Comprensión de Algoritmos de Ruteo

Mario M. Berón
Universidad Nacional de San Luis, Departamento de Informática
San Luis, Argentina, 5700
e-mail: mberon@unsl.edu.ar

Pedro R. Henriques
Universidade do Minho, Departamento de Informática
Braga, Portugal
e-mail: prh@di.uminho.pt

Maria J. Varanda Pereira
Instituto Politécnico de Bragança, Departamento de Informática
Bragança, Portugal
e-mail: mjoao@ipb.pt

Roberto Uzal
Universidad Nacional de San Luis, Departamento de Informática
San Luis, Argentina, 5700
e-mail: ruzal@sinectis.com

Resumen

La comprensión de programas es un área de la Ingeniería del Software que se encarga del estudio y construcción de modelos y herramientas con el objetivo de facilitar el mantenimiento, la modificación y el estudio de aplicaciones de informática.
En este artículo presentamos los trabajos realizados, en el contexto de la comprensión de programas, destinados a analizar la posibilidad de aplicar las estrategias de comprensión de programas a los algoritmos de Ruteo Geométrico. Las tareas fueron llevadas a cabo siguiendo cuatro líneas de trabajo bien definidas. La primera consistió en el estudio de diferentes herramientas de comprensión de programas con el objeto de establecer un ranking y analizar las estrategias de comprensión utilizadas. La segunda se centro en la construcción de un Evaluador de Algoritmos de Ruteo con propósitos educativos y de investigación. La tercera analiza la posibilidad de adaptar estrategias de comprensión al evaluador de algoritmos para aumentar su capacidad explicativa. Finalmente, la cuarta línea se encarga de generalizar los resultados obtenidos con el evaluador con el objeto de utilizar las nuevas estrategias de comprensión definidas en sistemas en general.

Palabras Claves: Comprensión de Programas, Estrategias de Comprensión, Geometría Computacional, Herramientas de comprensión de programas.

1. INTRODUCCIÓN

La comprensión de programas consiste en la habilidad de entender varias unidades o módulos de códigos, escritos en un lenguaje de programación de alto nivel, que integran una aplicación informática. La comprensión de programas está profundamente ligada a la Ingeniería del Software y se necesita para la reutilización, inspección, manutención, reingeniería, migración y extensión de los sistemas de software [5,9] existentes.
La comprensión de un programa puede tener diferentes significados o perspectivas. Podemos estar interesados en la forma en que la computadora ejecuta las instrucciones con el objetivo de entender el flujo de control; o podemos enfocar nuestro interés en los efectos que producirá la ejecución del programa en el dispositivo (físico o virtual) que es manipulado por el mismo. En cualquiera de los dos casos pensamos que una herramienta de inspección visual es fundamental para efectuar este tipo de tareas.
En este contexto el proyecto PCVIA (Program Comprehensión by Visual Inspection and Animation) tiene como propósito: i) estudiar los modelos cognitivos; ii) analizar continuamente el estado del arte de sistemas de comprensión, iii) desarrollar herramientas de comprensión de programas basadas en vistas complementarias, iv) proponer nuevas estrategias de comprensión y arquitecturas de sistemas.

En términos generales, las herramientas de comprensión de programas utilizan estrategias de comprensión estáticas y dinámicas. Las primeras utilizan técnicas de exploración de código para construir una representación visual del sistema que se desea estudiar. Las segundas se basan en el análisis del comportamiento del programa en tiempo de ejecución. Ambas técnicas ayudan al programador a entender programas. Sin embargo, como se podrá observar en este artículo, hemos detectado que la mayoría de las herramientas no utiliza a la animación como estrategia de comprensión de programas. Pensamos que este hecho se debe al problema de definir representaciones visuales de animación y a la dificultad de su implementación. Teniendo esto presente, decidimos analizar la posibilidad de utilizar este tipo de técnica de comprensión en un domino de aplicación que se caracteriza por tener una representación visual bien definida como es el caso del Ruteo Geométrico. Deseamos realizar estudios que nos develen las dificultades implicadas en la construcción de herramientas de comprensión que utilicen técnicas estáticas y dinámicas (incluyendo a las de animación). Para llevar a cabo esta tarea realizamos una evaluación de herramientas de comprensión de programas para analizar las técnicas usadas y establecer un ranking de las mismas. Tomamos a esta evaluación como base para definir los requisitos esenciales que deberían cumplir las herramientas de comprensión dinámicas que usan animación como estrategia de comprensión. Luego desarrollamos Un *Evaluador de Algoritmos de Ruteo* cuyo objetivo es facilitar el aprendizaje y evaluar el desempeño de estrategias de ruteo en general. Como paso siguiente, analizamos la factibilidad de aumentar la capacidad explicativa, a través del uso de animación, del evaluador de algoritmos de ruteo. Finalmente, planteamos una generalización de las técnicas usadas en el evaluador a sistemas en general.
Este artículo está organizado como sigue. En la sección 2 presentamos las herramientas de comprensión estudiadas y su evaluación correspondiente. La sección 3 describe al evaluador de algoritmos de ruteo, su arquitectura, funcionalidades. La sección 4 justifica la necesidad de incorporar estrategias de comprensión al evaluador, define criterios que las herramientas de comprensión de algoritmos de ruteo deberían poseer y generaliza los resultados. Finalmente, presentamos las conclusiones de este artículo.

2. HERRAMIENTAS DE COMPRENSIÓN DE PROGRAMAS

En términos generales podemos decir que el software utilizado en el contexto de la comprensión de programas puede ser clasificado en dos grandes categorías: desarrollo en general y de propósito específico. La primer clase hace referencia a aquellas herramientas que son de desarrollo de sistemas en general, pero que poseen un conjunto de funcionalidades que facilitan el desarrollo de herramientas de comprensión como por ejemplo: Eclipse SDK, Visual Estudio .Net. La segunda hace referencia a herramientas destinadas específicamente al proceso de comprensión de programas como por ejemplo: SHriMP, Creole, Codesurfer, Imagix 4D y Jeliot 3. Este artículo está orientado al estudio de la segunda clase de software porque consideramos que, al explorar sus características, nos permitirá desarrollar herramientas de comprensión más avanzadas y completas.

2.1. Herramientas Estudiadas

Según nuestro criterio, pensamos que las herramientas abajo descriptos son muy significativos para el estudio de esta temática debido a que poseen un conjunto bastante diverso de técnicas de comprensión. En las siguientes subsecciones damos una breve descripción de cada una de ellos.

2.1.1. SHriMP

SHriMP es una aplicación diseñada para visualizar y explorar software desarrollado en el lenguaje java. SHriMP provee diferentes visiones [16] del software por ejemplo: vistas anidadas de módulos, jerarquía de clases e interfaces, grafos de llamadas, dependencias de paquetes vía llamadas de métodos y accesos a campos, visión del sistema usando treemaps (una representación de árboles compacta y útil para mostrar sistemas), entre otras tantas características. Todas esas técnicas de visualización ayudan al programador a comprender un sistema. Además, SHriMP tiene facilidades para la navegación de software y diferentes formas de personalizar el ambiente. Todas esas características hacen de SHriMP una buena herramienta de comprensión de programas. Una particularidad de SHriMP es la posibilidad de obtener vistas. Esta peculiaridad es importante porque brinda más alternativas de comprensión.

