televisión digital se realiza haciendo uso de herramientas que únicamente usuarios con cierto nivel de conocimientos en lenguajes de programación pueden emplear correctamente. La creación de herramientas complementarias, así como de nuevas herramientas que permitan generar contenido interactivo de televisión digital en menor tiempo de desarrollo y sin necesidad de tener conocimientos avanzados en lenguajes de programación, como las

implementadas en este proyecto, es un aspecto importante aún en desarrollo que permitirá a cualquier tipo de usuario ser capaz de crear aplicaciones interactivas.

- El IDE NCL Composer es un IDE bastante flexible en el desarrollo de aplicaciones interactivas. Además de permitir el desarrollo de *plug-ins* para extender su funcionalidad, dentro de su módulo *extensions* dispone de dos interfaces a partir de las cuales se pueden crear *plug-ins* que permitan al IDE hacer uso de otros lenguajes declarativos empleados en televisión digital como SMIL y XHTML. De esta manera, el IDE abre la posibilidad de que a futuro pueda ser empleado para crear aplicaciones interactivas en distintos lenguajes de programación declarativos y no tan solo en NCL.

- El desarrollo de aplicaciones interactivas para televisión digital es un campo bastante promisorio, ya que además de programadores también se requiere de diseñadores gráficos y comunicadores sociales para lograr aplicaciones interactivas llamativas para el televidente y que presenten información de calidad.

REFERENCIAS BIBLIOGRÁFICAS Y ELECTRÓNICAS

[1] Blog Spot. Funciones inline en C++. [Online]. Disponible en: http://codigomaldito.blogspot.com/2005/12/funciones-inline.html.

[2] CDMarket. Linksys WRT120N. [Online]. Disponible en: http://www.cdmarket.com.ar/View/3184/linksys-wrt120n-wireless-n-home-router.aspx.

[3] CodePlex. HTMLAgilityPack. [Online]. Disponible en: http://htmlagilitypack.codeplex.com/.

[4] Consejo de Evaluación, Acreditación y Aseguramiento de la Calidad de la Educación Superior (CEAACES). Categorización Universidades. [Online]. Disponible en: http://www.ceaaces.gob.ec/index.php?option=com_content&view=article&id=19&Itemid=22.

[5] Consejo Nacional de Evaluación y Acreditación de la Educación Superior del Ecuador (CONEA). Evaluación del Desempeño Institucional de las Universidades y Escuelas Politécnicas del Ecuador. [Online]. Disponible en: http://www.educacionsuperior.gob.ec/wp-content/uploads/downloads/2012/07/Extracto_informe_CONEA.pdf**.**

[6] D. Mejía. (2012) Servicios WCF (Windows Communication Foundation). [Diapositivas]. Quito-Ecuador. Escuela Politécnica Nacional.

[7] D. Nehab. (2006, Abril) LuaSocket:HTTP Support. [Online]. Disponible en: http://w3.impa.br/~diego/software/luasocket/http.html.

[8] E. Granizo Montalvo, *Lenguaje C Teoría y Ejercicios*, Segunda edición ed., ESPE, Ed. Quito, Ecuador:Editorial ESPE, 1999.

[9] E. Méndez Rodríguez, "RDF: Un modelo de metadatos flexible", Dpto. de Biblioteconomía y Documentación Universidad Carlos III de Madrid, Madrid.

[10] EiTV Entretenimiento e Interactividad para la TV digital. EiTV Developer Box. [Online]. Disponible en: http://www.eitv.com.br/devbox_es.php.

[11] G. Colouris, J. Dollimore, T. Kindberg y G. Blair, *Distributed Systems Concepts and Design*, Quinta edición, Addison-Wesley, USA: Pearson, 2012.

[12] G. González. Servicios Web. [Online]. Disponible en: http://kalistog.wordpress.com/servicios-web.

[13] Ginga Bolivia. NCM 3.0 (Nested Context Model, Modelo de Contexto Anidado). [Online]. Disponible en: http://ginga.softwarelibre.org.bo/lib/exe/fetch.php?media=ncm.pdf.

[14] Git Foundaton. Git distributed even if your workflow isnt Documentation. [Online]. Disponible en: http://git-scm.com/documentation.

[15] iCarnegie, “Modelando Clases – Diagramas UML de clases”, Departamento de Ciencias de Computación Universidad Carnegie Mellon, Pittsburgh.

[16] J. Blanchette y M. Summerfield, *C++ GUI Programming with Qt 4*, Primera edición ed., Trolltech, Ed. Stoughton, USA:Prentice Hall, 2006.

