Testing Automático de Atributos NCL Utilizando Reconocimiento de Patrones

Alejandro Alvarez
LIFIA, Facultad de Informática, Universidad Nacional de La Plata
50 y 115 1er Piso
La Plata, Argentina
alejandro.alvarez@lifia.info.unlp.edu.ar

Federico Balaguer
LIFIA, Facultad de Informática, Universidad Nacional de La Plata
50 y 115 1er Piso
La Plata, Argentina
federico.balaguer@lifia.info.unlp.edu.ar

RESUMEN

Ginga es una parte fundamental de la especificación de la norma ISDB-T. Ginga permite que el televidente aumente su experiencia por medio de contenidos interactivos. Uno de los lenguajes que permite especificar estos contenidos es NCL. Este lenguaje declarativo permite manejar dónde y cómo se mostrarán contenidos multimediales.

Aquellos que desarrollen o mantengan implementaciones de Ginga deben asegurarse del correcto funcionamiento de los objetos MEDIA. Esto suele llevarse adelante mediante el uso de pruebas de regresión. Si se trata de pruebas visuales y las mismas son realizadas de forma manual, se observan dos problemas: confiabilidad y costo.

Este trabajo presenta una alternativa que permite probar el posicionamiento de objetos MEDIA en forma automática. Para esto se hace uso de técnicas de reconocimiento de patrones, que aunque conocidas, son utilizadas de una manera novedosa y eficiente.

Términos Generales

Ginga, NCL, Lua, Testing, Visual Computing, pyopencv

Palabras clave

Ginga, NCL, Lua, Testing, Visual Computing, pyopencv

1. INTRODUCCIÓN

Asegurarse que una implementación de Ginga[2] se ajusta al *standard* es una tarea difícil de implementar y sostener en el tiempo. Ésto se debe a la gran cantidad de horas hombre que demanda tanto en pruebas funcionales como en pruebas de regresión.

Este artículo describe como probar el correcto funcionamiento de los atributos del lenguaje NCL[8]. Más específicamente se detalla la experiencia adquirida al probar los atributos de posicionamiento en objetos MEDIA sobre la versión de Ginga en la que está trabajando el LIFIA [1].

Esta implementación ejecuta aplicaciones escritas en NCL. NCL es un lenguaje declarativo que permite especificar aspectos de interactividad y sincronismo entre elementos de presentación como HTML, video, imagen y sonido. La mayoría de estos objetos MEDIA tienen atributos de posicionamiento que permiten especificar su lugar y tamaño en una pantalla. La necesidad de contar con una herramienta de *testing* automático surge del trabajo realizado para completar la implentación del *middleware* en lo que respecta a atributos de posicionamiento. Este proceso de automatización no fue directo, sino que es resultado de la evolución en la manera de realizar las pruebas. Este trabajo no intenta presentar una comparación exhaustiva de los distintos métodos de *testing*, manuales y automáticos. La propuesta que se presenta es la de hacer uso de técnicas de reconocimiento de patrones para automatizar por completo la ejecución de este tipo de pruebas.

En la sección 2, se plantea la necesidad de probar estas propiedades con el objetivo de llevar al *middleware* a un grado de mayor madurez. También se estudian los atributos NCL y como fueron evolucionando las pruebas hasta llegar a automatizarlas por completo. En las sección 3 se verá como utilizando técnicas de reconocimiento de imágenes podemos saber si un objeto MEDIA se está mostrando en una posición específica de la pantalla y con un tamaño determinado. También se analiza por que fue necesario que los *test cases* estén sincronizados con la ejecución de la aplicación NCL. En la sección 4 se comparan los resultados obtenidos y como cada variable en juego en el proceso de *testing* fue cambian-

[1]LIFIA (Laboratorio de Investigación y Formación en Informatica Avanzada). El Laboratorio es parte de la Facultad de Informática de la Universidad Nacional de La Plata. La página web del Laboratorio es http://lifia.info.unlp.edu.ar

do durante la evolución del mismo. Finalmente se presentan las conclusiones y trabajo a futuro.