2.1.2. Creole

Creole es el término usado para describir un plug in para Eclipse SDK [11], un ambiente de desarrollo de aplicaciones cuyo lenguaje de programación es java, que integra SHriMP con Eclipse SDK. Con Creole, podemos analizar códigos java visualmente viendo su estructura y enlaces (llamadas, accesos, entre otras) entre las diferentes piezas. En otras palabras esta aplicación posee las mismas funcionalidades que SHriMP y además se le incorporan las del ambiente de programación Eclipse.

2.1.3. Codesurfer

Codesurfer es una aplicación diseñada para reducir defectos y verificar si el software cumple con todos los requerimientos. Esta aplicación permite analizar sistemas escritos en C y C++ y presenta las siguientes funcionalidades:

construye un grafo de llamadas de funciones, permite visualizar las llamadas a funciones realizadas a través de punteros, posibilita ver las variables y sus usos, entre otras tantas facilidades. Además se puede navegar fácil y eficientemente por el código fuente del sistema de estudio, posee herramientas de búsquedas eficaces y puede ser usado por distintos lenguajes de programación. Otra de las características importantes de esta aplicación es la posibilidad de invocarla con las opciones específicas del compilador que se está usando. La precisión de las operaciones que realiza hace de ésta una herramienta destacable y de alta calidad. Por otra parte, puede ser ejecutada en distintas plataformas (Unix, Linux, Windows, entre otras tantas).

2.1.4. Imagix 4D

Imagix 4D es una herramienta para la Ingeniería Inversa estática. Esta herramienta funciona en distintas plataformas (Unix, Linux, Windows, entre otras tantas) y posibilita el análisis de sistemas escritos en C y C++. En Imagix 4D existen ventanas que muestran lo siguiente: i) relaciones entre archivos, clases, funciones, variables, macros y tipos; ii) lugares donde los símbolos se encuentran ubicados dentro de la estructura y también posibilita ver el contenido de los archivos; iii) formas de navegar entre los distintos archivos del sistema de estudio; iv) resultados de búsquedas; v) navegación entre las distintas clases del sistema; vi) diagramas de flujos.

Las funciones provistas con Imagix 4D ayudan a diferentes tareas relacionadas con la comprensión de programas por ejemplo: i) permite una rápida navegación a través de: el flujo de control; herencias y relaciones entre las clases; ii) facilita el estudio del programa por medio de: componentes reusables y el proceso de construcción; iii) analiza el software de las siguientes maneras: focalizando en actividades de verificación; ubicando cuellos de botellas; iv) permite generar documentación: actualizada y de código indocumentado.

2.1.5 Jeliot 3

Jeliot 3 [13] es una aplicación de visualización de la dinámica [15] de un programa escrito en lenguaje java. Esta herramienta fue pensada para los principiantes. La característica más interesante, desde nuestro punto de vista, es la máquina de animación. Con este mecanismo, los alumnos pueden ver los objetos del programa, los mensajes llamados, la evaluación de expresiones, entre otras tantas aplicaciones, todas esas actividades son muy útiles para la comprensión de programas. Las animaciones muestran el comportamiento interno del sistema. Jeliot 3 es una herramienta simple si la comparamos con otras pero el atributo descrito previamente (las animaciones) son difíciles de desarrollar y generalmente, no se incluyen en aplicaciones más complejas porque requiere de aspectos más específicos del dominio del problema de estudio y de implementación.

2.2. Evaluación De Las Herramientas de Comprensión

La evaluación de las herramientas fue llevada a cabo usando los criterios definidos por Storey, M; Fracchia, F; Müller, A; en [17]. Esos autores declaran que para evaluar herramientas de comprensión de programas el programador debe observar si las aplicaciones pueden: i) *mejorar la comprensión de programas*, ii) *reducir el tiempo de cognición requerido por el programador*. La primera característica intenta clasificar a las herramientas de acuerdo al modelo cognitivo usado [12,14]. La segunda considera las facilidades y complejidad de las herramientas. En lo siguiente, presentamos los criterios de evaluación que utilizaremos para el estudio de estas herramientas.

Mejorar la Comprensión del Programa

- Aumento la comprensión bottom up
  - Indica las relaciones sintácticas y semánticas entre los distintos objetos de software (C1).
  - Reduce los efectos de planes deslocalizados (C2).
  - Provee mecanismos de abstracción (C3).
- Aumento de la Comprensión top down
  - Soporta comprensión dirigida por hipótesis orientada a objetivos. (C4).
  - Provee visiones de la arquitectura del programa en distintos niveles de abstracción (C5).
- Integra las aproximaciones top down y bottom up
  - Soporta la construcción de múltiples modelos mentales (dominio, situación, programa) (C6)
  - Referencias cruzadas de modelos mentales (C7).

Reduce el tiempo de cognición del programador

- Facilita la navegación
  - Provee navegación direccional (C8).
  - Soporta navegación arbitraria (C9).
  - Provee navegación entre diferentes modelos mentales (C10).
- Provee señales de orientación
  - Le indica al programador la sección de trabajo corriente (C11).
  - Muestra el paso que conduce a la sección de trabajo corriente (C12).
  - Indica las opciones que conducen a los nodos corrientes (C13).
- Reduce la desorientación

- Reduce el esfuerzo para ajustarse a la interfaz de usuario (C14).
- Provee estilos efectivos (C15).

2.2.1. SHriMP y Creole

SHriMP es una herramienta de exploración y visualización de sistemas escritos en lenguaje java. Considerando el modelo cognitivo podemos decir que SHriMP muestra la arquitectura del sistema como un grafo anidado. El programador puede explorar el programa en forma top down. Por otra parte es posible obtener diferentes visiones del programa. SHriMP muestra las relaciones semánticas y sintácticas claramente. Por lo tanto, con SHriMP es posible establecer relaciones entre los diferentes objetos del sistema (C1). Además podemos ver distintas visiones del sistema por ejemplo: grafos de llamadas, listas de código fuente, variables, relaciones entre variables, entre otras características. Cada visión se puede visualizar con colores elegidos por el usuario. SHriMP no tiene mecanismos para código fuente pertenecientes a planes deslocalizados. Este problema disminuye la capacidad de esta aplicación para reducir los efectos negativos producidos por este tipo de técnica de comprensión (C2). La abstracción mostrada con SHriMP se corresponde con las clases del programa y los niveles de abstracción más bajos con los métodos de las clases. Sin embargo, esta herramienta no tiene estrategias para obtener abstracciones que se correspondan con la funcionalidad del sistema (C3). SHriMP no tiene técnicas para la comprensión dirigida por hipótesis (C4) pero provee visiones de la arquitectura del sistema utilizando grafos anidados (C5). Como podemos observar esta herramienta tiene mecanismos de comprensión top down y bottom up por lo tanto posee algún soporte para múltiples modelos mentales (C6). La distinción entre diferentes modelos en SHriMP es difícil porque la implementación de los mismos es difusa (C7).

