GRAPHVIRUS[1]:

Una Herramienta para el Tratamiento de fallas en Redes.

Laurent Freyss, Oscar Ordaz, Domingo Quiroz, Jesús Yépez

Centro de Ingeniería de Software y Sistemas (ISYS), Ap.47567, Los Chaguaramos 1041-A,Caracas,Venezuela.

e-mails: *freyss@anubis.ciens.ucv.ve, oordaz@conicit.ve, dquiroz@anubis.ciens.ucv.ve, jyepez@anubis.ciens.ucv.ve*

Oscar Meza

Universidad Simón Bolívar Ap. 89000, Baruta, Caracas, Venezuela.

e-mail: meza@usb.ve

RESUMEN

Una red de interconexión puede confrontar fallas, es decir, poseer pequeñas estructuras que hacen que la red deje de poseer una determinada funcionalidad. A estas pequeñas estructuras las llamamos virus. GRAPHVIRUS es una herramienta computacional que permite editar virus gráficamente, verificar si una estructura dada es un virus, detectar la presencia de un virus en la red y generar redes que posean ciertos virus dados a priori. En este artículo se presenta la herramienta, y se describe sus funcionalidades. GRAPHVIRUS ha sido diseñado e implementado mediante una metodología orientada a objetos utilizando como base el modelo multiagentes PAC (Presentación, Abstracción, Control). Se presenta brevemente la metodología utilizada en el desarrollo de la herramienta.

Palabras Clave: Digrafos, Virus en grafos, Ambientes gráficos interactivos, Fallas en Redes, diseño orientado a objetos, modelo PAC, Ingeniería de Software.

1. INTRODUCCIÓN

En la actualidad existen numerosas herramientas computacionales para resolver problemas específicos en el área de la Teoría de Grafos. Los editores de grafos CABRI [2], UNICORN [5], AMDI ([8], [9]), GReAt y AMDI_OO ([10], [15], [17]) y los generadores de conjeturas GRAFFITI [6], AGORA [16]. GRAPHVIRUS forma parte del ambiente GReAt (Graph Researcher Assistant), el cual agrupa un conjunto de herramientas para asistir en sus tareas a un investigador en Teoría de Digrafos. Este conjunto de herramientas comprenden un editor de Digrafos, un editor de familias de grafos, cálculo de propiedades y generación aleatoria de grafos, un generador de conjeturas

[1] Proyecto de Investigación financiado por el CDCH de la Universidad Central de Venezuela, No. 03-12-3726-96.

(AGORA) y un editor de virus (GRAPHVIRUS). Estas herramientas han sido desarrolladas utilizando métodos de orientación a objetos basados en el modelo multiagente PAC ([4], [12], [13]).

El objetivo principal de este artículo es presentar la herramienta GRAPHVIRUS y sus principales funcionalidades. Aparte de la introducción y las conclusiones, el artículo comprende 3 secciones: la sección 2 presenta la justificación para el desarrollo de GRAPHVIRUS y establece la terminología a utilizar, la sección 3 presenta la descripción de la herramienta GRAPHVIRUS y sus funcionalidades principales, en la sección 4 se presenta una breve descripción de la metodología de desarrollo utilizada.

2. JUSTIFICACIÓN Y TERMINOLOGÍA

Una Red de Interconexión consiste de unidades de hardware y software las cuales están interconectadas para facilitar procesos de computación y comunicación. Estas entidades pueden ser procesadores, procesos o módulos de memoria. Una red se representa por un grafo dirigido. Los vértices del grafo representan los nodos o procesadores de la red, los lados corresponden a las líneas de interconexión. La funcionalidad de una red la definen los valores de sus invariantes gráficas tales como: número de nodos, conectividad, diámetro, índice y diámetro de transmisión, hamiltonicidad, etc. Por ejemplo, el diámetro (i.e., la máxima distancia entre dos vértices del grafo) expresa el máximo retardo cuando en la red se transmite un mensaje entre dos nodos. Dada una red de interconexión y la ruta que deben seguir los mensajes entre cada par de nodos, el índice y el diámetro de transmisión miden respectivamente, la vulnerabilidad de los nodos y el tiempo promedio para transmitir un mensaje entre cada par de nodos. Una falla o virus en una red es una estructura local que impide que la red posea una determinada funcionalidad. Por ejemplo, respecto a la funcionalidad de que entre cada par de nodos haya un camino en ambos sentidos, una falla es un nodo que solo envía mensajes en un solo sentido. El concepto de virus ha sido estudiado por los autores ([1],[3],[9],[14],[19],[20]) y constituye, como veremos, una herramienta adecuada para la detección y la verificación de esta fallas en redes.

