jGUIAr: una herramienta para la enseñanza y el aprendizaje de interfaces gráficas en Java™

Molina, Hernán
Facultad de Ingeniería, Universidad Nacional de La Pampa

Abstract
La enseñanza del paradigma Orientado a Objetos (OO) a estudiantes de carreras informáticas que han aprehendido los conceptos y experiencias de programación en el paradigma procedural es un desafío no menor. Adicionalmente a dicha complejidad, y particularmente con el uso del lenguaje de programación Java™, el desafío es aún mayor cuando se introducen por primera vez los conceptos involucrados en la creación de interfaces gráficas de usuario. En la literatura existen enfoques y herramientas que pueden ser aplicados a la enseñanza de la programación OO. Sin embargo no existen herramientas apropiadas para asistir en la enseñanza y/o aprendizaje de los conceptos involucrados en la creación de interfaces gráficas en dicho paradigma. En este artículo se presenta jGUIAr, una herramienta diseñada y creada para asistir en la enseñanza y aprendizaje de la construcción de interfaces gráficas con el paquete AWT del lenguaje Java™. Se describen además un conjunto de actividades prácticas a desarrollar por el estudiante utilizando la herramienta presentada.

Palabras Clave
programación, paradigma orientado a objetos, herramienta, interfaz gráfica, AWT

Introducción

Cuando un estudiante incorpora por primera vez los conocimientos y experiencias de la programación de aplicaciones de software a través del paradigma procedural, el aprendizaje posterior del paradigma orientado a objetos (OO) constituye un desafío importante [2,10] ya que el estudiante debe cambiar la forma de pensar el problema y la solución, incorporando nuevos conceptos y estructuras mentales. Entre éstos, el estudiante debe entender el concepto de objeto como la representación programática de un individuo tomado de la realidad, muchas veces abstracto, y a la vez como valor complejo de una variable que posee su propio estado y comportamiento; que los objetos se definen a partir de una clase como abstracción o modelo de la realidad; que éstas pueden ser definidas a partir de otras clases para ser reusadasy aplicadas eficientemente para lograr características de extensibilidad y modificabilidad del código; que la comunicación entre los objetos se realiza mediante mensajes intercambiados entre ellos, constituyendo estos los bloques de construcción de cualquier programa de software OO; entre otros conceptos. Adicionalmente, se deben comprender los procesos y técnicas ingenieriles necesarias para obtener los requerimientos de un cliente y diseñar una solución utilizando apropiadamente los nuevos conceptos.

Si se considera además que, durante este proceso de incorporación de los conceptos del paradigma OO, el estudiante es introducido por primera vez a la creación y programación de interfaces gráficas utilizando el mismo paradigma, el desafío es aún mayor. Aquí el estudiante debe aplicar los conceptos propios del paradigma OO para comprender los diferentes tipos de componentes de una interfaz de usuario, los mecanismos y estructuras de agregación que permiten componer una interfaz, las estrategias de organización o disposición de los componentes agregados y la forma en que estos pueden ser utilizados para obtener el comportamiento deseado para la aplicación de software. Tal es la situación de la cátedra *Programación Orientada a Objetos* de las carreras de informática de la *Facultad de Ingeniería* de la *Universidad Nacional de La Pampa*, en la cual se utiliza el lenguaje *Java™* para las prácticas de

laboratorio y se incorpora, por primera vez en la currícula de la carrera, la creación y programación de interfaces gráficas, en este caso, mediante los paquetes *AWT* (*Abstract Window Toolkit*) y *Swing.*

El aspecto metodológico y pedagógico de la enseñanza del paradigma OO ha sido tratado ya por diversos autores [2,5,6,8,10,11] de la misma forma que el uso de herramientas que asistan a diversas metodologías [3,1,7]. No obstante, ya que la enseñanza del paradigma OO no constituye el tema principal de este trabajo no se hará mención a las diferentes propuestas más allá de citar aquellas relacionadas al tema central de este artículo.

Respecto de la enseñanza y aprendizaje de la creación de interfaces gráficas en el paradigma OO, particularmente en *Java™*, no se encuentran en la literatura propuestas que provean enfoques o herramientas de soporte. Por otro lado, existen un número de herramientas que buscan acortar las distancias entre el código y la apariencia final de la interfaz, aunque éstas se alejan del propósito de este trabajo.

En este artículo se presenta una herramienta, llamada *jGUIAr* (por *Java™ Graphical User Interface Architect*), diseñada y creada para asistir al profesor y al estudiante en la enseñanza y aprendizaje, respectivamente, de la creación de interfaces gráficas en *Java™* mediante el uso del paquete *AWT*. La herramienta se enfoca en los conceptos involucrados en la creación de interfaces gráficas más que en la apariencia deseada, recurriendo a abstracciones gráficas previamente conocidas y ofreciendo mecanismos que aceleren el ciclo programación-compilación-ejecución para comprobar rápidamente los resultados de la interfaz creada. Se describen además, como soporte metodológico, un conjunto de actividades prácticas que el estudiante puede desarrollar con el uso de la herramienta para incorporar, de forma progresiva e incremental, los diferentes conceptos involucrados en la creación de interfaces gráficas.

El resto del artículo se estructura de la siguiente forma: a continuación se enumeran las particularidades de las interfaces gráficas por las cuales se argumenta la necesidad de una herramienta para asistir en su enseñanza y aprendizaje; luego se describen brevemente algunas herramientas que tratan con la creación de interfaces gráficas para *Java™;* luego se describe la herramienta *jGUIAr* creada para cubrir dicha necesidad; a continuación se presenta una guía de actividades didácticas que pueden ser realizadas por los estudiantes utilizando *jGUIAr* para incorporar los conocimientos y experiencia en la construcción de interfaces gráficas en *AWT*. Finalmente se enuncian conclusiones y trabajos futuros.