En cuanto a los criterios para reducir el tiempo de cognición del programador podemos afirmar que SHriMP tiene navegación direccional (C8), arbitraria (C9) y no provee navegación entre modelos (C10). Por otra parte, la herramienta tiene estrategias para mostrar el contexto de trabajo del programador (C11), por ejemplo usa cuadros iluminados para enfatizar el lugar de trabajo corriente. En la representación del sistema de estudio usando grafos la herramienta rotula los nodos en el paso. En otras visiones gráficas permite acceder al código del sistema (C12). Básicamente esta aplicación presenta una visión gráfica del sistema, por lo tanto el usuario puede observar el grafo correspondiente en su totalidad. SHriMP no tiene otros métodos para indicar las opciones que permitan alcanzar nuevos nodos (partes de la aplicación de estudio) (C13). Finalmente presenta muchas representaciones del sistema que pueden causar desorientación. Particularmente, cuando el sistema es muy grande su grafo también es muy grande. Sin embargo, SHriMP tiene formas de dibujar un grafo que lo muestran en una forma elegante (C14, C15). Creole es una versión de SHriMP adaptada para usarla con Eclipse SDK por lo tanto tiene las mismas características que SHriMP incorporándosele además las del ambiente Eclipse SDK.

2.2.2. Jeliot 3

Jeliot 3 es otra herramienta para la comprensión de programas más simple que SHriMP y Creole. Jeliot 3 fue diseñado para la enseñanza de la POO por lo tanto permite el estudio de sistemas pequeños. Esta herramienta presenta una visión dinámica del sistema y un ambiente de programación muy simple. Por consiguiente, no tiene muchos de los criterios usados aquí. Creemos que la principal razón de este desafortunado resultado es que para la evaluación de sistemas con estas características necesitamos criterios orientados a las estrategias de comprensión de programas dinámicas. La tabla 1 muestra las herramientas y los criterios que ellas cumplen.

2.2.3. Codesurfer

Codesurfer es una herramienta muy completa que se usa para la comprensión de programas. La precisión de esta aplicación permite reconocer fácilmente las relaciones sintácticas posibilitando el acceso inmediato a las unidades atómicas del sistema como por ejemplo el código fuente. Al momento de realizar el acceso ilumina las partes pertinentes del texto facilitando de esta forma su ubicación. Además, tiene mecanismo que permiten inferir el flujo de control del sistema, por ejemplo, cuando posibilita determinar que funciones serían utilizadas cuando se llama a una función utilizando una variable puntero. La forma que tiene de mostrar el sistema de estudio permite reducir el efecto de los planes deslocalizados (C1). Por ejemplo, al permitir ver el sistema utilizando diferentes visiones: código fuente, grafos de llamadas, entre otras tantas, posibilita la reducción de la desorientación a la cual está sujeto el programador cuando analiza un sistema utilizando este tipo de estrategias (C2). Codesurfer también provee mecanismos de abstracción. Es posible observar los distintos módulos del sistema en forma textual y además se puede bajar un nivel de abstracción, ver las funciones definidas en esos módulos y también observar las variables declaradas en esas funciones. Asimismo se pueden mirar abstracciones a nivel gráfico; el gráfo de llamadas a funciones se puede ir expandiendo de forma tal de mostrar todas las interconexiones existentes y también se puede llegar hasta el lugar donde la función fue definida (C3). Esta herramienta no permite comprensión dirigida por la hipótesis – orientada al objetivo ya que no presenta formas de ingreso, manipulación, y uso de hipótesis (C4). El sistema no muestra una vista de la arquitectura del sistema (C5). Al no poseer soporte para la comprensión de programas top down esta aplicación no integra las dos principales técnicas de comprensión (top down y bottom up). Esta herramienta permite observar el sistema de estudio utilizando diferentes visiones. Estas visiones pueden ser usadas para la utilización de diferentes modelos cognitivos (C6). Si bien las múltiples

visiones admitidas por Codesurfer posibilita el uso de distintos modelos mentales. Sin embargo la implementación de cada uno de estos es vaga. Es más, no se percibe claramente la presencia explícita de alguno de ellos; por lo tanto, la aplicación, no provee navegación entre distintos modelos mentales (C7). La herramienta posee diversos mecanismos de navegación secuenciales y arbitrarios. El programador puede leer código fuente en forma secuencial y luego navegar a través de él en forma totalmente arbitraria. El grafo de llamadas es una componente que facilita esta actividad (C8) (C9). Por otra parte este software no provee navegación entre modelos mentales (C10). Codesurfer posee mecanismos claros para indicarle al programador el lugar del sistema donde se encuentra trabajando. Además de la iluminación correspondiente (estrategia común de ubicación), posee otra aproximación utilizando el grafo de llamadas. Si un programador desea inspeccionar un módulo puede crear el grafo de llamadas y luego hacer clic en la función deseada y la herramienta mostrará el código de dicha función y conjuntamente dejará iluminado el nodo del grafo de llamadas (C11). El mecanismo mencionado previamente permite que Codesurfer muestre parcialmente el paso que se necesitó recorrer hasta llegar al lugar corriente de trabajo (C12). Esta herramienta también provee un mecanismo de expansión de nodos. Una vez construido el grafo de llamadas se puede visualizar con nodos etiquetados con el signo – o el signo +. El primero indica que el nodo puede ser condensado en uno solo y el segundo que puede ser expandido en un nivel. Esto en cierta manera, aunque no tan explícitamente, es un mecanismo para permitir alcanzar nuevos lugares del sistema de estudio (C13). La interfaz de usuario está claramente definida presentado inconvenientes de interpretación solamente cuando el programador mantiene abiertas un gran número de ventanas (C14). Finalmente, los gráficos y las representaciones que posee del sistema de estudio son claros y adecuados para la comprensión del mismo (C15).