[17] J. Conesa Caralt, A. Rius Gavidia, J. Ceballos Villach, y D. Gañán Jiménez, *Introducción a .NET*, Primera edición ed., Sónia Poch, Ed,: UOC, 2010.

[18] J. R. Groff y P. N. Weinberg, *Manual de Referencia SQL*, Segunda edición, Ed. Madrid, España: McGraw-Hill, 2003.

[19] J. Thelin, *Foundations of Qt Development*, Primera edición ed., Trolltech, Ed. Berkeley CA, USA:Apress, 2007.

[20] J. Torres. (2011, Junio) Comunidad Ginga Ecuador. [Online]. Disponible en: http://comunidadgingaec.blogspot.com/2011/06/middleware-ginga.html.

[21] Kioskeda net. La función inline en C++. [Online]. Disponible en: http://es.kioskea.net/faq/2823-la-funcion-inline-en-c.

[22] L. F. Gomes Soares y S. D. Junqueira Barbosa, *Programando em NCL 3.0*, Segunda edición ed., PUC-Rio, Ed. Río de Janeiro, Brasil, 2012.

[23] L. F. Gomes Soares, R. Ferreira Rodrigues, R. Rezende Costa, y M. Ferreira Moreno, "Nested Context language 3.0 Part 9 - NCL Live Editing Commands", Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Monografía en Ciencia de Computación ISSN 0103-9741.

[24] L. Gomes y R. Ferreira, "Nested Context Model 3.0 Part 1 - NCM Core", Pontificia Universidad Católica de Río de Janeiro, Río de Janeiro, Monografía en Ciencia de Computación ISSN 0103-9741.

[25] Lua Reference Manual. [Online]. Disponible en: http://pgl.yoyo.org/luai/i/about.

[26] Microsoft Technet. Motor de base de datos de SQL Server. [Online]. Disponible en: http://technet.microsoft.com/es-es/library/ms187875.aspx.

[27] Microsoft. (2013) Fundamentos de la depuración. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/aa290881(v=vs.71).aspx.

[28] Microsoft. BinaryFormatter (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.runtime.serialization.formatters.binary.binaryformatter(v=vs.110).aspx.

[29] Microsoft. Encoding (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.text.encoding%28v=vs.110%29.aspx.

[30] Microsoft. File (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.io.file(v=vs.110).aspx.

[31] Microsoft. HttpUtility (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.web.httputility%28v=vs.110%29.aspx.

[32] Microsoft. Introducción a las aplicaciones de Servicios Windows. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/d56de412%28v=vs.110%29.aspx.

[33] Microsoft. MIME Type Detection in Windows Internet Explorer. [Online]. Disponible en: http://msdn.microsoft.com/en-us/library/ms775147%28VS.85%29.aspx.

[34] Microsoft. SerializableAtributte (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.serializableattribute.aspx.

[35] Microsoft. SqlCommand (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.data.sqlclient.sqlcommand(v=vs.110).aspx.

[36] Microsoft. StreamWriter (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.io.streamwriter%28v=vs.110%29.aspx.

[37] Microsoft. System.Data.SqlClient (Espacio de nombres). [Online]. Disponible en:http://msdn.microsoft.com/es-es/library/system.data.sqlclient(v=vs.110).aspx.

[38] Microsoft. Tutorial: Crear una aplicación de servicios de Windows en el Diseñador de componentes. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/zt39148a%28v=vs.110%29.aspx.

[39] Microsoft. WebClient (Clase). [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/system.net.webclient%28v=vs.110%29.aspx.

[40] MSDN Magazine. An Introduction to RESTful Services with WCF. [Online]. Disponible en: http://msdn.microsoft.com/es-es/magazine/dd315413.aspx.

[41] Nokia Developer. Qt Reference Documentation Signals&Slots. [Online]. Disponible en: http://harmattan-dev.nokia.com/docs/platform-api-reference/xml/daily-docs/libqt4/signalsandslots.html.

[42] Nokia Developer. Qt SDK. [Online]. Disponible en: http://developer.nokia.com/Develop/Qt/Tools/.

[43] P. Deitel y H. Deitel, *C# 2010 for programmers*, Cuarta edición ed., RR Donnelley, Ed. Crawfordsville, USA: Prentice Hall, 2010.

[44] Qt Project. QtSettings Class. [Online]. Disponible en: http://qt-project.org/doc/qt-5.0/qtcore/qsettings.html.

[45] Qt Project. Signals&Slots. [Online]. Disponible en: http://qt-project.org/doc/qt-5.0/qtcore/signalsandslots.html.