2. TESTING DE ATRIBUTOS NCL

El lenguaje NCL consta de distintos componentes para poder construir una aplicación. La mayoría de éstos tienen una característica común, se pueden definir atributos en cada uno de ellos. Por ejemplo, en un componente REGION se puede definir el atributo *height*. En esta sección se introducen los atributos NCL y luego se describe la evolución que siguió el proceso de *testing* visual hasta llegar a su completa automatización.

2.1 Atributos de los objetos NCL

El lenguaje declarativo NCL permite especificar aspectos de interactividad y sincronismo espacial/temporal entre objetos MEDIA. Entre los componentes más representativos podemos citar a REGION, DESCRIPTOR, MEDIA, LINK, CONNECTOR, entre otros [2, 8]. A su vez sobre estos objetos, el lenguaje permite especificar atributos, con los cuales se pueden describir los límites, posición o comportamiento del objeto en cuestión. En la Figura 1 se puede ver la definición de un componente REGION con varios atributos. Ésta es una definición estática de los atributos, la cual aplica a los objetos MEDIA que utilicen esta REGION.

```
1  <region id="bgRegion"
2          left="0"
3          top="0"
4          width="100%"
5          height="50%"/>
```

Figura 1: Definición de una región NCL

Los atributos de posicionamiento, como muchos otros del lenguaje NCL, pueden variar en tiempo de ejecución. En la Figura 2, se puede ver como se redefine la propiedad *left* (línea 4), y como la misma se puede volver a modificar al presionar el botón rojo (línea 12).

```
1  <media descriptor="bgDesc"
2         id="bg"
3         src="images/background.png">
4    <property name="left" value="20%">
5  </media>
6  ...
7  <link xconnector="onKeySelectionSet">
8    <bind component="bg" role="onSelection">
9      <bindParam name="keyCode" value="RED"/>
10   </bind>
11   <bind component="bg" interface="left" role="set">
12     <bindParam name="varSet" value="30%" />
13   </bind>
14 </link>
```

Figura 2: Definición de atributos de un objeto Region

Estos atributos además de variar en tiempo de ejecución pueden combinarse, haciendo que los casos de prueba sean más complejos. Sobre ésto, la norma ABNT[1] en uno de sus párrafos menciona:

> *"When the user specifies top, bottom and height information for the same <region>, spatial inconsistencies can occur. In this case, the top and height values shall have precedence over the bottom value."*

Al analizar este conjunto de variantes es que surge la necesidad de realizar un amplio conjunto de pruebas visuales para saber si un objeto MEDIA está mostrándose o no en el lugar correspondiente.

2.2 Testing visual utilizando imágenes

En la implementación de Ginga sobre la que se realizó el estudio, los atributos de posicionamiento se aplican a superficies. Estas superficies pueden contener objetos MEDIA, por ejemplo una imagen, un video, un texto. Las pruebas se realizaron utilizando imágenes.

Para probar los atributos de posicionamiento fue necesario construir varias aplicaciones NCL que ejercitan las propiedades *left, top, right, botton, width, height* sobre imágenes. Estas aplicaciones fueron orientadas a probar estos atributos individualmente o combinados, de forma estática y dinámica. Estas aplicaciones permitieron contar con un conjunto de pruebas predefinido para realizar las pruebas visuales de manera manual. Nótese que el costo de implementar los casos de prueba es considerable, pero éstos son reutilizables cada vez que sea necesario realizar las pruebas sobre una implementación de Ginga.

En las primeras aplicaciones realizadas todos los atributos se probaron utilizando siempre la misma imagen. Estas aplicaciones al recibir un determinado evento del control remoto ejercitaban ciertos atributos de posicionamiento y una persona entrenada determinaba si la imagen se estaba mostrando de manera correcta o no. En este punto se observan dos desventajas:

1. Es necesario tener un *training* para poder realizar las pruebas, dado que la persona encargada de *test* debe saber cual es la posición correcta en que la imagen debería estar mostrándose. Esto implica un *overhead* de tiempo para este tipo de pruebas.

2. El *testing* visual como el que se necesita en este caso, requiere de un grado de atención muy alto. Ésto hace que la confiabilidad de la prueba no sea muy alta y que no pueda realizarse durante un tiempo muy prolongado [9].