2.2.4. Imagix 4D

Imagix 4D posee un conjunto de facilidades que posibilitan acceder a las componentes del sistema de estudio tales como código, representaciones visuales. Además, las relaciones entre los elementos son sintáctica y semánticamente claras (C1). La desorientación se puede reducir utilizando distintas visiones del sistema que permitan una fácil navegación, como por ejemplo mostrar el grafo de llamadas y haciendo clic en cada uno de sus nodos ir observando el código fuente. Podemos decir que Imagix 4D reduce la desorientación y esto es una ayuda para subyugar los planes deslocalizados (C2). Los mecanismos de abstracción que posee esta herramienta consisten en: una visión de las clases del sistema (si éste fue desarrollado con un lenguaje orientado a objetos), expansión de los nodos del grafo de llamadas y variables, el grafo de llamadas y variables que es una abstracción del código fuente (C3). Imagix 4D no posee mecanismos para el manejo de hipótesis (C4). Por otra parte, la herramienta provee diferentes instrumentos para visualizar la arquitectura del sistema como por ejemplo: diagramas de clases, diagramas de estructuras, entre otras (C5). La aplicación posee soporte para múltiples modelos mentales y diferentes visiones del sistema de estudio, por ejemplo: textuales (código fuente), gráficas (grafo de llamadas y usos de variables), diagramas de clases de UML, diagramas de estructuras, entre otros tantos (C6). Las referencias cruzadas entre diferentes modelos mentales pueden ser obtenidas por medio de la focalización de la/s componente/s del sistema que se está estudiando (C7). Además, la herramienta posee mecanismos para soportar navegación direccional. Por ejemplo, a través del diagrama de control de flujo se puede ir observando el código fuente (C8). Con Imagix 4D no es necesario seguir los enlaces estrictamente. El programador puede visualizar componentes en forma arbitraria (C9). No se puede inferir si la herramienta posee facilidades para la navegación entre los diferentes modelos mentales debido a que la implementación de los mismos es imprecisa (C10). La herramienta posee distintas formas de indicar el foco actual, un ejemplo de esto es el diagrama de control de flujo (C11). Por otra parte, posee un mecanismo de navegación similar a los utilizados por los navegadores que permite conocer la secuencia de acciones que se llevaron a cabo para llegar al estado actual de análisis (C12). Al tener una vista general del sistema, ya sea a través del diagrama de clases de UML, los diagramas de control de flujo, entre otros tantos, el programador puede alcanzar nuevas partes del sistema con facilidad (C13). La interfaz gráfica y textual de Imagix 4D es clara y no dificulta la tarea de comprensión de programas (C14). Esta herramienta posee diferentes estilos de presentación cada uno de ellos, según nuestro criterio, se adecua a la característica de estudio que estamos llevando a cabo en el sistema (C15). Por ejemplo, si estamos analizando la arquitectura del sistema el diagrama de clases del mismo es una presentación óptima.

2.3. Conclusiones de la Evaluación de Herramientas de Comprensión

La tabla 1 muestra un resumen de los resultados obtenidos luego de realizar la evaluación de las herramientas de comprensión. En la fila final se encuentra calculado el promedio de criterios poseídos por las herramientas de comprensión estudiadas. Como se puede observar, las más completas son Imagix 4D y Codesurfer siguiéndole en jerarquía SHriMP y Creole. Con respecto a éstas últimas podemos decir que Creole posee más ventajas que SHriMP porque, además de su propio ambiente se incorpora el de Eclipse SDK que también posee facilidades de Comprensión.

El pobre desempeño de Jeliot 3 se debe a que se necesitan incorporar o redefinir criterios que contemplen con más exactitud las características dinámicas de las herramientas de comprensión. Esta última peculiaridad nos permite inferir que se necesitan más trabajos de investigación y desarrollo en este aspecto.

Criterio	ShriMP	Creole	Jeliot 3	Imagix4D	Codesurfer
C1	Si	Si	No	Si	Si
C2	No	No	No	Si	Si
C3	No	No	No	Si	Si
C4	No	No	No	No	No
C5	Si	Si	No	Si	No
C6	Si	Si	Si	Si	Si
C7	No	No	Si	Si	No
C8	Si	Si	No	Si	Si
C9	Si	Si	No	Si	Si
C10	No	No	No	No	No
C11	Si	Si	Si	Si	Si
C12	Si	Si	No	Si	Si
C13	No	No	No	Si	Si
C14	Si	Si	Si	Si	Si
C15	Si	Si	Si	Si	Si
Promedio	0.6	0.6	0.33	0.86	0.73

Tabla 1*:* Resumen de los resultados de la evaluación de las herramientas de comprensión

3 CASO DE ESTUDIO: UN EVALUADOR DE RUTEO GEOMÉTRICO

En esta sección describimos una herramienta que permite evaluar algoritmos de ruteo. Esta aplicación esta destinada a estudiar el funcionamiento y realizar análisis del desempeño de esta clase de algoritmos. Presentamos la arquitectura y funcionalidades de la herramienta. Luego nos centraremos analizar la posibilidad de incrementar su capacidad explicativa a través de la incorporación de estrategias de comprensión de programas. El lector interesado puede encontrar en [1,2,3,4,6,7,10] los detalles internos del evaluador y resultados de la evaluación de algoritmos de ruteo geométrico.

3.1 Arquitectura

En esta sección presentamos la arquitectura del software de evaluación y describimos sus componentes. En términos generales, una herramienta para la evaluación de algoritmos de ruteo, debería: i) facilitar la incorporación de nuevas estrategias de búsqueda de caminos; ii) contemplar un conjunto de métricas y criterios de evaluación; iii) permitir la incorporación de diferentes topologías de red y iv) generar automáticamente los resultados. En la figura 1 mostramos una arquitectura que reúne estos requerimientos.

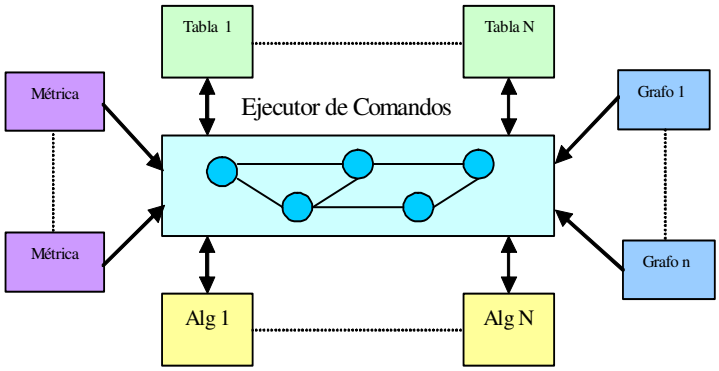


Figura 1: Arquitectura del Simulador

Como podemos observar esta arquitectura permite la incorporación de algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación con simplicidad; solo necesitamos que la interfaz entre cada una de estas componentes y el ejecutor de comandos esté bien definida.

El ejecutor de comandos es una componente de software, programable por el usuario del sistema orientada a comandos. Los comandos son programas cuya finalidad es implantar un algoritmo de ruteo o bien realizar de distintas clases de evaluación. Cada algoritmo o evaluación se construye haciendo uso de un conjunto de funciones primitivas que incorporamos en el evaluador; las cuales se corresponden con la implantación de diferentes tipos de datos abstractos que son útiles para abordar esta clase de problemas, como por ejemplo: grafos, grafos planares, triangulaciones, listas, pilas, colas, entre otros tantos tipos de datos abstractos.

Los algoritmos de ruteo son una componente principal del sistema. Como podemos apreciar en la arquitectura mostrada en la figura 1, el sistema pretende dar la posibilidad de que los algoritmos de ruteo puedan ser incorporados por el

usuario. Para esto necesitamos definir una interfaz estándar entre los algoritmos de ruteo y el evaluador. Generalmente esta interfaz está dada por el nodo origen, nodo destino, tamaño de la memoria que usa el algoritmo, longitud de paso y posiblemente una función de evaluación.
Ciertos algoritmos de ruteo necesitan que los nodos y los arcos posean atributos para que estos puedan funcionar. Dichos atributos pueden ser incorporados por los usuarios, ya que junto con el sistema proveemos una librería extensible de tipos de datos destinados a facilitar la incorporación de estas nuevas características. Por consiguiente, la incorporación de algoritmos de ruteo podría llevarse a cabo sin mayores inconvenientes.
Las métricas y los criterios de evaluación ocupan un lugar muy importante en el sistema. A través de estas componentes se hace posible la selección del mejor algoritmo de ruteo de acuerdo a una métrica o criterio determinado. En general, los procedimientos de cálculo de los valores de las métricas y criterios de evaluación recibirán como entrada el camino encontrado por la estrategia de ruteo de estudio.
Las tablas de resultados proporcionan al usuario la facilidad de observar los resultados. Cada tabla registrará los valores de las métricas y criterios de evaluación obtenidos a través de la ejecución de los algoritmos de ruteo incorporados en el evaluador. Pensamos que estos valores pueden ser recuperados por utilidades que permitan una representación gráfica de los mismos.