En lo que sigue se usarán indistintamente los términos digrafo y red .

Sea N el conjunto de enteros no negativos. Sea P una propiedad definida sobre todos los digrafos (por ejemplo, 'posee circuito hamiltoniano').

Un virus para digrafos es una 5-upla $V = (H,T^+,T^-,f^+,f^-)$ *donde* $H=(V(H),E(H))$ *es un grafo dirigido,* T^+,T^- *son subconjuntos de* $V(H)$ *y* f^+,f^- *son funciones de* T^+,T^- *en N respectivamente.*

Un virus para digrafos (H,T^+,T^-,f^+,f^-) ***está presente*** *en un digrafo D si D posee un subgrafo inducido propio (es decir, un subgrafo cuyo número de vértices sea menor estricto que el de D y cuyo conjunto de arcos sean aquellos de D que conectan dos vértices del subgrafo) H' isomorfo a H (por conveniencia identificamos H' con H), tal que las igualdades siguientes se satisfacen:*

$d^+_D(x) = f^+(x) + d^+_H(x)$ y $d^-_D(x) = f^-(x) + d^-_H(x)$ *para cada vértice x del grafo.*

Un virus para digrafos V es un virus para la propiedad P si todo digrafo donde V está presente no posee la propiedad P.

El concepto de virus puede ser de gran utilidad en la generación de grafos aleatorios que cumplan ciertas propiedades y no cumplan otras (por ejemplo, generar grafos bipartitos no hamiltonianos). El editor de grafos de GReAt posee una funcionalidad que permite generar grafos aleatorios utilizando el concepto de virus. De igual forma, los virus pueden ser de utilidad para dar una solución exacta o aproximada a problemas de decisión en grafos. Por otro lado, el problema de determinar si un grafo G posee una determinada propiedad puede ser un problema NP-completo (ej., Hamiltonicidad). Si un virus V para una propiedad P es conocido, y si la ausencia o presencia de V en un grafo puede ser determinada en un tiempo computacionalmente razonable, entonces una solución aproximada al problema de decisión puede ser dada chequeando la ausencia o presencia de V en G. . En [3] se presenta la siguiente 'metaconjetura': Para muchas propiedades importantes de grafos existen virus de orden pequeño en *casi todos* los grafos que no posean la propiedad. De ser esta metaconjetura cierta, la presencia o ausencia de un virus de orden k en un grafo de orden n puede ser detectado por un algoritmo $O(n^{k+1})$. Si k es un número pequeño (digamos, menor o igual a 4) entonces podemos dar una respuesta *casi segura ('casi'* debe entenderse en el sentido probabilístico usual, es decir, una fracción que tiende a 1 cuando el orden del grafo tiende a infinito), con un algoritmo de bajo costo computacional, al problema de decidir si un grafo G posee la propiedad P. En [1] y [3] se presentan dos ejemplos a favor de esta metaconjetura.

Una observación interesante en relación a la utilidad del concepto de virus es la siguiente: sabemos que el problema de decidir si un digrafo dado posee un circuito hamiltoniano es un problema NP-Completo. Para el caso de la copropiedad 'no hamiltoniano' no se conoce un algoritmo polinomial no determinístico que nos diga si un grafo dirigido dado no es hamiltoniano y esto se debe precisamente a la falta de una 'firma' (la firma para una grafo hamiltoniano sería un circuito hamiltoniano en ese grafo) que garantice que el grafo es no hamiltoniano (ver [18]). Una solución parcial a este problema (parcial porque es posible encontrar Digrafos no hamiltonianos sin virus) es proporcionar un virus, cuya presencia en el grafo puede también ser verificada en tiempo polinomial.