2.3 Testing con imágenes autodescriptivas

El siguiente paso fue intentar reducir el tiempo de *training* necesario para poder realizar las pruebas. Para ello se comenzaron a usar imágenes autodescriptivas en lugar de utilizar siempre la misma imagen. El objetivo de esta técnica es indicar a la persona encargada de realizar la prueba en que lugar debe estar la imagen, agregando información en la misma imagen. En primera instancia se utilizó texto para agregar la información complementaria. En la Figura 3, podemos ver un ejemplo de esta alternativa para la prueba del atributo *top*.

Esta alternativa tuvo tres concecuencias:

Figura 3: Prueba del atributo *top* utilizando texto para la información adicional

Figura 4: Error de posicionamiento del atributo *top*

- El tiempo de *training* necesario para realizar las pruebas se redujo.
- La necesidad de tener una imagen autodescriptiva para cada *test case*, hizo que el tiempo de generación del *test case* se incrementara notoriamente.
- Al intentar probar atributos combinados, describir con un texto la posición donde debe verse la imagen es bastante complejo. Por ejemplo, describir la imagen que se forma del código de las Figuras 1 y 2, sería algo como:

  "Top, desplazado un 20 % desde la izquierda"

En este punto, el objetivo fue seguir utilizando imágenes autodescriptivas, intentando resolver el problema encontrado al probar los atributos combinados. Para ello, en lugar de utilizar un texto para la información complementaria se utilizó otra imagen. De esta manera la imagen original, contiene otra imagen la cual representa la región de la pantalla donde se debería estar viendo.

La Figura 4 muestra esta alternativa para la caso de prueba del atributo *top*. En la misma se puede observar un error, dado que la información complementaria indica que la imagen debería estar mostrándose en la parte superior de la pantalla. De esta manera se pudo utilizar la técnica de imágenes autodescriptivas para cubrir los casos de atributos combinados. Un desventaja del uso de imágenes para agregar la información complementaria es que el tiempo necesario para generar cada caso de prueba se incrementó con respecto a la opción de utilizar texto.

2.4 Testing visual automático

Hasta aquí la necesidad de *training* se había podido reducir considerablemente con el uso de imágenes autodescriptivas. No obstante lo cual, había dos problemas sin solución: confiabilidad de las pruebas y el costo de repetirlas. Una posible solución viene del hecho que en los intentos anteriores el patrón de las imágenes se repite tuviera o no información complementaria. En este punto es que el problema se reduce a sólo detectar patrones en pantalla, lo cual se puede realizar automáticamente.

3. TESTING AUTOMÁTICO BASADO EN PATRONES

En está sección se detalla el procedimiento que permite automatizar por completo las pruebas de los atributos de posicionamiento. Se plantea como utilizando técnicas de reconocimiento de imágenes es posible saber si una imagen se está mostrando en una posición específica de la pantalla y con un tamaño determinado. También se analiza por qué es necesario especificar los datos de la región a detectar en un *test case*, el cual necesariamente debe estar sincronizado con la ejecución de la aplicación NCL/Lua.

3.1 Arquitectura de la plataforma de pruebas

La plataforma de pruebas pasó a tener además de un STB (Set Top box) y una TV, una *webcam* de resolución media y un equipo dedicado para procesar el video y ejecutar sobre estas capturas los casos de prueba (Figura 5). La ejecución de un caso de prueba se compone de los siguientes pasos:

- Indicar a la aplicación NCL/LUA que aplique a la imagen los atributos que el *test case* va a analizar.
- Procesar durante n segundos la captura de video, buscando detectar una región que el *test case* tiene definida con una determinada posición y tamaño.
- En este punto si la región es detectada, el caso de prueba se considera existoso y se continúa con el siguiente.