3.2 Funciones

El evaluador posee dos modos de operación: *manual* y *automático*. Cada uno de ellos posee funciones que posibilitan llevar a cabo diferentes clases de estudios sobre los algoritmos de ruteo. En esta subsección describimos las funciones de ambos modos con la finalidad de que el lector conozca las capacidades de esta aplicación y luego, en la subsección siguiente, nos centraremos en aumentar sus capacidades de explicación.

3.2.1. Funciones del Modo Manual de Ejecución

En ciertas ocasiones necesitamos analizar el comportamiento y/o funcionamiento de algoritmos de ruteo particulares. Esto se debe a que podemos obtener resultados no previstos en la evaluación de los algoritmos, no conocemos el funcionamiento porque fueron programados por otras personas, entre otras tantas cosas. Por esta razón, incorporamos un modo manual de trabajo.
El modo manual consta de una interfaz gráfica, amigable al usuario, para la visualización de las clases de grafos y del camino encontrado por el algoritmo de ruteo que se está evaluando. Las funcionalidades provistas por la interfaz son las siguientes:

- Ingreso de redes, por parte del usuario, en forma manual. El evaluador está preparado para crear automáticamente un conjunto de clases de grafos. Sin embargo, existen situaciones en donde se desea estudiar un algoritmo de ruteo sobre una red que no forma parte del evaluador. En este caso, el usuario del sistema tiene dos posibilidades: i) construir una rutina que la genere automáticamente e incorporarla al evaluador; ii) construir el grafo en forma manual utilizando la interfaz gráfica. La primera opción es una operación primitiva del sistema. La segunda es útil para realizar estudios particulares y analizar la factibilidad de incorporación de nuevas rutinas al evaluador. Por otra parte, el modo manual posibilita que el usuario del sistema pueda modificar una red generada automáticamente con el fin de adaptarla a una situación específica.
- Inicialización de los parámetros para los algoritmos de ruteo. A través de la experiencia en el entendimiento e implementación de los algoritmos de ruteo detectamos que ciertos algoritmos requieren parámetros adicionales a los ya conocidos (clase de grafo, nodo origen, nodo destino), como por ejemplo, cantidad de memoria utilizada, longitud del camino, entre otros tantos. El modo manual posibilita que el usuario indique estos valores de entrada por medio de una línea de comandos o a través de cuadros de diálogos.
- Ejecución parcial. Esta funcionalidad permite ver paso a paso los nodos que el algoritmo de ruteo de estudio selecciona. Es útil para aquellos algoritmos en donde la selección del próximo nodo en el camino no es determinística y cuando el funcionamiento del algoritmo de ruteo no se conoce.
- Ejecución total. Ejecuta el algoritmo de ruteo en su totalidad y muestra el camino correspondiente. Esta opción es útil para comparar visualmente los caminos encontrados por diferentes algoritmos de ruteo.
- Visualización de las tablas con los resultados de las métricas y criterios de evaluación. Esta operación se necesita para el análisis de los resultados.
- Reconstrucción parcial del sistema. La incorporación de métricas, clases de grafos, algoritmos de ruteo, conduce a una compilación de los distintos módulos que conforman el evaluador. Sin embargo, no es necesario un análisis profundo para comprender que solo el algoritmo que se está incorporando debería ser compilado. En este sentido, la herramienta posee la reconstrucción parcial. La primera vez que el sistema se pone en funcionamiento crea módulos objetos y librerías necesarias para que pueda ser usado. Este conjunto de rutinas conforma el núcleo de la herramienta. Por lo tanto, cuando se incorpora una nueva componente de software (algoritmo de ruteo, métrica, criterio de evaluación, clase de grafo) se enlaza con las rutinas del núcleo de la herramienta. Este procedimiento es transparente al usuario.

- Reconstrucción total del sistema. En ciertas ocasiones, el usuario necesita la incorporación de clases de grafos, algoritmos, métricas o criterios de evaluación que poseen atributos no contemplados por la herramienta. En este caso se debe modificar el núcleo del sistema para aumentar su funcionalidad. El núcleo ha sido construido con tipos de datos abstractos con interfaces y operaciones bien definidas que junto con una documentación formal, realizada con herramientas proporcionadas por la Ingeniería del Software, facilitan la modificación de las rutinas del núcleo. Una vez realizados los cambios se debe recompilar el sistema en su totalidad (rutinas del núcleo, algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación). Si bien se espera que los usuarios del sistema sean informáticos la cantidad de rutinas utilizadas (las del núcleo y aquellas que han sido incorporadas en usos previos) dificulta que el usuario realice esta tarea en forma manual. Para subsanar este inconveniente, la herramienta posee un método de compilación automático que hace que el procedimiento de reconstrucción total de la herramienta sea, al igual que la reconstrucción parcial, transparente al usuario.
- Interacción con otras utilidades. Como hemos descrito previamente, la herramienta necesita básicamente de un compilador para su reconstrucción parcial o total y un editor de texto para escribir las rutinas definidas por el usuario. Por otra parte es una característica deseable que el sistema posea utilidades que faciliten el estudio y seguimiento de los algoritmos de ruteo. En este sentido, el ambiente del sistema permite que el usuario pueda incorporar las aplicaciones que considere necesarias para simplificar la operación de la herramienta, los estudios y producir resultados más precisos.
- Creación de comandos de alto nivel. Si bien la herramienta está orientada a expertos informáticos es posible que sea utilizada por profesionales de áreas afines. Por esta razón, el sistema admite la creación de comandos que son simples de ejecutar, incorporados a su barra de menú principal, y administrar.
- Personalización del ambiente gráfico de trabajo. El usuario puede personalizar su ambiente gráfico de trabajo cambiando el color de las componentes, incorporando nuevos programas entre otras operaciones.
- Visualización de información relacionada con los algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación. A medida que el usuario trabaja con la herramienta va incorporando nuevos algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación. Este incremento en rutinas hace que sea difícil recordar, aún colocando nombres semánticamente claros, la funcionalidad y significado de los parámetros utilizados por cada una de ellas. Para subsanar este inconveniente, la herramienta posee subambientes dedicados para cada una de las clases de rutinas de interés. En cada uno de ellos se puede visualizar el nombre de la rutina, su ubicación en el árbol de carpetas del disco, los parámetros que utiliza y una descripción en texto de la tarea que realiza.
- Facilidades para la incorporación, eliminación y modificación de los algoritmos de ruteo, clases de grafos y métricas. En cada uno de los subambientes destinados a los algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación, la herramienta permite la supresión, inserción y recuperación de la información necesaria para la modificación de la rutina que se desee.