3. DESCRIPCIÓN DEL AMBIENTE GRAPHVIRUS

GRAPHVIRUS es un ambiente gráfico interactivo para dibujar y manipular virus para Digrafos. Como se muestra en la figura 1, el usuario puede editar y manipular simultáneamente varios virus (cada uno en una ventana diferente). El portapapeles permite intercambiar información entre diferentes ventanas. Las opciones del menú principal son: *file, edit, view, properties, tools.* Las funcionalidades se pueden acceder a través de los menúes de barras o mediante botones que ofrecen algunas operaciones específicas que son aplicadas al virus en la ventana. Estas operaciones permiten la creación de nuevos vértices, la creación de nuevos arcos, etc.

La opción *file* permite recuperar y almacenar virus, generar aleatoriamente un virus $V = (H, T^+, T^-, f^+, f^-)$, donde H posea determinadas propiedades, imprimir el virus en la ventana activa. La operación de recuperación permite

cargar en una ventana un virus o un grafo ya almacenado; en el caso de que se cargue un grafo este se inicializa automáticamente con $T^+=V(H)$, $T^-=V(H)$, $f^+ = 0$ y $f^- = 0$, brindando así, como veremos, la posibilidad al usuario de construir un virus que esté presente en un grafo dado de antemano. La opción *edit* permite crear y modificar un virus en la ventana activa mediante operaciones de edición para agregar vértices, arcos, definir T^+,T^-,f^+ y f^-, operaciones sobre un conjunto de vértices seleccionados de H, etc. Todas las operaciones de edición referentes a la modificación de H son las mismas operaciones del editor de Digrafos de GReAt. Existen dos modos de edición de virus, los cuales serán discutidos en la sección 3.1. La opción *view* permite redibujar el grafo H de acuerdo a distintas heurísticas de dibujo. La opción *properties* permite hacer cálculo de propiedades de H, como por ejemplo orden del virus, número de arcos, grados máximo y mínimo de entrada y salida, conectividad, diámetro, etc. La opción *tools* es la que fundamentalmente diferencia a GRAPHVIRUS de un simple editor de grafos y será descrita en la sección 3.2.

