Análisis de Información Informal para Facilitar la Comprensión de Programas

Javier Azcurra, Mario Berón, Germán Montejano
Facultad de Ciencias Físico Matemáticas y Naturales - Universidad Nacional de San Luis
San Luis- Argentina
email: javierazcurra_m@yahoo.com.ar,mberon,gmonte@unsl.edu.ar

Pedro Rangel Henriques
Departamento de Informática - Universidade do Minho
Braga-Portugal
email: pedrorangelhenriques@gmail.com

Maria J. Pereira
Departamento de Informática - Instituto Politécnico de Bragança
Bragança - Portugal
email: mjoao@ipb.pt

Abril, 2012

Resumen

La Comprensión de Programas es un área de la Ingeniería de Software cuyo objetivo principal es desarrollar métodos, técnicas y herramientas que faciliten al programador el entendimiento de las funcionalidades de los sistemas de software. Una forma de alcanzar este objetivo consiste en relacionar el Domino del Problema, es decir la salida del sistema, con el dominio del programa, o sea con las partes del programa utilizadas para generar la salida del sistema. La construcción de esta relación representa el principal desafío en el contexto de la Comprensión de Programas. Una solución posible al desafío previamente mencionado consiste en construir una representación para cada dominio y luego vincular ambas representaciones. La representación de ambos dominios se construye en base a la información, estática y dinámica, que se extrae de los mismos. La estrategia de vinculación usa esa información para construir un mapeo entre los elementos de ambos dominios. La información estática se extrae desde el código fuente del sistema usando técnicas de compilación. La información dinámica requiere que el sistema sea modificado sin cambiar su semántica y luego ejecutado.

En este artículo se presenta una línea de investigación que se centra en el estudio, creación e implementación de técnicas de extracción de la información estática desde los sistemas de software. Esta información puede ser estrictamente relacionada con el código del programa, o bien con la información informal provista por los programadores a través de comentarios, literales y documentación.

Palabras clave: Comprensión de Programas, Extracción de Información Estática, Procesamiento de Lenguaje Natural.

1. Contexto

La línea de investigación descrita en este artículo se encuentra enmarcada en el contexto del proyecto: *Ingeniería del Software: Conceptos Métodos Técnicas y Herramientas en un Contexto de Ingeniería de Software en Evolución* de la Universidad Nacional de San Luis. Dicho proyecto, es reconocido por el programa de incentivos y es la continuación de diferentes proyectos de investigación de gran éxito a nivel nacional e internacional.

También se forma parte del proyecto bilateral entre la Universidade do Minho (Portugal) y la Universidad Nacional de San Luis (Argentina) denominado *Quixote: Desarrollo de Modelos del Dominio del Problema para Inter-relacionar las Vistas Comportamental y Operacional de Sistemas de Software*. Quixote[1] fue aprobado por el Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación (MINCyT) y la Fundação para a Ciência e Tecnología (FCT) de Portugal. Ambos entes soportan económicamente la realización de diferentes misiones de investigación desde Argentina a Portugal y viceversa.

2. Introducción

Uno de los principales problemas al que se ven enfrentados los desarrolladores de software es mantener los sistemas en buen funcionamiento [14]. Esta tarea es imposible de llevar a cabo de forma manual debido a que consume muchos costos y esfuerzo humano. Por esta razón, existe una subárea de la Ingeniería de Software que se encuentra dedicada al desarrollo de técnicas de inspección y comprensión de software. Esta área tiene como principal objetivo que el desarrollador logre un entendimiento acabado del software de estudio de forma tal de poder modificarlo disminuyendo en lo posible la gran mayoría de costos [1]. Esta área se conoce en la jerga de la Ingeniería de Software como: *Comprensión de Programas*.

Uno de los principales desafíos en Comprensión de Programas consiste en relacionar dos dominios muy importantes. El primero, el Dominio del Problema, hace referencia a la salida producida por el sistema de estudio. El segundo, el Dominio del Programa, se refiere a las componentes de software utilizadas para producir dicha salida.

Una de los caminos más apropiados para facilitar la comprensión de software consiste en el uso/creación de Herramientas de Comprensión. Una Herramienta de Compresión presenta diferentes perspectivas del software que posibilitan que el ingeniero pueda percibir el funcionamiento del sistema. Para construir herramientas de comprensión, se deben tener en cuenta tres pilares importantes, ellos son: *Interconexión de Dominios*, *Visualización de Software* y *Extracción de la Información* [12, 3].