3.2.2. Funciones del Modo Automático de Ejecución

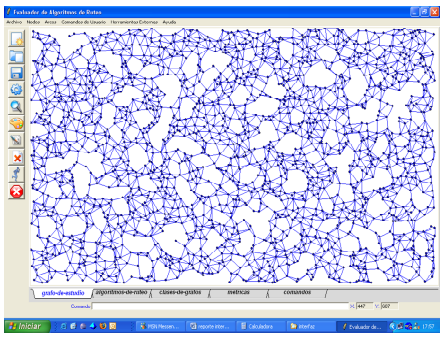
Hasta ahora hemos descrito las funcionalidades que presenta el modo manual de la herramienta. Sin embargo, para realizar estudios empíricos con el fin de hacer generalizaciones, es necesario la ejecución de experimentos que implican la utilización de una gran variedad de algoritmos de ruteo, clases de grafos, métricas y criterios de evaluación. Para proveer esta funcionalidad incorporamos en la herramienta un modo de ejecución automático. Con este modo, el usuario programa el experimento, lo incorpora a la herramienta, utilizando las funciones del modo manual y lo ejecuta. La ejecución puede ser llevada a cabo a través de una línea de comandos o cuadros de diálogos creados automáticamente por el sistema.

La realización de experimentos, en general, implica el uso de algoritmos con una alta complejidad computacional y por consiguiente la herramienta quedará bloqueada por mucho tiempo. Solucionamos este problema a través del uso de la ejecución en forma secundaria (background) que permite que un comando se ejecute y deja disponible a la herramienta para la ejecución de otro experimento o bien para el uso del modo manual. La incorporación de esta peculiaridad no implica un gran esfuerzo de programación ya que forma parte de las operaciones primitivas de los lenguajes que se usaron para construir la herramienta.

El uso del modo automático sigue los mismos lineamientos que para el modo manual. Las diferencias principales son las siguientes: i) no hace falta la generación de un grafo ya que esta tarea esta embebida en el experimento; ii) los comandos que se ejecutan implican la generación de muchas redes, ejecución repetida de uno o más algoritmos de ruteo y la obtención de resultados, proporcionados por las métricas y criterios de evaluación, para llevar a cabo estudios estadísticos. El resto de las operaciones proporcionadas por la herramienta son transparentes al usuario y su comprensión es intuitiva. El lector interesado puede ver los detalles internos de los modos de ejecución del evaluador en [7].

El evaluador tiene incorporado, como funciones primitivas, las siguientes clases de grafos: *Grafo Unidad*, *Grafo de Morelia*, *Grafo de Gabriel*, *Grafo de Vecindad Relativa*, *Triangulación de Delaunay*. También posee los siguientes algoritmos de ruteo: *Ruteo por Brújula*, *Ruteo Voraz*, *Ruteo Voraz-Brújula*, *Ruteo por Brújula Aleatorizado*, *Ruteo por Caras*. Asimismo posee las métricas: *Tasa de Éxito*, *Dilatación Euclidiana Promedio* y *Dilatación de Enlace Promedio*. En la figura 2 se pueden observar distintas ventanas del evaluador. En la figura 2.a se puede ver el resultado de un

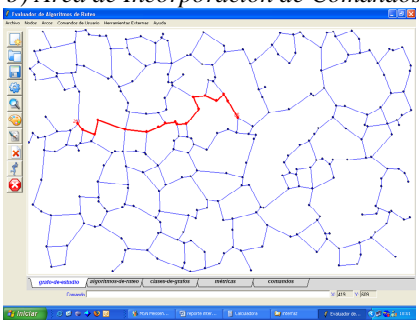
comando que construye a un Grafo de Morelia de aproximadamente 1000 nodos. En la figura 2.b se muestra la interfaz que permite la incorporación de nuevos comandos al sistema, es importante notar que en otras solapas del ambiente gráfico se puede visualizar la forma de incorporar algoritmos de ruteo, métricas y clases de grafos. En la figura 2.c se puede visualizar un Grafo de Vecindad Relativa con la interfaz de un algoritmo de ruteo específico. Dicha interfaz permite seleccionar el nodo origen y destino y el archivo de salida. Finalmente en la figura 2.d se muestra el camino encontrado por una estrategia de ruteo al vuelo.



a) Grafo de Morelia

b) Área de Incorporación de Comandos

c) Búsqueda de un camino



d) Selección de un camino

Figura 2: Ventanas típicas del Evaluador de Algoritmos de Ruteo

3.4. Necesidad de Explicar Operaciones Internas

En esta sección mostramos la necesidad de explicar operaciones internas cuando se intenta estudiar los algoritmos de ruteo. Con este objetivo tomamos como ejemplo al ruteo por brújula, una estrategia de ruteo incorporada al evaluador, y veremos que su especificación en términos matemáticos es simple, sin embargo su implementación requiere de distintas subrutinas que dificultan la comprensión del programa que lo implementa.

El ruteo por brújula es una estrategia de Ruteo Geométrico que funciona como sigue: Suponga que $G=(N,R)$ es un grafo geométrico plano, es decir los nodos del conjunto N están conectados por segmentos de línea recta y no existen cruces entre los segmentos de G. Además, admita que el nodo u desea enviar un mensaje al nodo v. El nodo u sólo conoce las coordenadas de sus vecinos y del nodo destino. Si u usa el ruteo por brújula entonces envía el mensaje al vecino de él que minimiza el ángulo con respecto a la recta determinada por u y v. La figura 3 muestra la operación del ruteo por brújula gráficamente.

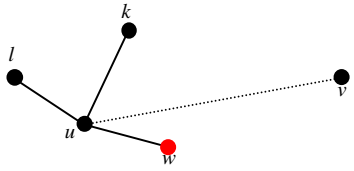


Figura 3: Operación del Ruteo por Brújula

Matemáticamente este algoritmo se puede definir como sigue:

$$\text{RuteoPorBrújula}(u,V^1(u),v)= w_1 \in V(u) : \angle w_1uv < \angle w_2uv \text{ para todo } w_2 \in V(u) \quad (1)$$

[1] $\mathbf{V}(u)$ hace referencia a la operación que calcula los vecinos del nodo u en un grafo $G=(N,R)$ con $u \in N$.

Podemos observar que la definición (1) es simple y elegante. Sin embargo, su implementación necesita de otros mecanismos ocultos en la definición (1) como veremos a continuación.
Una implementación posible del ruteo por brújula debería:

1. Agrupar los vecinos del nodo *u* en tres conjuntos. *Izquierdos*, *Centros*, *Derechos*. El conjunto *Izquierdos* contendrá a los vecinos de *u* que están a la izquierda del segmento *uv*. El conjunto *Centros* contendrá a los nodos que están sobre el segmento y finalmente *Derechos* contendrá los nodos que están a la derecha del segmento *uv*.
2. Si el conjunto *Centro* es distinto de vacío. Entonces seleccionar un elemento de él y finalizar el algoritmo. Caso contrario continuar con el paso 3.
3. Seleccionar, utilizando los conjuntos *Izquierdo* y *Derecho*, los vecinos del nodo *u* más cercanos a la línea recta determinada por los nodos *u* y *v*.
4. Sean *k* y *w* los nodos elegidos en el paso 3. El algoritmo debe tomar el nodo que tiene ángulo más pequeño.