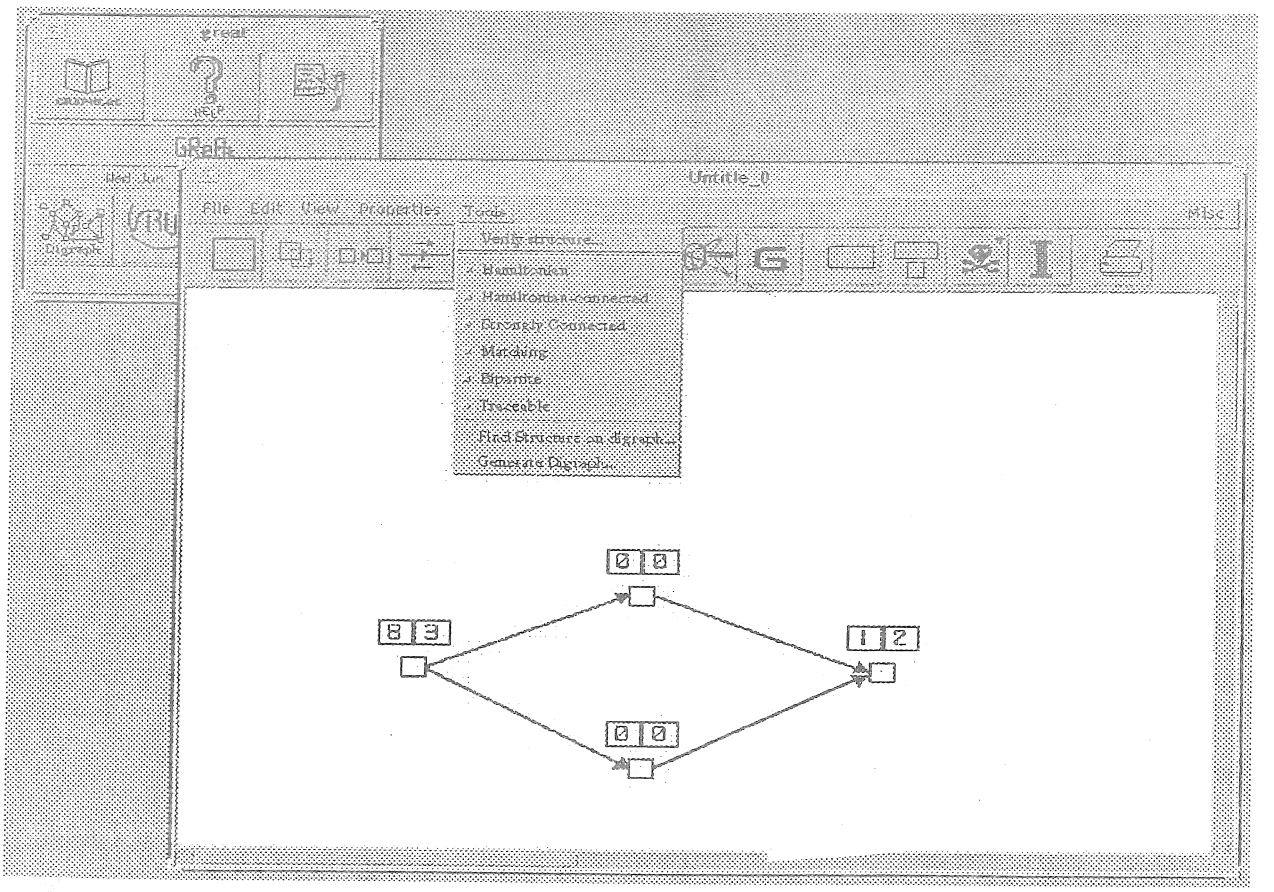

Figura 1. El ambiente GRAPHVIRUS.

3.1 Modos de edición de virus.

El usuario puede escoger entre dos modos de edición de virus:

1) La eliminación de un vértice afecta los valores de f^+ y/o f^- de los vértices adyacentes al vértice eliminado.

2) La eliminación de un vértice no afecta los valores de f^+ y/o f^- de los vértices adyacentes al vértice eliminado.

Cuando se edita un virus, es responsabilidad del usuario definir completamente f^+ y f^- para cada vértice creado. Al agregar un vértice o un arco a H no se alteran los valores de T^+,T^-,f^+ y f^- de los demás vértices de H, y es responsabilidad del usuario definir f^+ y f^- para el vértice que está creando. Cada vértice posee dos casillas adyacentes a él. El valor de la primera (segunda) casilla corresponde al valor de f^+ (f^-) para ese vértice; si en la casilla no se coloca ningún valor esto se entiende como que el vértice no pertenece a T^+ (T^-). En relación a la eliminación de vértices, sea v un vértice a ser eliminado. En el primer modo de edición, al eliminar el vértice v, si el arco (x,v) (el arco (v,x)) existe y $f^+(x)$ ($f^-(x)$) está definido, su valor aumenta en una unidad. En el segundo modo de edición, la eliminación de un vértice no afecta los valores de f^+ ni de f^- de ningún vértice de H.

El primer modo de edición permite construir un virus que esté presente en un grafo: al cargar un grafo en una ventana, f^+ y f^- serán iguales a cero para todo vértice, y la eliminación de vértices (en este modo) garantiza que la estructura que se vaya obteniendo sea un virus presente en el grafo originalmente cargado. El usuario puede cambiar de modo en cualquier momento y en ambos modos tiene plena libertad de modificar los valores de f^+ y f^- de cualquier vértice.

3.2 Descripción de las funcionalidades de la opción *'tools'* del menú principal.

La opción 'tools' ofrece las siguientes funcionalidades en su menú vertical: generar aleatoriamente un grafo que posea el virus en la ventana de edición, verificar si el virus en la ventana de edición es un virus para una determinada propiedad de grafos y detectar si el virus en la ventana de edición está presente en un grafo dado. A continuación describimos cada funcionalidad.