Interfaces gráficas en Java™

Como se mencionó en la introducción, *Java™* ofrece más de una librería para generar interfaces gráficas. Entre las utilizadas en la cátedra de *Programación Orientada a Objetos* (recordar el contexto introducido en la introducción) se encuentran *AWT* y *Swing*. La decisión de enfocarse en *AWT* para enseñar el tema reside en que *AWT* es más simple de comprender que *Swing*, ya que éste último introduce el modelo *Model View Controler* [4], y una vez que el estudiante ha comprendido los conceptos principales, puede dar el salto a *Swing* con menores dificultades. No obstante, *AWT* presenta igualmente un desafío al momento de comprender la estructura y funcionamiento de sus componentes. Primero, *AWT* cuenta con un número importante, aunque básico, de controles gráficos, todos ellos organizados y relacionados en una jerarquía de herencia mediante la cual establecen estados y comportamientos comunes (ver Figura 1). Segundo –y a partir de dicha jerarquía de herencia-, los contenedores (como *Panel*) pueden ser agregados de

forma anidada para lograr diversas interfaces.

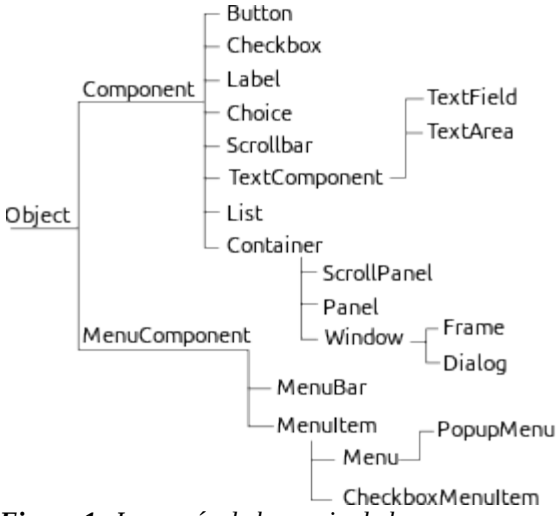

***Figura 1**: Jerarquía de herencia de los componentes gráficos de la librería AWT*

Tercero, los componentes agregados en cada contenedor pueden ser dispuestos de diversas formas, utilizando los gestores de disposición o *layout managers* que provee la librería, cada uno con sus reglas y opciones de configuración particulares. El estudiante debe incorporar todos estos nuevos conceptos estructurales y de composición para comenzar a crear sus propias interfaces gráficas.

Adicionalmente, el caso con *Java™* presenta otra dificultad al respecto: por defecto, una interfaz gráfica se crea desde el código fuente, a diferencia de lenguajes como *C#* donde el entorno de desarrollo sólo permite la edición de la interfaz gráfica mediante un editor *WYSIWYG (What You See Is What You Get).* Esto implica que el estudiante deba escribir una cantidad relativamente importante de líneas de código, previendo el diseño deseado, hasta poder compilar y ejecutar el programa y ver realmente el resultado. Esto produce una distancia entre la codificación y la interfaz gráfica resultante que dificulta al estudiante establecer una relación entre el código escrito y el resultado real.

Trabajos relacionados

Existe un importante número de herramientas cuyo propósito es facilitar, de alguna forma, la creación de interfaces gráficas en *Java™*. Uno de los enfoques más comunes encontrados es el uso de herramientas WYSIWYG en las cuales el usuario "dibuja" la interfaz gráfica deseada utilizando los componentes gráficos disponibles. Algunas de las herramientas que siguen este enfoque son *WindowBuilder* de *Eclipse*[1], *Swing GUI Builder* de *Netbeans*[2] y *Swing GUI Designer* de *IntellijIDEA*[3]. Todas ellas se enfocan en ocultar al programador los detalles del código fuente y en mantener los ajustes realizados al diseño una vez que la aplicación final sea ejecutada. Este enfoque es útil cuando la aplicación se sigue manteniendo con el mismo entorno con el que se generó la interfaz, ya que el código fuente necesario para generar la misma suele encontrarse sobrecargado de ajustes de aspecto avanzados en los componentes, difícil de analizar y comprender si nos encontramos en un entorno de aprendizaje; más aún considerando que el estudiante debe poder relacionar el código fuente con la interfaz generada. *GuiGenie*[4] es otra de las herramientas existentes bajo el mismo enfoque en este caso destinada a principiantes ya que apunta a la simplicidad de la herramienta de diseño y a generar código fuente "limpio". No obstante posee algunas desventajas que la descartan para ser usada en un entorno de aprendizaje: por un lado, recomienda utilizar un *layout null* –en el que los componentes se deben posicionar usando coordenadas gráficas en lugar de ubicaciones lógicas (enfoque que el *Tutorial de Java™ de Oracle*[5] no recomienda porque dificulta la tarea de diseño y produce interfaces poco flexibles a los cambios del entorno de ejecución)- y,

1 http://www.eclipse.org/windowbuilder/
2 https://netbeans.org/features/java/swing.html
3 http://www.jetbrains.com/idea/features/gui_builder.html
4 http://www.guigenie.com
5 http://docs.oracle.com/javase/tutorial/

por otro lado, no permite la agregación anidada de paneles para componer la interfaz –muy útil para lograr diseños flexibles y diversos.
Otro de los enfoques encontrados en estas herramientas es el uso de documentos *XML (eXtensible Markup Language)*, con una estructura predefinida, que son procesados en tiempo de ejecución generando la interfaz gráfica deseada. Algunas de las herramientas existentes de este tipo son *jXUL*[6], *Luxor XML UI Language*[7] y *SwingML*[8]. La desventaja principal de este enfoque para ser utilizado en un entorno de aprendizaje es que se debe aprender un nuevo lenguaje y reglas sintácticas para componer la interfaz, además de ocultar el código fuente *Java™* que producirá la misma, complicando aún más la situación para el estudiante.
Es importante mencionar que la situación respecto de herramientas enfocadas en la enseñanza y aprendizaje de la programación OO es similar. Existen muchos entornos de programación enfocados más en los aspectos sintácticos del lenguaje, y en mecanismos para facilitar y acelerar la escritura del código, que en los aspectos conceptuales propios del paradigma OO. Por otro lado, y como una excepción, el entorno de desarrollo *BlueJ* [7] fue diseñado con una filosofía enfocada en la enseñanza y aprendizaje de los conceptos del paradigma OO. En este entorno el estudiante manipula –además del código fuente- clases, objetos, sus estados y mensajes enviados mediante abstracciones gráficas. Esta filosofía fue la utilizada para diseñar *jGUIAr*, la herramienta presentada en este trabajo.

