Comprensión de Programas por Inspección Visual y Animación

Mario M. Berón Roberto Uzal
Universidad Nacional de San Luis - Departamento de Infomática
San Luis - Argentina
mberon@unsl.edu.ar, ruzal@sinectis.com.ar

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

Maria J. Varanda Pereira
Instituto Politécnico de Bragança
Braga - Portugal
mjoao@ipb.pt

Resumen

PCVIA (**P**rogram **C**omprehension by **V**isual **I**nspection and **A**nimation) es un proyecto de investigación que estudia la construcción de métodos, técnicas y herramientas que ayuden al ingeniero del software en el análisis y comprensión de aplicaciones. Estos estudios tienen como objetivo contribuir en distintas actividades de la Ingeniería del Software como por ejemplo mantenimiento, reingeniería, ingeniería reversa, entre otras tantas aplicaciones. Para construir ambientes de comprensión de programas es necesario concebir herramientas que permitan extraer y visualizar información de los sistemas. Para lograr este objetivo es necesario analizar los métodos, técnicas, herramientas, etc. existentes con la finalidad de incrementar la funcionalidad de las mismas, o bien, proponer otras nuevas.
En este artículo describimos un abordage para la construcción de herramientas de comprensión que se basa en la instrumentación del código fuente del sistema de estudio. Entre los objetivos de esta aproximación se encuentran la elaboración de estrategias de navegación y relación entre las distintas perspectivas de sistemas desarrolladas usando el paradigma imperativo. Por otra parte se planifica analizar la extensibilidad de las mismas a otros paradigmas como por ejemplo el orientado a objeto.

Palabras Claves: Comprensión de Programas, Métodos, Técnicas, Herramientas.

1. Introducción

El proyecto PCVIA tiene como principal objetivo estudiar, explorar e implementar técnicas y herramientas de comprensión de programas. La comprensión de programas se traduce en la habilidad de entender una pieza de código escrito en un lenguaje de alto nivel. Un programa no es mas que una secuencia de instrucciones que serán ejecutadas de forma de garantir una determinada funcionalidad. El lector de un programa consigue extraer el significado de un programa cuando comprende de que forma el código cumple con la tarea para la cual fue creado.

El área de comprensión de programas es una de las más importantes de la Ingeniería del Software porque es necesaria para tareas de reutilización, inspección, manutención, migración y extensión de sistemas de software. Puede también ser utilizada en áreas como ingeniería reversa o enseñanza de lenguajes de programación. La tarea de comprensión de programas puede tener diferentes significados y puede ser vista desde diferentes perspectivas. El usuario puede estar interesado en como la computadora ejecuta las instrucciones con el objetivo de comprender el flujo de control y de datos, o puede querer verificar los efectos que la ejecución del programa tiene sobre el objeto que esta siendo controlado por el programa. Considerando estos dos niveles de abstracción, una herramienta versátil de inspección visual de código es crucial en la tarea de comprensión de programas.

Existe un conjunto enorme de herramientas de comprensión de programas [4][5][7] construidas para diversos lenguajes que usan varios abordages. En el ámbito del proyecto PCVIA están siendo desarrolladas herramientas usando diferentes perspectivas. En este artículo se usa un abordaje dependiente del lenguaje, que permite la generación de visualizaciones [3] en varios niveles de abstracción y refuerza la importancia del mapeamiento entre las mismas.

2. Sistema de Comprension de Programas

Normalmente, cuando el programador quiere entender un sistema necesita inspecionar diferentes aspectos del mismo. Para construir estas perspectivas es necesario extraer información desde los sistemas y representarla adecuadamente. La extracción de la información es importante porque es la base para construir diferentes vistas. Por otra parte, la visualización de la información es escencial para propósitos de comprensión. En las secciones siguientes describimos las vistas y las estratégias de extracción de información usadas en la construcción de herramientas de comprensión.

2.1. Vistas de un Sistema

Una vista es una representación de la información de un sistema que facilita la comprensión de un aspecto del mismo.

Cuando un programador esta comprendiendo un sistema la primera vista con la que se enfrenta es su código fuente. Esta vista es útil porque el programador esta familiarizado con los lenguajes de programación. Sin embargo, cuando el tamaño del sistema crece pierde claridad y otras prespectivas del sistema son necesarias.