Generación aleatoria de un grafo de orden n que contenga el virus en la ventana:

El proceso consiste en generar un grafo aleatorio cualquiera de orden n menos el orden del virus, luego agregar arcos que conecten vértices de ese grafo generado con los vértices del virus de una manera compatible con f^+ y f^-. El grafo así construido será mostrado en una ventana del editor de grafos de GReAt.

Verificación del virus en la ventana, para una determinada propiedad:

El usuario escoge del menú vertical de 'tools' una propiedad de grafos y enseguida el sistema procede a verificar si el virus en la ventana es un virus para la propiedad seleccionada. Las propiedades que actualmente contempla la operación de verificación son: 'es conexo', 'es fuertemente conexo', 'posee apareamiento perfecto', 'es bipartito', 'posee circuito hamiltoniano'. Para todas estas propiedades, salvo 'posee circuito hamiltoniano' existen algoritmos

polinomiales de verificación (ver [3],[19]). Para la propiedad 'posee circuito hamiltoniano' no se conoce un algoritmo exacto de verificación de orden polinomial. En [20] se presenta un algoritmo genético para la verificación de esta propiedad.

Detección de un virus en un grafo dado: A continuación describimos el algoritmo que permite decidir si el virus en la ventana de edición está presente en un grafo dirigido, que el usuario escoge de una lista de Digrafos ya almacenados.

Buscamos una inyección I de V(H) en V(D) tal que la imagen de H por I sea isomorfa a H y tal que las condiciones de conexión definidas f^+ y f^- se satisfagan. El algoritmo genético está definido por su codificación (la manera de representar las soluciones) y por su función de evaluación (la manera de elegir las mejores soluciones)

Codificación:

La inyección buscada está representada por un cromosoma con n =|V(H)| genes. Cada gen representa un vértice de V(D) imagen por I de cada vértice de V(H).

Sean $X_1, X_2, \ldots, X_n$ los vértices de H, entonces un cromosoma $C = Y_1\ Y_2\ \ldots\ Y_n$ representa la inyección naturalmente definida por $I(X_1) = Y_1, \ldots, I(X_i) = Y_i, \ldots, I(X_n) = Y_n$. Por ejemplo sean 1, 2, 3, 4 y 5 los vértices del virus, el cromosoma C = 3 5 6 7 2 representa la inyección $1\rightarrow3$, $2\rightarrow5$, $3\rightarrow6$, $4\rightarrow7$, $5\rightarrow2$.

Función de evaluación:

Esta función mide si la inyección es isomorfa y si ella respeta las condiciones de conexión.

$$F(C) = \frac{1}{2}\left(\frac{\left|(xy) \in E(H) tal\ que (I(x)I(y)) \in E(D)\right|}{|E(H)|} + \frac{\left|x \in V(H) tal\ que\ d_D^+(x) = f^+(x) + d_H^+(x)\ y\ d_D^-(x) = f^-(x) + d_H^-(x)\right|}{|V(H)|}\right)$$

El primer término evalúa la proporción de arcos del virus cuya imagen por la inyección están presente en el grafo dirigido, el segundo término evalúa la proporción de vértices del virus cuya imagen por la inyección respetan las condiciones definidas por f^+ y f^-. El algoritmo consiste en hacer evolucionar la población de soluciones hasta conseguir un individuo *perfecto* C que tenga la evaluación máxima de F. (F(C) = 1).

4. EL ENFOQUE ORIENTADO A OBJETOS Y EL DESARROLLO DE GRAPHVIRUS

En esta sección presentamos una breve descripción del método Orientado a Objetos (OO) adoptado para desarrollar el sistema GRAPHVIRUS. Para mayores detalles referirse a [4], [10], [11], [12] y [13].