jGUIAr: de los conceptos al código

La herramienta permite diseñar y crear interfaces gráficas utilizando los componentes más comunes de la librería o paquete *AWT* de *Java™*. Con el propósito de asistir en la enseñanza y aprendizaje de la construcción de interfaces gráficas, en lugar de utilizar una representación gráfica idéntica a la interfaz deseada, la interfaz gráfica es diseñada de forma lógica, creando y agregando los componentes según las reglas de composición del paquete *AWT*. En todo momento el estudiante puede ver código fuente de la interfaz gráfica diseñada y exportarlo para ser compilado y ejecutado en tiempo de diseño, pudiendo ver los cambios realizados a medida que se construye la interfaz.
Para diseñar la interfaz gráfica de forma lógica, *jGUIAr* utiliza metáforas gráficas que representan la agregación de componentes que conformarán la interfaz final. La metáfora utilizada en este caso es una estructura jerárquica de tipo árbol *n-ario* en la que los nodos representan objetos o instancias de alguna de las clases de componentes del *AWT* (ver Figura 2).

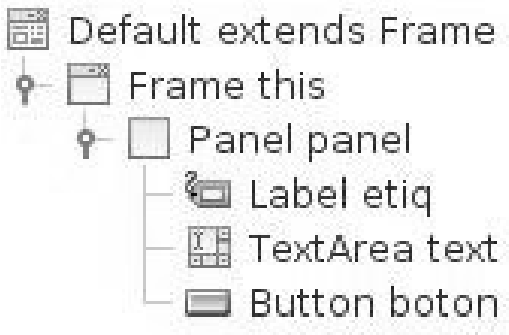

***Figura 2**: Estructura que representa la agregación de componentes de la interfaz diseñada.*

Los nodos que representan instancias de contenedores, como ventanas o paneles, pueden contener otros nodos. Los nodos que corresponden a instancias de otros componentes, tales como botones, áreas de texto, entre otros, constituyen los nodos hoja de la estructura. En particular, la raíz del árbol corresponde a la ventana de la aplicación (un objeto de la clase *Frame*) que contiene agregados al resto de los componentes de la interfaz. Adicionalmente, diferentes *layouts* pueden

6 http://jxul.sourceforge.net/
7 http://luxor-xul.sourceforge.net
8 http://swingml.sourceforge.net

ser creados y asignados a los contenedores deseados, para luego configurar la disposición de los componentes de un contenedor siguiendo las reglas del *layout* correspondiente.

Para crear y editar cada uno de los componentes de la interfaz se utiliza un formulario simple en el que se deben completar los parámetros requeridos por alguno de los constructores del componente correspondiente. De esta forma, se busca familiarizar al estudiante con los constructores y términos encontrados en el *API* (*Abstract Programming Interface*)[9] correspondiente a las clases de la librería *AWT*.

El código fuente generado por la herramienta se mantiene limpio y conciso en el sentido de que no agrega elementos o configuraciones que no fueron previstas en el diseño. Además el código se presenta de forma ordenada y comentado, buscando mostrar y animar al estudiante a producir códigos de las mismas características en sus aplicaciones.

La metáfora utilizada para diseñar la interfaz facilita la comprensión de la estructura de una interfaz gráfica ya que es una metáfora conocida por el estudiante, es decir, aquella utilizada para representar la estructura de directorios y archivos en un sistema operativo. Por otro lado, la analogía gráfica que ofrece el árbol de componentes utilizado para diseñar la interfaz, más la posibilidad de ver el código fuente y ejecutar la aplicación resultante en tiempo de diseño, acortan la distancia existente entre la codificación y la interfaz gráfica resultante, facilitando al estudiante establecer una relación entre el código escrito y el resultado real.

La herramienta no busca ocultar los detalles de configuración de la interfaz diseñada ni el código fuente que la genera, a la vez que ofrece un mecanismo intuitivo, simple y rápido de producir la interfaz gráfica deseada. Así, se espera que el estudiante comprenda y aprehenda rápidamente los conceptos y experiencia relacionada a la creación de interfaces gráficas en *Java™* con *AWT*.

Descripción de la herramienta

jGUIAr fue desarrollada en *Java™*, bajo una licencia *GNU GPL* (*General Public License*)[10].

La interfaz de *jGUIAr* se compone de cuatro grupos de controles, según se muestra en la Figura 3:

- *Componentes de la interfaz*: Lista los componentes creados mostrando el nombre de la clase a la que pertenece y el nombre del identificador con el que aparecerá en el código fuente. Un menú contextual permite editar o eliminar el componente.
- *Jerarquía de agregación de componentes*: Muestra la agregación de los componentes de la interfaz Cada componente se muestra también usando el nombre de la clase a la que pertenece y su identificador, y ofrece el mismo menú contextual mencionado en el punto anterior.
- *Área de configuración de componentes*: Muestra los formularios que permiten editar la configuración de los componentes así como la vista previa del código fuente de la interfaz diseñada.
- *Barra de herramientas*: Contiene accesos rápidos para la creación de *layout managers* y para previsualizar el código fuente de la interfaz.