Un aspecto del sistema que se encuentra en un nivel de abstracción más alto consiste en visualizar las funciones del sistema y las relaciones existentes entre las mismas. Un ejemplo de esto es el *Grafo de Funciones* (GF). GF es un grafo donde el conjunto de nodos esta compuesto por las funciones del sistema de estudio y la relación entre ellas está dada por la comunicación de las funciones através de sus invocaciones. Normalmente, el grafo GF es una vista deseada

por los programadores, sin embargo, de la misma forma que el código fuente, cuando el tamaño del sistema crece no presenta una ayuda a la comprensión. Como una alternativa al grafo GF el sistema puede ser visualizado a usando los módulos que lo componen. En este caso es posible definir un grafo que muestra la relación de comunicación entre ellos. Normalmente este grafo es conocido con el nombre de *Grafo de Comunicaciónes de Módulos* (GCM). Nuestros experimentos indican que GCM presenta una vista clara del sistema aun cuando el tamaño del mismo es grande. No obstante, al presentar un mayor grado de abstracción oculta detalles que pueden ser útiles en el proceso de comprensión.

Las vistas descriptas hasta este momento pueden ser construidas usando la información estática del sistema. Sin embargo no siempre todas las funciones o módulos son usados cuando el sistema bajo estudio se ejecuta. Teniendo en cuenta este aspecto es importante recuperar información dinámica para visualizar, animar o describir sólo las componentes utilizadas por el sistema. Por ejemplo, podría ser útil presentar un subrafo de GF o GCM mostrando sólo las componentes utilizadas.

Por otra parte también es posible visualizar el contenido los módulos objeto del sistema. Esta vista puede ser de importancia para el programador experto cuando desea modificar el código generado por el compilador.

Finalmente es importante notar que las vistas permiten mostrar aspectos del sistema bajo estudio en distintos niveles de abstracción.

2.2. Relación entre las Diferentes Vistas de un Sistema

Las vistas son importantes y ayudan en la comprensión de sistemas. Sin embargo, no es suficiente visualizarlas, es necesario posibilitar su navegación. Esto se debe a que normalmente el proceso de comprensión de programas implica visualizar aspectos de alto, medio y bajo nivel del sistema. Por ejemplo, puede ser importante estudiar cual es el módulo más importante del sistema, en ese caso GCM puede ayudar en esa tarea. Pero luego de identificar dicho módulo es interesante estudiar el GF correspondiente.

Teniendo en cuenta esta observación construimos un prototipo de una arquitectura que permite la navegación entre las distintas vistas del sistema de estudio. Acontinuación describimos las componentes básicas que la arquitectura contiene actualmente.

Sistema de Extracción de la Información: extrae información estática y dinámica de un sistema. Utiliza técnicas descriptas en [1][2][6][9] para lograr este objetivo.

Repositorio de Información: almacena datos producidos por el Sistema de Extracción de la Información, como por ejemplo: módulos, funciones, tipos, datos, etc.

Administrador de Visualizacion y Navegación: utiliza la información disponible en el repositorio de información para proporcionar navegación entre las distintas vistas.

Visualizador Operacional: consta de un conjunto de módulos que implementan las distintas vistas. Es el sistema encargado de mostrar los objetos del dominio del programa

Visualizador Comportamental: consite de la salida del sistema, en otras palabras el resultado.

La relaciones entre las diferentes vistas es llevada a cabo por el Administrador de Visualización y Navegación. Esta componente recupera información desde el Repositorio de Información para lograr su objetivo. Podemos decir que todas las relaciones, con excepción de la vinculación entre las vistas operacional y comportamental, son logradas de esta manera.

2.2.1. Estrategia de Relación Operacional-Comportamental

La estrategia de relación operacional-comportamental, denifinida por nuestro grupo de investigación, utiliza infromación dinámica y estática del sistema de estudio. Además se basa en la siguiente observación:

La salida de un sistema esta compuesta de objetos del dominio del problema. Usualmente estos objetos son implementados por tipos de datos abstractos, en el caso de lenguajes imperativos, o por clases, en el caso de lenguajes orientados a objetos. Tanto los TDAs como las Clases tienen objetos de dato que almacenan su estado y un conjunto de operaciones que los manipulan. Entonces es posible describir cada objeto del dominio del problema utilizando los TDAs o clases que los implementan.

Esta estrategia denominada BORS (Behavioral-Operational Relation Strategy) aplica los siguientes pasos para alcanzar su objetivo.

1. Detectar las funciones relacionadas con cada objeto del dominio del problema
2. Construir un árbol de ejecución de funciones usadas en tiempo de ejecución
3. Explicar las funciones encontradas en el paso 1 usando el árbol construido en el paso 2.