[46] R. Carvalho, J. A. Ferreira dos Santos, J. Ribeiro Damasceno, J. Varanda Da Silva, y D. C. Muchaluat Saade, *Introducao ás Linguagens NCL e Lua: Desenvolvendo Aplicacoes Interactivas para TV Digital*, Primera edición ed., Universidad Federal Fluminense, Ed. Brasil: Laboratorio MídiaCom y Peta5, 2009.

[47] R. Ierusalimschy, L. Henrique de Figueiredo, y W. Celes. (2011, Septiembre) Manual de Referencia de Lua 5.1. [Online]. Disponible en: http://www.lua.org/manual/5.1/es/manual.html.

[48] R. Ierusalimschy. Lua Programming Error Handling and Exceptions. [Online]. Disponible en: http://www.lua.org/pil/8.4.html.

[49] S. Cheng, *Microsoft Windows Communication Foundation Cookbook for Developing SOA Applications*, Primera edición, Ed. Loncoln Road, USA: Packt Publishing, 2010.

[50] StackOverFlow Forum. WCF ResponseFormat for WebGet. [Online]. Disponible en: http://stackoverflow.com/questions/992533/wcf-responseformat-for-webget.

[51] Superintendencia de Telecomunicaciones del Ecuador. (2011, Noviembre) Preguntas frecuentes sobre Televisión Digital. [Online]. Disponible en: http://www.supertel.gob.ec/index.php?option=com_content&view=article&id=211:preguntas-frecuentes-sobre-television-digital&catid=61:articulos-recomendados&Itemid=311.

[52] T. García. SISTEMAS DISTRIBUIDOS RESTful. [Online]. Disponible en: http://tedhygarcia.blogspot.com/2012/06/restful.html#!/2012/06/restful.html.

[53] TeleMídia (PUC-Rio). AttributeReferences Class. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1util_1_1AttributeReferences.html.

[54] TeleMídia (PUC-Rio). Como compilar o NCL Composer a partir do código-fonte. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/pt-br/doc/tutorial/how_to_build_ncl_composer_from_source_code.

[55] TeleMídia (PUC-Rio). ComposerSetting.cpp. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/ComposerSettings_8cpp_source.html.

[56] TeleMídia (PUC-Rio). ComposerSettings utilities. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1util_1_1Utilities.html.

[57] TeleMídia (PUC-Rio). Entity Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1model_1_1Entity.html.

[58] TeleMídia (PUC-Rio). How to: build NCL Composer from source code. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/doc/tutorial/how_to_build_ncl_composer_from_source_code.

[59] TeleMídia (PUC-Rio). IDocumentParser Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IDocumentParser.html.

[60] TeleMídia (PUC-Rio). ILanguageProfile Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1ILanguageProfile.html.

[61] TeleMídia (PUC-Rio). IPlugin Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IPlugin.html.

[62] TeleMídia (PUC-Rio). IPluginFactory Abstract Interface. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1extension_1_1IPluginFactory.html.

[63] TeleMídia (PUC-Rio). LanguageControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1LanguageControl.html.

[64] TeleMídia (PUC-Rio). Lua Módulo canvas. [Online]. Disponible en: http://www.telemidia.puc-rio.br/~francisco/nclua/referencia/canvas.html.

[65] TeleMídia (PUC-Rio). MessageControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1MessageControl.html.

[66] TeleMídia (PUC-Rio). MessageControl Member List. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1MessageControl-members.html.

[67] TeleMídia (PUC-Rio). NCL Composer Plugins. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/plugins/start.

[68] TeleMídia (PUC-Rio). NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/_media/screenshot/nclcomposer-0.1.5-01.png.

[69] TeleMídia (PUC-Rio). PluginControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1PluginControl.html.

[70] TeleMídia (PUC-Rio). PluginControl.cpp. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/PluginControl_8cpp_source.htmlhttp://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1PluginControl.html.

[71] TeleMídia (PUC-Rio). Project Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1model_1_1Project.html.

[72] TeleMídia (PUC-Rio). ProjectControl Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1ProjectControl.html.

[73] TeleMídia (PUC-Rio). ProjectReader Class NCL-Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/doc/classcomposer_1_1core_1_1ProjectReader.html.

[74] TeleMídia (PUC-Rio). Welcome to NCL Composer. [Online]. Disponible en: http://composer.telemidia.puc-rio.br/en/start?redirect=1#welcome_to_ncl_composer.

[75] Visual Studio. Sintaxis de XPATH. [Online]. Disponible en: http://msdn.microsoft.com/es-es/library/vstudio/ms256471%28v=vs.100%29.aspx.