Un *test case* está compuesto por dos partes, cada una de ellas ejecutándose en equipos diferentes, las cuales necesariamente deben estar sincronizadas. Por un lado tenemos la aplicación NCL que necesita de una indicación para aplicar los atributos a probar. Por otra parte, es necesario definir cual es el tamaño y la posición de la región que debe detectarse, con la cual se define el *assert* del *test case*. A conti-

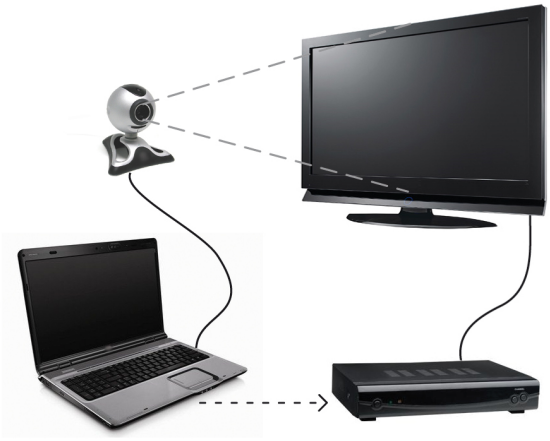

Figura 5: Esquema de la plataforma de pruebas

nuación se detalla cada uno de estos pasos y las tecnologías utilizadas.

3.2 Descripción de la aplicación NCL/Lua

La aplicación que se ejecuta en el STB se compone de una parte NCL y una parte Lua. En NCL se cuenta con un objeto MEDIA al cual se le modifican los atributos de posicionamiento durante la ejecución de las pruebas. El componente Lua es quien recibe eventos de teclado para sincronizar la ejecución de los *test cases*. El objeto MEDIA presenta una imagen en pantalla la cual es detectada haciendo uso del reconocimiento automático de imágenes. Para indicar la ejecución del próximo caso de prueba, se utilizan eventos de control remoto enviados automáticamente como se verá a continuación. El componente Lua, además de recibir estos eventos, define la lógica que determina cual es el próximo caso de prueba y haciendo uso de un *attribution event*[6, 7] indica a la componente NCL que atributo del objeto MEDIA modificar.

3.3 Sincronización entre la ejecución de un test case y Ginga

Como se mencionó anteriormente es necesario que el *test case* esté sincronizado con la aplicación NCL/LUA que está ejecutándose en el STB. Esta aplicación puede recibir eventos a través del control remoto. Para poder realizar ésto de manera automática se utilizó Lirc [10], más especificamente la herramienta *irsend* que nos permite enviar comandos a través de la red. De esta manera podemos enviar un evento antes de ejecutar cada *test case*, el cual es procesado por la componente Lua de la aplicación.

3.4 Definición de test cases en Python

Para la definición de las pruebas de unidad, utilizamos el lenguaje Python [16], más específicamente el *framework* unittest. El mismo soporta la automatización de pruebas y permite especificar el *setup* y *shutdown* para cada *test case*. En el *setup* de cada *test case* es donde se produce la sincronización con la aplicación NCL/Lua.

Para definir un *test case* es necesario especificar una región, la cual se intentará encontrar durante un tiempo preestablecido, para determinar si el resultado del *test case* es el esperado. Para definir una región es necesario especificar en porcentajes el *x,y* inicial de la región y su *width, height*. Esto coincide con una de las formas de especificar los *bounds* de un objeto NCL, lo cual no es casual dado que la intención fue hacer lo más directo posible la construcción del *test case*.

El *test case* para el atributo *top* lo podemos ver en la Figura 6. En la línea 2 se crea un objeto Python que representa una región que comienza en el (0,0) y tiene un *width* del 100 % y un *height* del 50 % de la pantalla.

```
1  def test_top(self):
2          region = self.createRegion(0,0,1,0.5)
3          self.assertTrue(self.findRegion(region))
```

Figura 6: Definición del test case para la propiedad top

3.5 Reconocimiento automático de regiones

Como se mencionó anteriormente para poder automatizar completamente el *testing* de atributos de posicionamiento recurrimos a reconocimiento automático de regiones. Para el procesamiento de imágenes se utilizó la librería **pyopencv** [15], la cual provee *bindings* desde Python para OpenCV 2 [14]. Esta tecnología nos permite reconocer patrones en las imágenes capturadas en tiempo real.