PAC: Un modelo multiagente

El modelo PAC (Presentación, Abstracción y Control) [4] es un modelo multiagente que permite diseñar la arquitectura de un sistema interactivo. Los agentes de interfaz (representados gráficamente mediante óvalos) se organizan de acuerdo a tres perspectivas básicas: - La *Presentación* (representada como P en el óvalo), la cual define la imagen del sistema, reflejando su comportamiento respecto a la entrada/salida del usuario. - La *Abstracción* (representada como A en el óvalo), la cual define de manera abstracta (independientemente de la presentación de los conceptos en la pantalla del computador) los conceptos y funcionalidades del sistema. - El *Control* mantiene la coherencia y las comunicaciones entre las perspectivas *Presentación* y *Abstracción*, debido a que no se permite la comunicación directa entre estas últimas. Mas aún, las comunicaciones (representadas por líneas que conectan los óvalos) entre los agentes PAC solo se realiza a través de sus respectivos controles.

El diagrama PAC de la aplicación posee una estructura de árbol. Cada nivel del árbol corresponde a un nivel semántico de la aplicación. Por un lado, el nivel más alto, llamado nivel de la aplicación, contiene un agente complejo que representa las características semánticas de toda la aplicación (modelo de datos). Por otro lado, los niveles más bajos (las hojas) contienen características gráficas elementales de la aplicación (objetos gráficos). Los niveles intermedios se agrupan en lo que se denomina nivel de la interfaz. La arquitectura PAC de GReAt y GRAPHVIRUS es mostrada en la figura 2. Al nivel del agente "Virus_window" están las demás herramientas que conforman GReAt (el editor de digrafos, editor de familias, etc.)

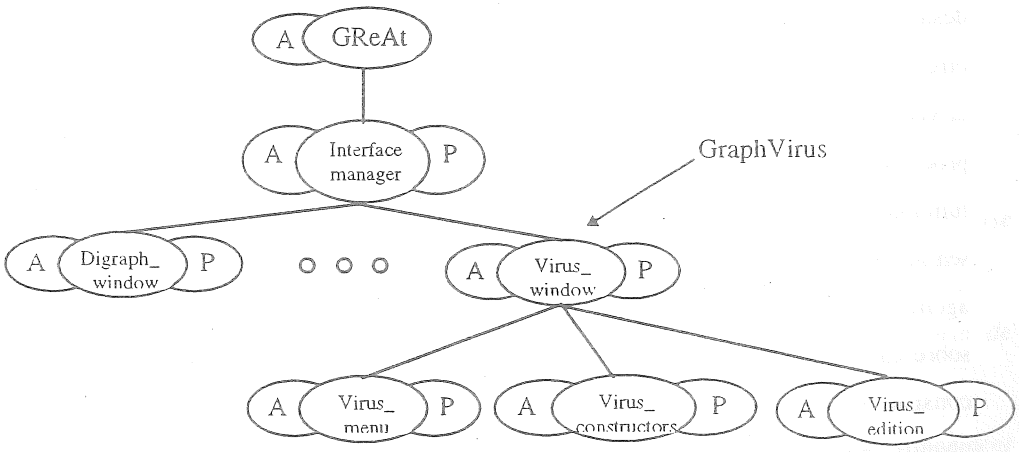

Figura 2. Arquitectura de GRAPHVIRUS según el modelo PAC

La Metodología Orientada a Objetos utilizada para desarrollar GRAPHVIRUS

El modelo PAC no impone ninguna metodología particular de diseño, sin embargo, los agentes PAC pueden de manera natural ser implementados utilizando técnicas orientadas a objetos. En [10], [11], [12] y [13] se propone un método OO para desarrollar aplicaciones interactivas cuya arquitectura está basada en el modelo PAC. Los pasos, descritos muy brevemente, son los siguientes:

i. Formulación de los requerimientos de la aplicación.

ii. Estudio de las funcionalidades del sistema con respecto al GUI, de acuerdo a los requerimientos formulados en el paso i.

iii. Identificación de los objetos gráficos y de los agentes de interfaz de acuerdo a los resultados anteriores. Se construye un diagrama general de agentes. En paralelo, se identifican las clases que implementan las perspectivas Abstracción y Presentación. En este paso también se construye un diagrama de clases que representa el modelo objeto del sistema.

iv. Cada agente de interfaz constituye un ´framework´ o conjunto de clases relacionadas para resolver un problema de diseño particular, en el sentido de [24]. El framework es instanciado, especializando sus clases abstractas, de acuerdo a las características del agente que debe ser implementado en una aplicación particular.

v. Implementación de frameworks que corresponden a los agentes identificados en los pasos iii y iv. Reutilización del 'toolkit' de clases disponibles para implementar los objetos gráficos (hojas en el diagrama PAC).