La aplicación crea por defecto interfaz gráfica de ejemplo que extiende de la clase *Frame.* (ver nodo etiquetado "*Ejemplo extends Frame*" con un sub-elemento "*Frame this*" correspondiente a un objeto de tipo *Frame).* El objeto *Frame this* tiene agregado un contenedor de tipo *Panel* que a su vez tiene agregados tres componentes. Los componentes de la interfaz gráfica son creados usando el menú contextual

9 http://docs.oracle.com/javase/7/docs/api/

10 La herramienta será publicada una vez que se su utilidad haya sido probada.

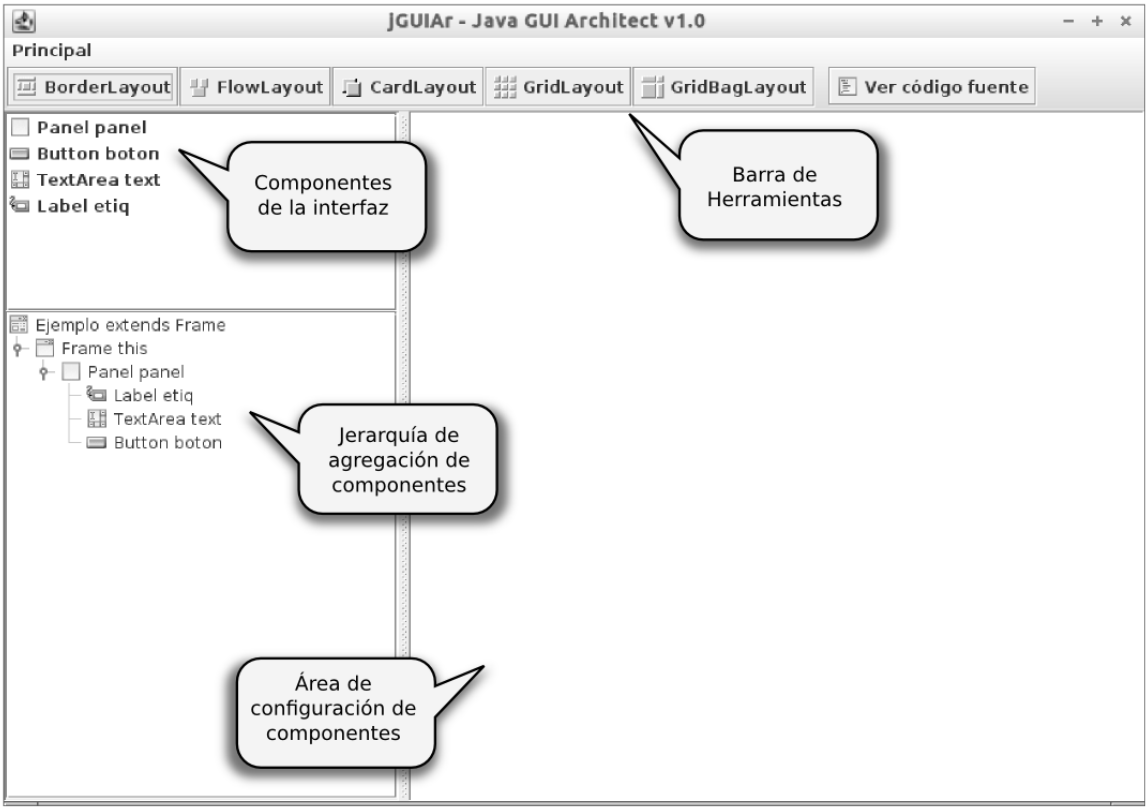

***Figura 3**: La interfaz de edición de jGUIAr*

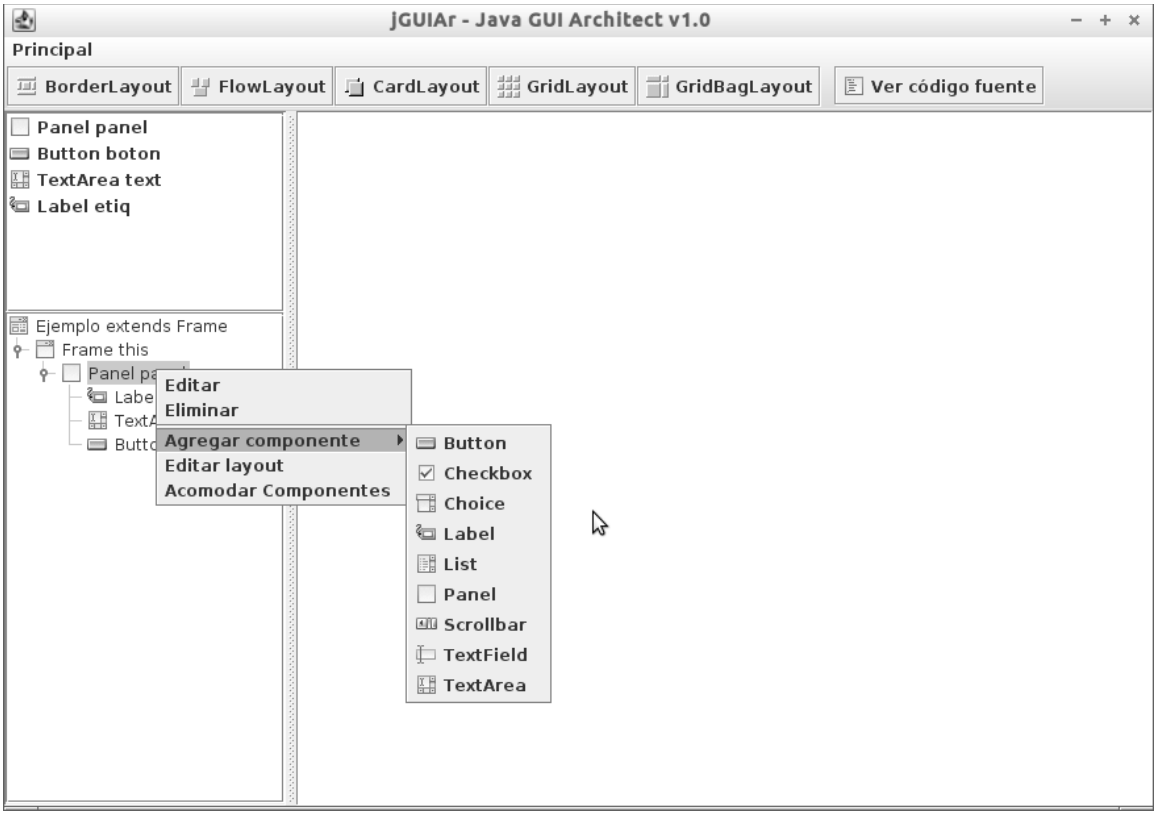

***Figura 4**: La herramienta permite agregar a un contenedor los componentes permitidos.*

"*Agregar componente*" de los contenedores, como se muestra en la Figura 4, y configurados mediante el formulario correspondiente que se muestra en el *Área de configuración de componentes*. El componente creado será agregado al contenedor cuyo menú contextual fue utilizado para crearlo. Note en la figura citada que un objeto de la clase *Panel* puede ser agregado a un contenedor, consiguiendo así la agregación anidada de los mismos.