Haciendo uso de las funcionalidades provistas por el *framework* se construyó un reconocedor de regiones. El mismo es capaz de detectar durante el procesamiento de video rectángulos blancos sobre un fondo negro. De esta manera, no se necesita generar un imagen para cada *test case*, sino que alcanza con usar siempre una imagen blanca, reduciendo el tiempo necesario para la generación de cada caso de prueba.

Por cada región detectada el reconocedor nos permite saber su posición y tamaño. Con esta información es posible saber si la región definida en el *test case* coincide con la detectada y de esta manera saber si el resultado de la prueba fue o no el esperado.

Para reducir el margen de error de las pruebas automáticas es necesario calibrar la cámara frente a la pantalla. Ésto requiere que se realicen pruebas preliminares para ajustar el escenario donde se llevarán a cabo los *test cases*. Además en la definición de los *test cases* Python, es posible especificar un margen de error en *pixels*. El mismo representa la distancia euclidiana máxima permitida entre los vértices de la región detectada y la región del *test case*. De esta manera es posible ajustar el grado de precisión con que se detecta una región en un *test case*. Este margen se podrá reducir en caso de contar con cámaras de mejor resolución.

4. COMPARACIÓN DE LAS TÉCNICAS DE PRUEBA

A continuación se comparan las técnicas descriptas en la secciones 2 y 3. Las variables que permiten evaluar la técnica presentada son:

- El costo de generación de cada aplicación NCL/Lua
- El nivel de *training* requerido para realizar cada tipo de prueba
- El tiempo requerido para ejecutar cada *test case* según el tipo de prueba

En la Tabla 1 se puede ver una comparación entre estas variables según el tipo de prueba:

Tipo de prueba	Costo de generación	Necesidad de *training*	Tiempo de procesamiento
A. imagen	0	Alta	Alto
B. imagen + texto	Medio	Media	Medio
C. imagen + imagen	Alto	Media	Medio
D. testing automático	0	0	Bajo $\approx$ 250ms

Tabla 1: Comparación entre los tipos de *test cases*

El costo de generar las aplicaciones NCL/Lua, no es igual para todos los casos. Además de tener que generar imágenes con información complementaria para los casos B y C, es necesario codificar en el documento NCL la acción de aplicar los atributos a probar y el cambio de la imagen. Es importante destacar que las aplicaciones generadas son reutilizables, ésto hace que los costos que esta variable representa se vayan reduciendo con cada reuso de la aplicación.

El tiempo de *training* está directamente relacionado al tiempo de procesamiento. Ésto es debido a que para casos más complejos se necesita un mayor nivel de *training* y el tiempo necesario para ejecutar estas pruebas de manera manual es mayor. Por ejemplo para el caso A, es necesario que la persona encargada de realizar la prueba pueda comprender el código de la aplicación NCL o disponga de un documento que le indique el resultado esperado.

Además del tiempo de procesamiento, hay que tener presente la desventaja que tiene realizar el *testing* visual de manera manual. Ya sean una o más personas, el hecho de tener que ejecutar una tarea repetitiva implica que el tiempo de la prueba no puede ser muy extenso debido al grado de atención requerido. El costo de procesamiento de un *test case* en las pruebas automáticas ronda los 250ms. Durante este tiempo, se envía el evento via *irsend*, se actualiza la aplicación NCL/Lua, se ejecuta el *test case*, se capturan y procesan imágenes de video para analizar si se está mostrando en la pantalla la imagen de manera correcta. Este tiempo de procesamiento es mínimo comparado con el tiempo que se necesita para realizar las pruebas visuales de manera manual.

5. CONCLUSIONES Y TRABAJO A FUTURO

Las implementaciones de Ginga son aplicaciones complejas en las que el desarrollo debe realizarse utilizando prácticas frecuentes de *testing*. Esta necesidad se ve potenciada en proyectos basados en una comunidad donde se pueden recibir aportes de terceros para integrar al *middleware*. Una parte importante del problema es que los humanos tienden a cometer errores en tareas monótonas y repetitivas como son las de probar atributos NCL.