El enfoque use-case de Jacobson [7] se ha utilizado como un front-end para este método [22] (pasos i y ii) . La arquitectura del sistema es diseñada en el (paso iii) de acuerdo a la filosofía del modelo PAC (ver figura 2). Las técnicas de diseño de Booch [23] pueden ser utilizadas para modelar el nivel Abstracción o modelo de datos de la aplicación, el cual representa la componente del dominio del problema en el sentido de Booch.

Respecto a al paso v, GRAPHVIRUS está siendo desarrollado en C++, sobre una plataforma Sun/Unix. Para el desarrollo de la interfaz gráfica se ha utilizado como ´toolkit´ OSF/MOTIF, versión 1.1. Ahora bien, este toolkit ofrece elementos gráficos que poseen atributos y un cierto comportamiento, dependiendo de su tipo. Un widget en la versión de MOTIF que se utilizó, no es un objeto, en el sentido usual, ya que no posee ni atributos ni objetos o procesos encapsulados. Además, la presentación y el manejo de los eventos de los widgets es realizada a través de funciones estáticas, que pueden ser invocados desde cualquier parte del programa, donde la estructura del widget sea accesible. Para poder manipular convenientemente los widgets a partir de las clases C++ que implementan los agentes PAC de los niveles intermedios de la jerarquía impuesta por el modelo PAC, se ha generado una envoltura sobre cada tipo de widget, de acuerdo al enfoque de reutilización que trata el denominado ´legacy problem´. Este consiste, en convertir en clases, programas o funciones que no han sido desarrollados en lenguajes orientados a objetos. En nuestro caso, se trata del lenguaje C, en el cual están implementados los widgets de MOTIF. La envoltura o ´wrapper´ generada, encapsula atributos y comportamiento de los widgets. Estas nuevas clases C++ constituyen un nivel superior a los widgets de MOTIF y pueden ser combinadas para desarrollar objetos de interfaz más complejos que puedan ser reutilizados en general por otras aplicaciones escritas en C++.

5. CONCLUSIONES

Se han presentado las funcionalidades principales de la herramienta GRAPHVIRUS. Para el desarrollo sistemático de la aplicación se utilizó una metodología, definida en el Centro ISYS de la Facultad de Ciencias de la

Universidad Central de Venezuela, basada en modelos multiagentes y el paradigma orientado a objetos para construir sistemas gráficos interactivos. Una de las características relevantes de la metodología es construir una arquitectura que permite separar la interfaz usuario de los aspectos propios del dominio del problema, facilitando la portabilidad y las modificaciones eventuales que pueda tener el sistema.

6. AGRADECIMIENTOS

Los autores desean expresar su agradecimiento a Francis Losavio por la supervisión de las versiones preliminares de este artículo.

7. REFERENCIAS

[1] Brito, M.R. *A simple, almost surely correct algorithm for deciding if a graph has a perfect matching*. Discrete Applied Mathematics 63, (1995) 181-185.

[2] O. Boudon, C. Benzaken, J. Bordier, P. Eades, I. Havel, J.M. Laborde, G. Lejeune, M. Mollard. *CABRI: Cahier de Brouillon Informatique*. Reference Guide. Version 3.0, LSD-IMAG, Grenoble, France, Avril 1990.

[3] Brito M., Fernández W., Meza O., Ordaz O. *Viruses in graphs and digraphs.* Vishua International Journal of Graph Theory. Volume 2, number 1, pp.33-55. 1993.

[4] Joelle Coutaz. *Interfaces Homme-Ordinateur*. Dunod 1990.

[5] J.M. Fourneau. *Unicorn*. Rapport Interne Isem, Vol 51, 1987.

[6] S. Fajtlowicz. *On Conjectures of Graffiti, II*. Congressus Numerantum 60 187-197. 1987.