Como sucede en *AWT*, los paneles son creados utilizando un *layout manager* de tipo *FlowLayout* por defecto. Para crear nuevos *layout managers* pueden utilizarse los botones de la *Barra de Herramientas* o el menú contextual del componente de más alto nivel de la *Jerarquía de Agregación de Componentes* y luego editar el contenedor deseado para seleccionar el nuevo *layout* creado (ver Figura 5). Una vez que se han agregados los componentes a un contenedor y seleccionado un *layout manager* para el mismo se puede configurar la disposición de los componentes del contenedor usando las opciones o restricciones provistas por el *layout manager* seleccionado, accediendo a la opción "*Acomodar Componentes*" del menú contextual del contenedor. Para cualquier contenedor es posible, independientemente del *layout manager* seleccionado, cambiar el orden en que serán agregados sus componentes (ver Figura 6). Las restricciones a asignar a cada componente corresponden también a objetos y atributos tomados del *API* correspondiente a la clase del *layout manager* seleccionado.

Una vez que la interfaz ha sido diseñada se puede previsualizar el código fuente de la clase *Java™* resultante (ver Figura 7) y exportar el código fuente y compilar y ejecutar la aplicación (ver Figura 8).

Los diseños realizados pueden ser guardados para su posterior edición en archivos siguiendo un formato específico basado en *XML*.