Cada uno de los pasos que implementan al ruteo por brújula se traducen en una o varias rutinas y por lo tanto la implementación de la fórmula (1) se traduce en muchas líneas de código. Lo mismo sucede con la implementación de distintas estrategias de ruteo en general.
En este simple ejemplo podemos observar lo siguiente:

Una definición matemática elegante y sintética oculta estrategias que:

- *Posibilitan una visión parcializada del algoritmo.*
- *Requieren de muchos algoritmos para su implementación.*

Esta característica dificulta comprensión de programas de este tipo a partir de su implementación.

Con lo presentado en párrafos anteriores intentamos mostrar que una estrategia de ruteo simple de especificar puede ser difícil de comprender a partir de su implementación. Por esta razón, poseer una herramienta que permita explicar las acciones que toma un algoritmo de ruteo es fundamental para comprender algoritmos de ruteo tomando como base su implementación.

4 MODELO POSIBLE PARA LA COMPRENSIÓN DE ALGORITMOS DE RUTEO GEOMÉTRICO

Básicamente un sistema geométrico puede ser analizado usando las mismas estrategias utilizadas para otros sistemas. Sin embargo, creemos que esta clase de algoritmos necesita desarrollos visuales y de animación más avanzados. Por esta razón, el modelo cognitivo utilizado para entender este tipo de programas debe incluir diferentes visiones. Estamos principalmente interesados en la visión operacional y comportamental [8]. Nos hemos percatado, a través del estudio del estado del arte, que existen diversos trabajos que abordan la visión operacional pero no sucede lo mismo con la visión comportamental.
Pensamos que el modelo cognitivo más apropiado para la comprensión de programas debe soportar ambas formas de aprendizajes: top down y bottom up, porque estimamos que el uso de una estrategia u otra depende de la estructura de pensamiento del programador y de las características de la aplicación que estudia. Por esta razón utilizamos el *Modelo Integrado de Comprensión de Programas* [12,14,17]. Creemos esta opción es la más atractiva porque hace posible entender programas en forma top down y bottom up. Además este modelo tiene otros aspectos interesantes por ejemplo la base de conocimiento que representa todo el trasfondo conceptual del programador antes de comenzar la tarea de mantenimiento, evolución o comprensión.

4.1. Criterios de Evaluación para los Algoritmos de Ruteo Geométrico

En esta sección presentamos algunos criterios para evaluar/construir herramientas de comprensión de programas destinadas a estudiar los Algoritmos de Ruteo Geométrico.
Nuestro estudio estará enfocado en el aspecto comportamental de los algoritmos de ruteo geométrico. Esto se debe a que la vista operacional puede ser obtenida usando las herramientas de comprensión de programas clásicas.
En este contexto podemos decir que la representación visual de las operaciones y las animaciones son muy importantes. Consecuentemente, la herramienta de comprensión de programas debería tener:

- Un sistema de visualización interactivo / automático.
- Un modo de ejecución paso a paso.

El primer ítem puede ser usado para generar casos de pruebas aleatorios y específicos porque normalmente se desea estudiar el comportamiento en términos generales de esos algoritmos. Por otra parte, esas clases de algoritmos tienen actividades particulares que algunas veces determinan la diferencia entre una estrategia y otra. Esta es la razón de la necesidad de un modo interactivo de ejecución. La disponibilidad de un modo paso a paso nos permitirá estudiar los algoritmos en profundidad.
Las herramientas para estudiar algoritmos de ruteo geométrico tendrían que permitir visualizar las operaciones internas usadas por las operaciones geométricas. Este parámetro hace posible analizar la implementación de esas funciones. El

lector puede ver un ejemplo cuando se explicó el funcionamiento interno del ruteo por brújula. Por consiguiente otro criterio de evaluación es:

- La herramienta permite visualizar las operaciones geométricas de bajo nivel.

Esta característica es importante porque posibilita incorporar otra componente al modelo cognitivo usado. Construir una herramienta con todas las operaciones geométricas es imposible. Sin embargo, podemos concebir que el modelo cognitivo tiene una *Base de Conocimiento Asociada a la Aplicación* (BCAA). En otras palabras, el sistema de comprensión de programas tendrá un conjunto de operaciones posibles de visualizar pero, al igual que los humanos, tendría asociados mecanismos de aprendizaje que posibilitarían incrementar la capacidad de BCAA. Esta característica nos permitiría construir un sistema cada vez más experto en la comprensión de algoritmos de ruteo. Consecuentemente otro criterio de evaluación es:

- El sistema tienen la capacidad de incorporar nuevas primitivas para dejarlas disponibles para usos futuros.

Usualmente los humanos cuando resuelven problemas hacen tareas elementales y anotan los resultados parciales para potenciales usos futuros. Estos resultados pueden o no ser pertinentes por lo tanto no necesita ser almacenados en una memoria permanente. Llamaremos a este almacenamiento Memoria de Trabajo. Por lo tanto otro criterio de evaluación es:

- El sistema tiene elementos visuales para almacenar resultados parciales.

Y otro derivado de éste:

- El sistema tiene mecanismos para mostrarle al usuario los resultados parciales pertinentes al contexto del programa.

En ciertas ocasiones es necesario repetir alguna secuencia de animación para analizar situaciones claves. Por esta razón es deseable que el software de comprensión tenga esta alternativa.

- El sistema tiene estrategias para repetir eventos.

Pensamos que para considerar óptima una herramienta de comprensión de programas de estas características es necesario que posea requerimientos relacionados con la dinámica del programa. Además debe reunir los criterios para estudiar la visión estática del mismo.

4.2. Incorporación de Mecanismos de Explicación y Generalización

El evaluador de algoritmos presentado permite realizar animaciones que consisten en la visualización paso a paso de la selección del próximo nodo en el camino. Sin embargo, como describimos en la sección anterior, para comprender un algoritmo a partir de su implementación se necesitan más detalles. Con el fin de solucionar este problema, pensamos que es útil que el evaluador pueda ir explicando las tareas que un algoritmo de ruteo realiza cuando selecciona el próximo nodo. Pensamos que una forma útil de realizar explicaciones es a través de la visualización de las funciones que el algoritmo de ruteo invoca. A medida que el evaluador de algoritmos va mostrando paso a paso la selección del próximo nodo se visualizarán las funciones que la estrategia de ruteo invoca para llevar a cabo la acción.