Este trabajo presentó una infraestructura de *testing* basada en reconocimiento de patrones, simulación del envío de teclas del control remoto y definición de *test cases* de manera directa a partir de la aplicación NCL. Con ésto, es posible evaluar el correcto funcionamiento de los atributos de posicionamiento de una implementación de Ginga. A pesar que otras pruebas como *benchmarking* o *testing* específico del *rendering* no son aplicables a esta herramienta, la mejora en términos de resultados y utilización de recursos humanos es significativa.

A partir de los resultados obtenidos se puede observar que haciendo uso de distintas tecnologías es posible automatizar por completo el proceso de *testing* de los atributos de posicionamiento. Ésto permite poder continuar con el desarrollo del *middleware* teniendo un conjunto de pruebas que permite realizar pruebas funcionales y de regresión de manera muy ágil.

La automatización permite obtener ganancias notables en el tiempo requerido para la generación de los casos de prueba y la ejecución de los mismos. Sin embargo, el método automático tiene un grado de flexibilidad menor al manual, por ejemplo en el tipo de objeto MEDIA a utilizar. Además la fiabilidad de los *test cases* puede verse afectada ante un error en la calibración del esquema de pruebas de no haber personas observando su ejecución.

Se continuará con la extensión de esta herramienta para abarcar una mayor cantidad de atributos del lenguaje NCL, por ejemplo la propiedad VISIBLE[2, 8]. Así también, se estudiará la posibilidad de automatizar pruebas que permitan detectar si acciones como START, STOP sobre una MEDIA funcionan correctamente[].

6. BIBLIOGRAFÍA

[1] Associação Brasileira de Normas Técnicas. *ABNT NBR 15606-2: GingaNCL para Receptores Fixos e Móveis*.

[2] S.˜D.˜J. Barbosa and L.˜F. Gomes˜Soares. *TV Digital Interativa no Brasil se faz com Ginga*. JAI, 2008.

[3] B.˜Beizer. *Black-box testing: techniques for functional testing of software and systems*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[4] P.˜Cesar, D.˜C.˜A. Bulterman, and L.˜F. Gomes˜Soares. Introduction to special issue: Human-centered television—directions in interactive digital television research. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(4):1–7, 2008.

[5] R.˜Ferreira˜Rodrigues and L.˜F. Gomes˜Soares. *ProduÃ§Ã£o de ConteÃºdo Declarativo para TV Digital*.

[6] L.˜F. Gomes˜Soares, M.˜Ferreira˜Moreno, and F.˜Sant'Anna. Relating declarative hypermedia objects and imperative objects through the ncl glue language. In *Proceedings of the Symposium on*

Document Engineering, 2009.

[7] L.˜F. Gomes˜Soares, R.˜Ferreira˜Rodrigues, R.˜Cerqueira, and S.˜D.˜J. Barbosa. Variable handling in time-based xml declarative languages. In *2009 ACM Symposium on Applied Computing*, 2009.

[8] L.˜F. Gomes˜Soares, R.˜Ferreira˜Rodrigues, and M.˜Ferreira˜Moreno. Ginga-ncl: the declarative environment of the brazilian digital tv system. In *Journal of the Brazilian Computer Society, vol. 12; No. 4, Mars 2007; pp. 37-46. ISSN: 0104-6500*, 2007.

[9] P.˜Li, T.˜Huynh, M.˜Reformat, and J.˜Miller. A practical approach to testing gui systems. *Empirical Softw. Engg.*, 12(4):331–357, 2007.

[10] Lirc homepage. http://www.lirc.org.

[11] Lua homepage. http://www.lua.org/.

[12] D.˜Marinov. *Automatic Testing of Software with Structurally Complex Inputs*. PhD thesis, Massachusetts Institute of Technology, 2004.

[13] R.˜Monteiro, R.˜Costa, M.˜Ferreira˜Moreno, and L.˜F. Gomes˜Soares. Intermedia synchronization management in dtv systems. In *ACM 2008 Symposium on Document Engineering,*, 2008.

[14] Opencv homepage. http://opencv.willowgarage.com/.

[15] M.-T. Pham, Y.˜Gao, V.-D.˜D. Hoang, and T.-J. Cham. Fast polygonal integration and its application in extending haar-like features to improve object detection. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, San Francisco, California, Jun 2010.

[16] Python homepage. http://www.python.org/.