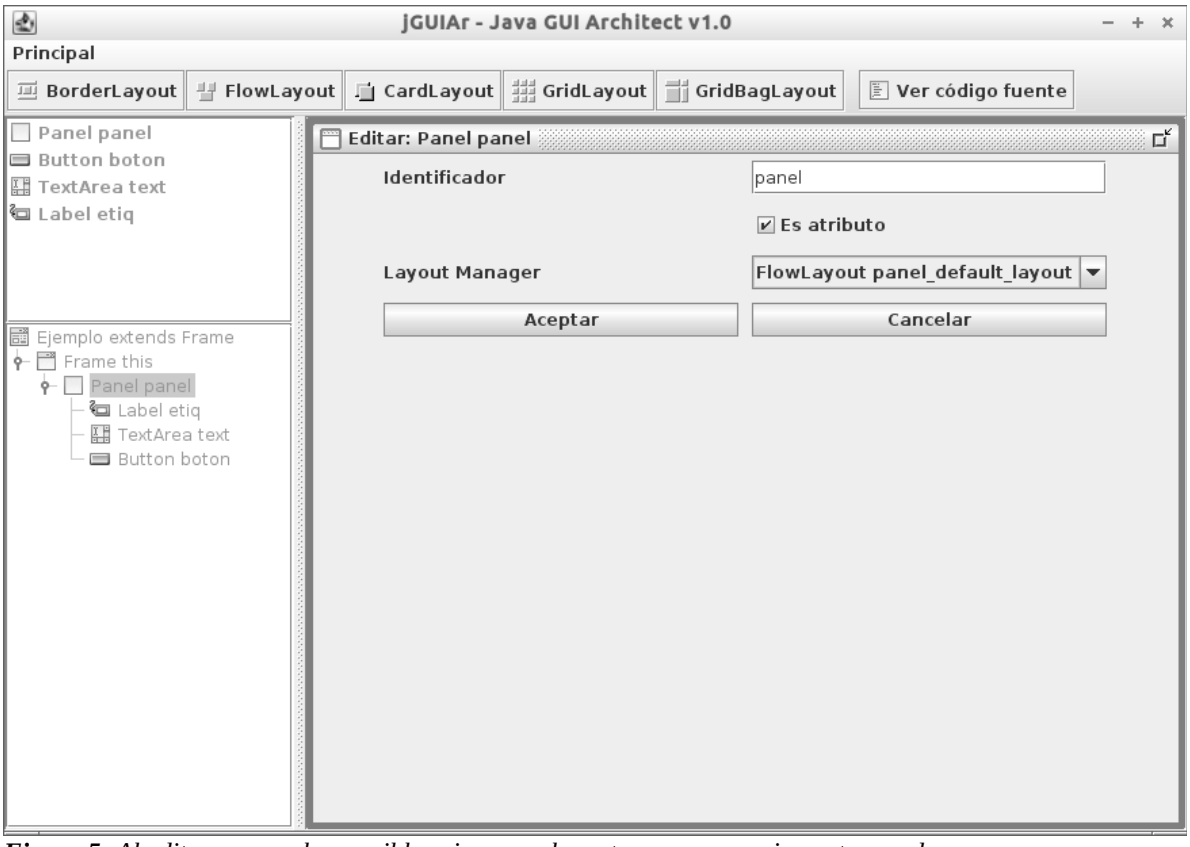

***Figura 5**: Al editar un panel es posible asignar un layout manager previamente creado.*

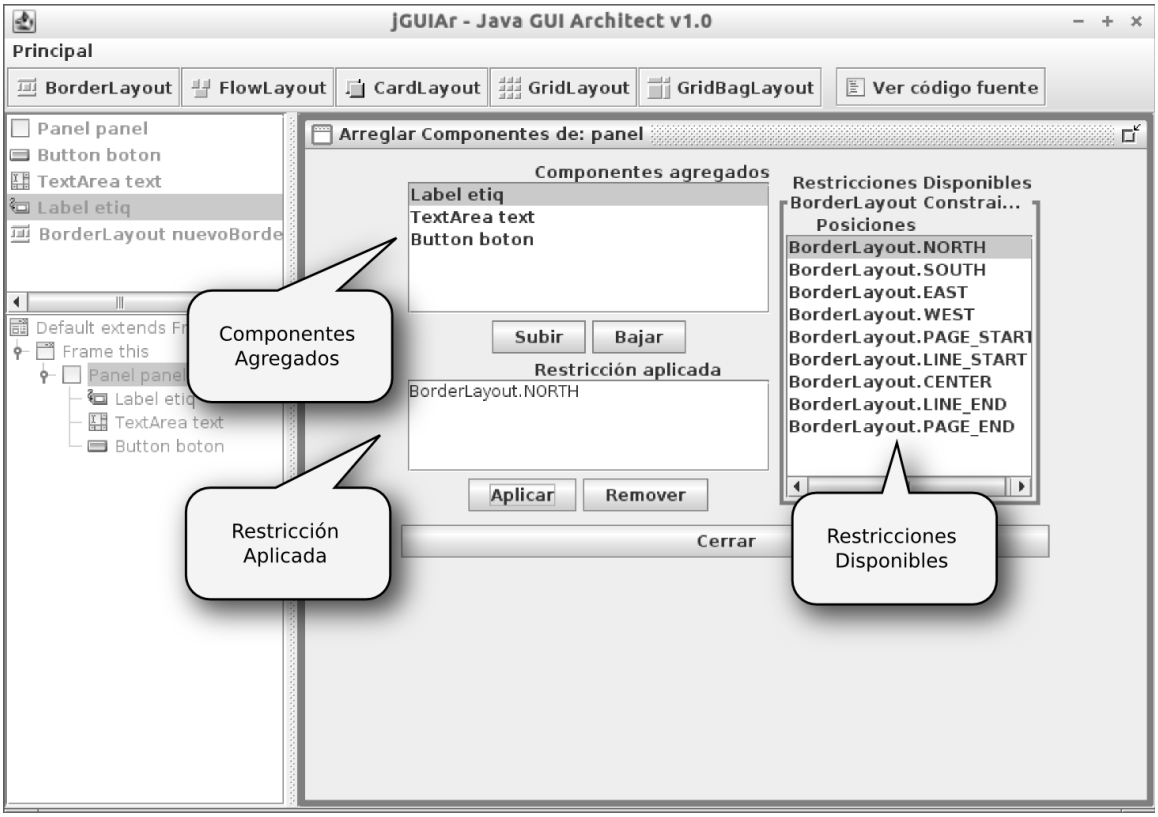

***Figura 6**: La herramienta permite disponer los componentes agregados a un contenedor siguiendo las reglas del layout manager asignado.*