El lector interesado puede encontrar en [8] una explicación detallada de la estrategia BORS. En la próxima sección presentaremos las técnicas de extracción de la información usadas para recolectar datos desde el sistema bajo estudio.

3. Métodos de Extracción de la Información

Para la extracción de la información fue necesario construir un parser del lenguaje de programación utilizado por el sistema de estudio. Una vez realizado esta tarea se procedió a incorporar los atributos y acciones semánticas necesarias para extraer información estática del sistema.

Para recuperar la información dinámica del sistema se instrumentó el código fuente con funciones de inspección y control. Las primeras fueron insertadas al inicio y en los puntos de retorno de cada función. Las segundas son utilizadas para controlar las iteraciones ya que dentro de ellas pueden haber invocaciones a funciones. Estas, en algunos casos, son redundantes porque son utilizadas para inicializar estructuras de datos. Con este esquema se pudo recuperar y controlar el flujo de ejecución de funciones. El lector interesado puede encontrar una explicación detallada de estas aproximaciones en [6][8]. Para la recuperación de datos se construyó una tabla de símbolos del lenguaje de programación (en este caso C) que posibilita recuperar información detallada de cada una de los objetos de datos del sistema. Esta característica permitirá en el futuro extender el esquema de instrumentación para inspeccionar datos.

4. Conclusión

En este artículo presentamos el proyecto PCVIA y describimos las actividades actualmente realizadas. Presentamos características útiles de una herramienta de comprensión de programas. Describimos las componentes básicas de una arquitectura que responde a los requisistos de las herramientas basadas en vistas. Además, mencionamos que es importante que las herramientas de comprensión de programas permitan la navegación entre las distintas vistas que ellas proveen. Por otra parte, describimos un procedimiento denominado BORS para relacionar dos vistas muy

importantes como lo son la operacional y la comportamental. Esta última técnica es uno de los resultados recientes y relevantes de nuestra investigación.

Además de la estrategia BORS describimos sintéticamente distintas técnicas de instrumentación y extracción de la información que hemos desarrollado y aplicado con éxito. Es importante notar que estos procedimientos (los de instrumentación de código y extracción) no requieren intervención del usuario por ser totalmente automáticos. Los resultados obtenidos fueron satisfactorios en el sentido de que se lograron explorar algunos sistemas y relacionar distintas vistas con la información recuperada por nuestras técnicas de extracción de la información. Como trabajo futuro nos proponemos extender BORS con instrumentación de datos como así también proveer algún mecanismo de indentificación precisa de elementos del dominio del problema. Por otra parte, pretendemos construir un ambiente que permita decorar la salida del sistema con los objetos del dominio del programa.

Referencias

[1] Abdelwahab Hamou-Lhadj.*The Concept of Trace Summarization.*PCODA: Program Comprehension through Dynamic Analysis.(2005), 38-42.

[2] Andy Zaidman, Bram Adams, and Kris Schutter. *Applying Dynamic Analysis in a Legacy Context: An Industrial Experience.* PCODA: Program Comprehension through Dynamic Analysis (2005),6-10.

[3] Franoise Balmas, Harald Werts, Rim Chaabane. *DDGraph: a Tool to Visualize Dynamic Dependences.*Program Comprehension through Dynamic Analysis (2005), 22-27.

[4] `http://wiki.di.uminho.pt/twiki/bin/view/Research/PCVIA`

[5] Maria J. Pereira,*Concepção, Especificação de uma Linguagem Visual*, Ph.D. thesis, Universidade do Minho, Braga, 1996.

[6] Mario M. Berón, Pedro Henriques, Maria J. Varanda, Roberto Uzal, Germán Montejano. *Language Processing Tool for Program Comprehension.* XII Argentine Congress on Computer Science(2006).

[7] Mario M. Berón, Pedro Henriques, Maria J. Varanda, Roberto Uzal. *Herramientas para la compresión de programas.* VIII Workshop de Investigadores en Ciencias de la Computación (2006).

[8] Mario M. Berón, Pedro R. Henriques, Maria J. Varanda Pereira, Roberto Uzal. *Static and Dynamic Strategies to Understand C Programs by Code Annotation.* European Join Conference on Theory and Practice of Software. Braga-Portugal. 2007.

[9] Wang Yuying, Li Qingshan, Chen Ping, Ren Chunde.*Dynamic Fan-in and Fan-out Metrics for Program Comprehension.* PCODA: Program Comprehension through Dynamic Analysis (2005), 38-42.