[7] Jacobson I., Christerson M., Jonsson P., Overgaard G. *Object-Oriented Software Engineering, a Use-case Driven approach*. Addison-Wesly. 1992.

[8] F. Losavio, O. Meza, O. Ordaz. *Un AMbiente interactivo de trabajo para un investigador en teoría de Digrafos AMDI.* Acta Científica Venezolana, 1990.

[9] F. Losavio, L.E. Marquez, O. Meza, O. Ordaz. *La génération aléatoire de digraphes dans l'environnement AMDI.* T.S.I., Vol. 10, No.6, 1991.

[10] F. Losavio, A. Matteo, O. Meza, O.Ordaz, W. Gontier. *Object-Oriented Approach and the PAC Model: Design of the GReAt environment.* Proceedings of the XX Latinoamerican Conference on Computing, PANEL'94. Mexico city, September 1994, Editorial LIMUSA, ISBN 968-18-5119-6. 1994

[11] F. Losavio, A. Matteo, O. Ordaz, O. Meza, W. Gontier. *An Implementation of the PAC architecture using Object-Oriented techniques.* Proceedings of IFIP'94. Hamburg, Germany, August 29 - September 2, 1994.

[12] F. Losavio, C. Gálvez. *The platform influence on Object-Oriented development based on the Multiagent Model.* Proceedings of the XXI Latinoamerican Conference on Computing PANEL'95, Canela, Brazil, pp.1175-1186. 1995.

[13] F. Losavio. *An Object-Oriented Methodology for User Interface Design based on the Multiagent Model.* Proceedings of the International Cnference on Information Systems Analysis and Synthesis ISAS'95, Baden-Baden, Germany, Ag. 1995, pp. 88-92.

[14] Oscar Meza, Oscar Ordaz. *Virus en grafos no hamiltonianos.* Proceedings of the XX Latinoamerican Conference on Computing, PANEL'94. Mexico city, September 1994, Editorial LIMUSA, ISBN 968-18-5119-6.

[15] Oscar Meza, Francis Losavio, Oscar Ordaz. *The Family Description Language and the Instantiation Process of GReAt.* Proceedings of the XXI Latinoamerican Conference on Computing, PANEL'95. Canela, Brazil. September 1995.

[16] García L., Losavio F., Ordaz O., Lastre E. *A reasoning System for Digraph Conjecture Solving.* Proceedings of The International Conference on Information Systems Analysis and Synthesis. Orlando, USA. July 22-26, 1996.

[17] Oscar Meza, Francis Losavio, Oscar Ordaz. *AMDI_OO: Using the Object-Oriented Approach for the Implementation of a Directed Graph Editor.* Proceedings of The International Conference on Information Systems Analisys and Synthesis. July 22-26, Orlando, USA. 1996.

[18] M.R. Garey and D.S. Johnson. *Computers and Intractability. A guide to the Theory of NP-Completeness.* W.H. Freeman and Company, N.Y. 1979.

[19] M.R. Brito, C. Delorme, L.E. Márquez, y O. Ordaz. *Virus para propiedades de grafos.* Anales de la XXII Conferencia Latinoamericana de Informática, PANEL'96. Bogotá. Junio, 1996.

[20] Laurent Freyss, Oscar Ordaz, Domingo Quiroz. *A Method for Identifying Hamiltonian viruses.* Enviado a la XXIII Conferencia Latinoamericana de Informática, PANEL'97.

[21] Whitley D., Starkweather T., Fuquay D'A. *Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator.* Proceedings of The Third International Conference on Genetic Algorithms. Morgan Kaufmann Publishers, San Mateo, C.A. 1989.

[22] Losavio F., Matteo A. "Use-case and multiagent Models for Object-oriented Design of user-Interface" Journal of Object-Oriented Programming, vol 10, No 5, mayo 1997 pp (30-40).

[23] Booch G. *Object-Oriented Analysis and Design with Applications.* Benjamin/Cummings Company. 1994.

[24] Pree W. Desing Patterns for Object-Oriented Development, Addison Wesley, 1994.