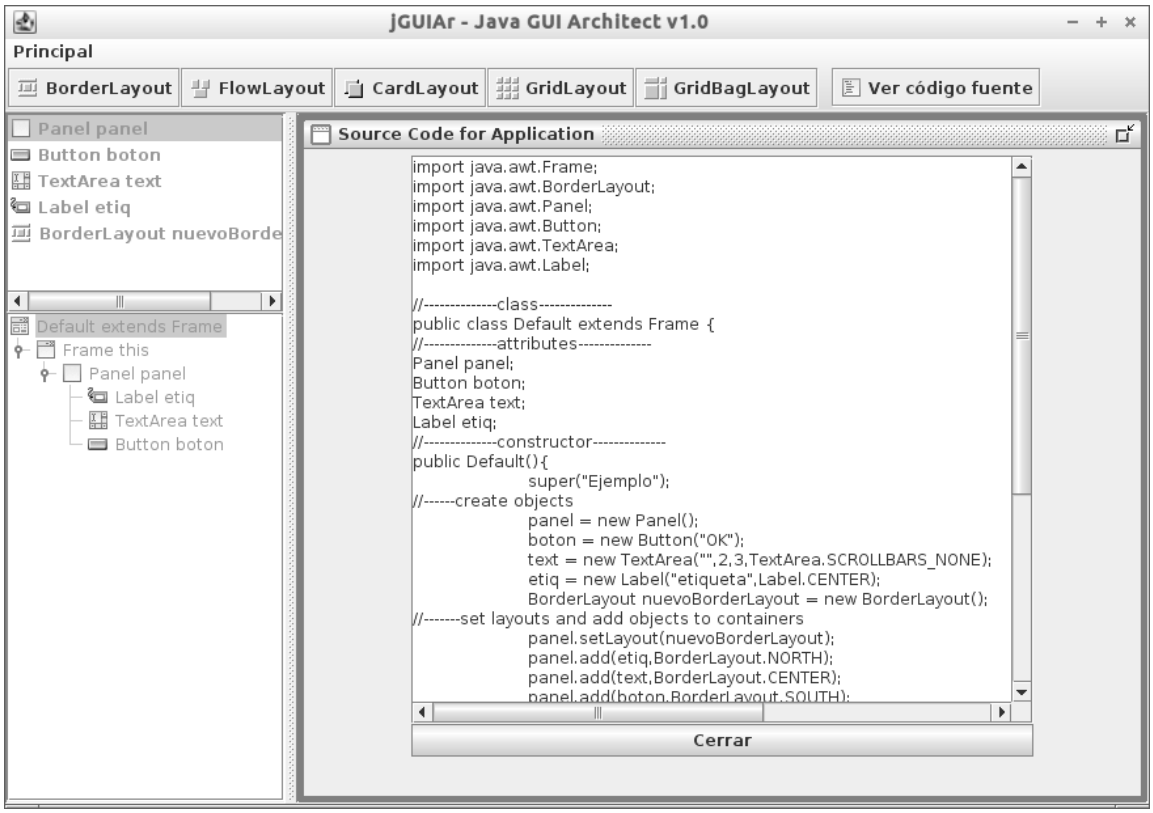

***Figura 7**: El código fuente que genera la interfaz gráfica diseñada puede ser previsualizado en tiempo de diseño.*

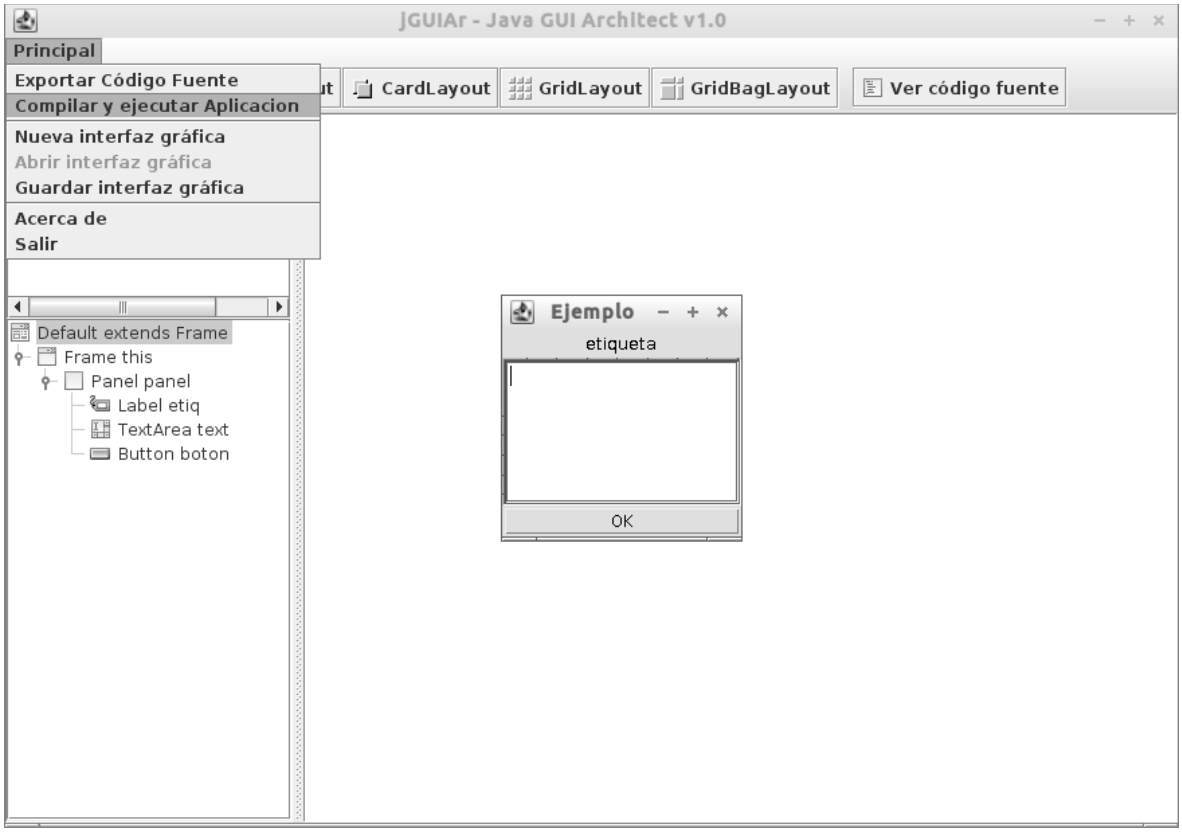

***Figura 8**: jGUIAr permite compilar y ejecutar la aplicación para ver la interfaz resultante en tiempo de diseño.*

Secuencia de actividades

Se han diseñado un conjunto de actividades a ser desarrolladas por el estudiante, utilizando la herramienta como guía en el aprendizaje de la creación de interfaces gráficas con *AWT*. Estas actividades fueron diseñadas siguiendo la filosofía de enseñanza citada en la sección *Trabajos relacionados*, incorporando nuevos desafíos de forma iterativa e incremental. Para estas actividades se utilizan diferentes interfaces gráficas previamente creadas con *jGUIAr* –distribuidas junto con la herramienta- a partir de las cuales los estudiantes realizan las actividades.

A continuación se describen los tipos de actividades que le son propuestas al estudiante de forma secuencial:

1. *Explorar*: se provee al estudiante con ejemplos de interfaces gráficas creadas en la herramienta con el propósito de explorar los diferentes componentes de una interfaz gráfica y las diferentes opciones utilizadas que generan la interfaz resultante. Se espera que el estudiante comprenda la estructura de objetos utilizados en la creación de interfaces gráficas y las diferentes opciones de configuración. Los ejemplos utilizados son simples y se utilizan interfaces gráficas que le son familiares al estudiante. No se pide que modifiquen los ejemplos.
2. *Configurar*: se pide al estudiante que haga pequeñas modificaciones a los ejemplos dados, tales como cambiar la configuración de los componentes ya existentes. Así se busca que el estudiante se familiarice con las opciones de configuración más utilizadas de los diferentes componentes del *AWT* y vea qué efectos producen los cambios realizados en la interfaz gráfica resultante.

3. *Agregar componen*tes: se solicita al estudiante que modifique los ejemplos provistos agregando nuevos componentes a los contenedores ya existentes en los ejemplos dados. Aquí se busca introducir nuevos componentes gráficos y sus opciones de configuración así como experimentar con los cambios que se producen en la interfaz gráfica al agregar estos componentes. No se busca producir interfaces gráficas correctas sino explorar las consecuencias de agregar nuevos componentes a una interfaz gráfica. La herramienta permite observar estos cambios de forma rápida.
4. *Cambiar disposición*: se pide al estudiante que modifique los ejemplos provistos cambiando la disposición de los componentes en los contenedores existentes. Se busca con estas tareas que el estudiante se familiarice con los diferentes *layout managers* del *AWT*, sus opciones de configuración y cómo son dispuestos los diferentes componentes con cada uno de los *layout manager*. Aquí se solicita al estudiante que obtenga una interfaz gráfica más apropiada y mejor diseñada que la propuesta por el ejemplo original.
5. *Agregar contenedores y componentes*: se pide al estudiante agregar nuevos contenedores con sus respectivos componentes, utilizando la disposición y configuración de componentes que considere más apropiada al propósito de la aplicación solicitada. Con esta tarea se busca que el estudiante obtenga sus primeras experiencias en componer interfaces gráficas en *AWT*.
6. *Crear interfaz*: Posteriormente, como ejercicio avanzado, se pide a los estudiantes que creen una interfaz gráfica desde cero a partir de imágenes de las interfaces que deben obtener como resultado (creada con la misma aplicación) y de enunciados que establecen los requerimientos que deben satisfacer la interfaces. Deben crear todos los componentes y configurar la disposición de los mismos seleccionando las opciones apropiadas.