[76] Wikinsonpc. Codificando y decodificando una dirección URL. [Online]. Disponible en: http://www.wilkinsonpc.com.co/free/articulos/codificar-decodificar-url.html.

[77] Wikipedia. Archivos de cabecera. [Online]. Disponible en: http://es.wikipedia.org/wiki/Archivo_de_cabecera.

[78] Wikipedia. Biblioteca de enlace dinámico. [Online]. Disponible en: http://es.wikipedia.org/wiki/Biblioteca_de_enlace_din%C3%A1mico.

[79] Wikipedia. HypertextTransferProtocol. [Online]. Disponible en: http://es.wikipedia.org/wiki/Hypertext_Transfer_Protocol.

[80] Wikipedia. Microsoft SQL Server. [Online]. Disponible en: http://es.wikipedia.org/wiki/Microsoft_SQL_Server.

[81] Wikipedia. MinGW. [Online]. Disponible en: Wikipedia. MinGW. [Online]. Disponible en: http://es.wikipedia.org/wiki/MinGW.

[82] Wikipedia. Qt (Biblioteca). [Online]. Disponible en: http://es.wikipedia.org/wiki/Qt_%28biblioteca%29.

ANEXOS

Los anexos se incluyen en el DVD que acompaña este documento

ANEXO A: *Plug-in* de generación automática de menús Menu Creator

- **ANEXO A.1:** Código fuente
- **ANEXO A.2:** Instalador
- **ANEXO A.3:** Manual de instalación
- **ANEXO A.4:** Manual de usuario
- **ANEXO A.5:** Videotutorial
- **ANEXO A.6:** Video de ejemplo del desarrollo de una aplicación interactiva empleando el *plug-in* Menu Creator

ANEXO B: Servicio web WCF

- **ANEXO B.1:** Código fuente (Incluye código fuente del servicio SOAP, RESTful y servicio de Windows para búsqueda periódica de Información)
- **ANEXO B.2:** Instalador (Incluye instalador del servicio SOAP con el servicio de Windows para búsqueda periódica de información e instalador del servicio RESTful)
- **ANEXO B.3:** Manual de instalación

ANEXO C: Aplicación cliente (Textual Data Creator)

- **ANEXO C.1:** Código fuente
- **ANEXO C.2:** Instalador
- **ANEXO C.3:** Manual de instalación
- **ANEXO C.4:** Manual de usuario
- **ANEXO C.5:** Videotutorial

ANEXO D: Aplicación Mixer (NCL-Textual Data Mixer)

- **ANEXO D.1:** Código fuente
- **ANEXO D.2:** Instalador
- **ANEXO D.3:** Manual de instalación
- **ANEXO D.4:** Manual de usuario
- **ANEXO D.5:** Videotutorial

ANEXO E: Aplicaciones interactivas, Base de Datos y Guía de usuario

- **ANEXO E.1:** Aplicación interactiva de las Islas Galápagos
- **ANEXO E.2:** Aplicación interactiva de Educación Superior del Ecuador (universidades categoría A, 2009-2013)
- **ANEXO E.3:** Aplicación interactiva de Educación Superior del Ecuador (universidades categoría A, 2013-actualidad)
- **ANEXO E.4:** Base de datos de Educación Superior del Ecuador
- **ANEXO E.5:** Manual para incorporar la base de datos de Educación Superior del Ecuador a SQL Server
- **ANEXO E.6:** Guía de usuario de las aplicaciones interactivas de Educación Superior del Ecuador

ANEXO F: STB híbrido EiTV Developer Box

- **ANEXO F.1:** Manuales de usuario
- **ANEXO F.2:** Especificaciones técnicas

ANEXO G: Papers

- **ANEXO G.1:** Paper del *plug-in* Menu Creator (enviado a 5th IEEE Latin American Symposium on Circuits and Systems LASCAS 2014)
- **ANEXO G.2:** Paper del *plug-in* Menu Creator (enviado a 12th LACCEI Latin American and Caribbean Conference for Engineering and Technology, July 22-24 2014/Organizado por Latin American and Caribbean Consortium of Engineering Institutions/Conference Proceedings indexadas en EBSCO Publishing/Aceptado para ser indexado, archivado on-line, publicado y expuesto en la conferencia de julio de 2014)
- **ANEXO G.3:** Paper del sistema de búsqueda, almacenamiento y procesamiento de información (enviado a Revista Politécnica indexada en Latindex/Indexado, archivado on-line y publicado en la edición 33 volumen 3 de enero de 2014)