Para realizar este objetivo estamos desarrollado, utilizando las herramientas lex y yacc, un analizador sintáctico del lenguaje ANSI-C. Este analizador tiene incorporada acciones semánticas que permiten detectar las funciones y llamadas a funciones. Esta tarea es útil porque posibilitará la incorporación de invocaciones a una función testigo que realice alguna actividad, como por ejemplo la impresión del nombre de la función que invoca y el nombre de la función invocada, deseada por el programador que estudia los algoritmos de ruteo. La salida de este proceso es un nuevo programa que implementa la misma estrategia de ruteo pero que posee funciones extras que permiten explicar, en forma textual y por medio de nombres de función, un algoritmo de ruteo. Por consiguiente, esta versión del evaluador contendrá una explicación gráfica que muestra el próximo nodo y una textual en la que se podrán ver las funciones implicadas en esa tarea. Ambas visones se visualizarán en simultáneo en la pantalla proporcionando al programador una perspectiva más completa. Al haber construido el analizador sintáctico también se podrá, haciendo un clic con el ratón, acceder al código fuente de las funciones invocadas por la estrategia de ruteo.

Creemos que una generalización posible es la siguiente. Los nodos del grafo son los módulos de un sistema genérico. La relación es la de comunicación entre módulos es decir: el módulo *M* esta en relación con el módulo *N* si *M* invoca a una función definida en el módulo *N*. La estrategia de visualización es la misma que la utilizada para los algoritmos de ruteo . La diferencia radica en que los nodos (módulos) que se iluminarán son aquellos que contienen a las funciones que se están invocando; y los arcos resaltados serán aquellos que conecte a módulos que invocan a funciones no definidos en ellos. Finalmente, para logar un lenguaje más unificado pensamos cambiar la representación estándar de grafo (como las definidas en el evaluador) utilizando UML.

5 CONCLUSIÓN

La comprensión de programas es un área de la Ingeniería del Software que presenta muchos desafíos de investigación. El desarrollo de sistemas en esta área implica un duro trabajo intelectual y laboral. Podemos hacer esta afirmación porque observamos el gran número de requerimientos que deben reunir las herramientas de comprensión de programas. También creemos que no existe una herramienta de comprensión de programas que satisfaga todos los requerimientos.

El análisis de las herramientas de comprensión de programas nos permitió detectar que existen estrategias de comprensión de programas poco utilizadas una de ellas es la animación, pensamos que esto se debe a su complejidad. Por otra parte, pudimos observar que se necesitan criterios de evaluación que contemplen características estáticas y dinámicas equitativamente.
Basándonos en los estudios de las herramientas de comprensión tradicionales y tomando como caso de estudio a los algoritmos de ruteo, por poseer una representación para la animación simple, definimos un conjunto de criterios de evaluación aplicables a las herramientas de comprensión de sistemas geométricos, y una técnica de animación que permite explicar acciones de los algoritmos de ruteo. Realizamos una generalización de la segunda describiendo brevemente como puede ser aplicada a sistemas en general.
Como trabajos futuros tenemos planificado a corto generalizar los criterios de evaluación definidos para herramientas de comprensión geométrica, extender la arquitectura del evaluador para que contemple la incorporación de herramientas de comprensión. Por otra parte, en el futuro, pensamos desarrollar una herramienta que implemente la generalización propuesta de nuestra estrategia de animación como así también un mecanismo más completo de evaluación teniendo en cuenta criterios de evaluación definidos por nuestro grupo de investigación.

Referencias

[1] Berón, M., Gagliardi, O., Hernández Peñalver, G. *Evaluación de Métricas en Redes de Computadoras*. Expuesto y publicado en el Congreso de Informática y Ciencias de la Computación CACIC 2003, realizado en la Universidad de La Plata. La Plata. 2003.

[2] Berón, M., Gagliardi, O., Flores, S. *Ruteo en Redes Inalámbricas*. Expuesto y publicado en el Acta de Resúmenes del Congreso Argentino de Ciencias de la Computación en el año 2002.

[3] Berón, M., Gagliardi, O., Hernández Peñalver, G. *"Evaluación de Algoritmos de Ruteo de Paquetes en Redes de Computadoras"*. Expuesto y publicado en el Workshop de Investigadores en Ciencias de la Computación (WICC 2004). Realizado en la Universidad Nacional del Comahue. Año: 2004.

[4] Berón, M., Gagliardi, O., Hernández Peñalver, G. *"Evaluación de Algoritmos de Ruteo en Redes de Computadoras"*. Expuesto y publicado en el V Workshop de Investigadores en Ciencias de la Computación, realizado en la Universidad Nacional del Centro. Tandil. 2003.

[5] Berón, M., Grosso, A., Gonzales, A. Printista, M., Maldocena, P., Ordoñes, G., Molina, S., Apolloni, R.. *"Una Aproximación hacia el estudio de los Sistemas de Computación"*. Expuesto y publicado en el V Workshop de Investigadores en Ciencias de la Computación realizado en la Universidad Nacional del Centro . Tandil. 2003.

[6] Berón, M., Hernández Peñalver, G., Gagliardi, O. *"Factibilidad de Uso del Ruteo Voraz en los Grafos de Gabriel, Vecindad Relativa y Triangulaciones"*. Expuesto y publicado en el Congreso Argentino de Ciencias de la Computación (CACIC 2004). Universidad Nacional de la Mantanza. Año: 2004.

[7] Berón, M; Hernádez Peñalver, G; Gagliardi, O. "*Un Evaluador de Algoritmos de Ruteo"*. Master Thesis in Software Engineering. Universidad Nacional de San Luís.

[8] Brooks, R. *"Using Behavioral Theory of Program Comprehension in Software Engineering"*. IEEE. 1978.

[9] Grosso, A., Riesco, D., Berón, M. *"Una Herramienta para la Ingeniería Inversa de Sistemas Escritos en Lenguaje C, Bajo Linux"*. Comunicación, Expuesta y Publicada en el Acta de Resúmenes del Congreso Argentino de Ciencias de la Computación CACIC en el año 2002.

[10] Hernández Peñalver, G., Berón, M., Gagliardi, O. *"Ruteo Geométrico Aplicado a las Redes de Computadoras"*. Expuesto y publicado en el Workshop de Investigadores en Ciencias de la Computación (WICC05). Universidad Nacional de Río Cuarto. Córdoba. 2005

[11] Linter, R; Michand, J; Storey, M; Wu, X. *"Plugging-in Visualization: Experiences Integrating a Visualization Tool with Eclipse"*. ACM Symposium on Software Visualizaton. 2003.

[12] Mayrhauser, A; Vans, M. *"Program Comprensión During Software Maintenace and Evolution"*. IEEE. 1995.

[13] Moreno, A; Myller, N; Satinen, E; Ben-Ari, M. *"Visualizing Programs with Jeliot 3"*. ACM. 2004.

[14] Oliveira, E; Henriques, P; Varanda, M. *"Características de um Sistema de Visualização para Compreensão de Aplicações Web através de Inspecção de Software e baseadas em Modelos Cognitivos"*. Tesis de Maestría en Ingeniería del Software. 23 de Febrero de 2006.

[15] Pacione, M; Roper, M; Wood, M. *"A Comparative Evaluation of Dynamic Visualization Tools"*. Proceedings of the 10th Working Conference on Reverse Engineering (WCRE'03). IEEE. 2003.

[16] Sajaniemi, J. "Program Comprehension through Multiple Simultaneous Views: A Session with VinEd". IEEE. 2000.

[17] Storey, D; Fracchia, F; Müller, H. "Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization". IEEE. 1997.