Conclusiones y Trabajos futuros

En este artículo se ha presentado una herramienta diseñada y creada para asistir en la enseñanza y aprendizaje de la construcción de interfaces gráficas con el paquete *AWT* del lenguaje *Java*™. La herramienta sigue un enfoque orientado a la enseñanza de los conceptos involucrados, complementando metodologías más apropiadas al paradigma OO que las comúnmente usadas para el paradigma procedural. Se espera que la herramienta presentada facilite la comprensión de la estructura de una interfaz gráfica a través de metáforas gráficas conocidas por el estudiante, ofreciendo a la vez un mecanismo intuitivo, simple y rápido de producir la interfaz gráfica deseada. La posibilidad de ver el código fuente de la interfaz diseñada y ejecutar la aplicación correspondiente en tiempo de diseño buscan acortar la distancia existente entre la codificación y la interfaz gráfica resultante, permitiendo al estudiante establecer una relación entre el código fuente generado y el resultado real. El código fuente generado se mantiene limpio y ordenado buscando mostrar y animar al estudiante a producir códigos de las mismas características en sus aplicaciones.

Además, se describieron un conjunto de actividades prácticas que pueden ser desarrolladas de forma iterativa e incremental por el estudiante, utilizando la herramienta como guía hacia la comprensión y adquisición del conocimiento necesario para construir sus propias interfaces gráficas. En resumen, con el uso de la herramienta se espera que el estudiante comprenda y aprehenda rápidamente los conceptos y experiencia relacionada a la creación de interfaces gráficas en *Java*™ con *AWT*.

Como trabajo futuro se utilizará *jGUIAr* en la cátedra *Programación Orientada a Objetos* de las carreras de informática de la *Facultad de Ingeniería* de la *Universidad Nacional de La Pampa*, en el período lectivo 2014 (la herramienta estuvo disponible posteriormente a la finalización de la cursada del período 2013), y se entregará la herramienta a los estudiantes, junto con una guía práctica diseñada según los lineamientos presentados en éste artículo, para analizar luego las ventajas de su utilización respecto de la comprensión de interfaces gráficas en *Java™*. En dicho período, también se diseñará y realizará una evaluación de calidad en uso (incluyendo satisfacción y usabilidad) para analizar la utilidad de la herramienta y efectuarle las mejoras pertinentes a su propósito, siguiendo la metodología propuesta en [9]. En forma paralela se incorporarán a la herramienta nuevos componentes a los actualmente soportados (tales como *ScrollPane*) y se analizará la posibilidad de: *(1)* agregar el soporte al modelo de eventos que permita, siguiendo la filosofía de diseño ya aplicada, capturar y procesar los eventos que pudieran generar los diferentes componentes de una interfaz gráfica, y *(2)* agregar el soporte necesario para crear interfaces gráficas con la librería gráfica *Swing*.

Agradecimientos

Al Dr. Luis Olsina, titular de la cátedra Programación Orientada a Objetos de la Facultad de Ingeniería de la Universidad Nacional de La Pampa, por la posibilidad de aplicar los avances de este trabajo.

Referencias

[1] Jocelyn Armarego and Geoffrey G. Roy, Teaching programming with objects, Proceedings of the 21st ASCILITE Conference, 2004 [http://www.ascilite.org.au/conferences/perth04/procs/contents.html].

[2] Kent Beck and Ward Cunningham, A Laboratory For Teaching Object-Oriented Thinking, Proceedings of OOPSLA'89 Conference, October 1-6, New Orleans, Louisiana, 1989.

[3] Ben-ari, M and Ragonis, N. and Ben-bassat Levy, R., A Vision of Visualization in Teaching Object-Oriented Programming, In Proceeding of 2nd Program Visualization Workshop, pp. 83-89, 2002.

[4] Robert Eckstein, Java SE Application Design With MVC, Oracle Technology Network, Marzo 2007 [http://www.oracle.com/technetwork/articles/javase/index-142890.html].

[5] Leoncio Jiménez C., Pascale Zaraté, Elizabeth Vidal, Analogías Gráficas como Método de Aprendizaje en un Curso de Programación Orientada a Objetos, Revista Ingeniería Informática, edición 13, noviembre de 2006.

[6] Knudsen, J. Lindskov and Madsen, O. Lehrmann, Teaching object-oriented programming is more then teaching object-oriented programming languages, Proceedings of ECOOP '88 (European Conference on Object-Oriented Programming), Oslo, Norway, pp. 21-40, Springer-Verlag. ISBN 0-387-50053-7, 1988.

[7] Kölling, M., Quig, B., Patterson, A. and Rosenberg, J., The BlueJ system and its pedagogy, Journal of Computer Science Education, Special issue on Learning and Teaching Object Technology, Vol 13, No 4, Dec 2003.

[8] Kölling, M. and Rosenberg, J., Guidelines for Teaching Object Orientation with Java, Proceedings of the 6th conference on Information Technology in Computer Science Education (ITiCSE 2001), Canterbury, 2001.

[9] Philip Lew, Luis Olsina, Pablo Becker, Li Zhang: An integrated strategy to systematically understand and manage quality in use for web applications. Requir. Eng. 17(4): 299-330 (2012).

[10] Kleanthis C. Thramboulidis, A sequence of assignments to teach object-oriented programming: a constructivism design-first approach, Institute of Mathematics and Informatics - Vilnius, Informatics in Education, 2003, vol. 2, No. 1, 103–122.

[11] Rubén Peredo Valderrama, Francisco F. Córdova Quiroz, Óscar Camacho Nieto, Una metodología para la enseñanza en la programación orientada a objetos, Temas de Ciencia y Tecnología 1 (Enero-Abril 1997) [http://www.utm.mx/temas/temas1-5.html].

Datos de Contacto:

Hernán Molina. Facultad de Ingeniería, Universidad Nacional de La Pampa. Calle 110 N° 390 - (6360) GENERAL PICO - La Pampa. hmolina@ing.unlpam.edu.ar.