La *Interconexión de Dominios* [1] tiene como principal objeto de estudio la transformación y vinculación de un dominio específico en otro dominio. Este último dominio puede estar en un alto o bajo nivel de abstracción. El punto importante es que cada componente de un dominio se vea reflejado en una o más componentes del otro y viceversa. A modo de ejemplo, se puede mencionar la transformación de un código fuente (Dominio del Programa) en un Grafo de Llamadas a Funciones (Dominio de Grafos). En este contexto existe una amplia gama de transformaciones siendo la más escasa y difícil de conseguir aquella que relaciona el Dominio del Problema con el Dominio del Programa.

La *Visualización de Software* [1] tiene como finalidad proveer una o varias representaciones visuales del sistema bajo estudio. Para alcanzar este objetivo, se utilizan diferentes técnicas de construcción de vistas o perspectivas de software. Dichas vistas, cuando están bien elaboradas, permiten analizar y percibir la información extraída desde un programa con mayor facilidad.

Por *Extracción de la Información* se entiende el uso/desarrollo de técnicas que permitan extraer información desde el sistema de estudio. Esta información puede ser: Estática o Dinámica, dependiendo de las necesidades del ingeniero de software o del equipo de trabajo.

Para la extracción de la información estática se utilizan técnicas de compilación tradicionales, que se encargan de recuperar información de cada componente del sistema. Todas las actividades que forman parte de esta tarea se realizan desde el código fuente

[1] http://www3.di.uminho.pt/ gepl/

sin ejecutar el sistema. Generalmente, en este tipo de trabajos se construye un analizador sintáctico con las acciones semánticas necesarias para extraer la información requerida. En cambio para la extracción dinámica la información del sistema se obtiene aplicando técnicas de instrumentación de código. La Instrumentación de Código consiste en insertar sentencias dentro del código fuente del sistema con el fin de recuperar las partes del programa que se utilizaron para producir la salida. Está demás decir que, a diferencia de las técnicas de extracción de la información estática, las técnicas de recuperación de la información dinámica requieren que el sistema se ejecute.

Los párrafos precedentes permiten percibir la importancia de las técnicas de extracción de la información. Sin ellas no sería posible la construcción de visualizaciones y técnicas de interconexión de dominios[1]. Por esta razón, en este artículo se describe una línea de investigación que tiene como principal foco de estudio el análisis, la creación y elaboración de técnicas de extracción de la información estática de los sistemas de software. Particularmente, se hace énfasis en la extracción de información estática que facilite la creación de estrategias que permitan relacionar el Dominio del Problema con el Dominio del Programa. Finalmente, es importante mencionar que la información dinámica es tan importante como la información estática, sin embargo su extracción requiere del estudio de otro tipo de aproximaciones que conforman en sí otra línea de investigación.

El artículo está organizado de la siguiente manera. La sección 3 detalla la línea de investigación que se desarrolla en el proyecto. La sección 4 explica sintéticamente los resultados obtenidos/esperados en los temas introducidos previamente. Finalmente, la sección 5 describe la formación de recursos humanos que se utilizan en el proyecto de investigación.

3. Líneas de Investigación y Desarrollo

En CP es fundamental la extracción de información estática y dinámica. La primera es importante porque la información recaudada permite que las herramientas de comprensión detecten errores lexicográficos, sintácticos, semánticos, relaciones entre las componentes del sistemas, entre otras tantas posibilidades [13, 2]. La segunda determina en tiempo de ejecución el comportamiento de los distintos componentes de un programa (variables , métodos, clases, etc).

La información estática puede ser recolectada sin ejecutar el sistema usando técnicas de compilación tradicionales. Dichas técnicas permiten capturar nombres de variables, tipos de las variables, los métodos de un programa orientado a objeto, las variables locales a un método, etc. Actualmente, existen muchas herramientas de comprensión que basan sus análisis en la construcción de estructuras que contienen la información mencionada previamente. Sin embargo, a través del estudio del estado del arte, se pudo detectar que son pocas las estratégias de análisis estático que analizan la información informal que se encuentra disponible en el código fuente, por información informal se entiende aquella contenida en los: *identificadores*, *comentarios de los módulos*, *comentarios de las funciones*, etc. Esto se debe a que dicha información se encuentra expresada en lenguaje natural y por lo tanto su interpretación escapa del análisis estático y requiere de la aplicacion de *Técnicas de Procesamiento de Lenguaje Natural* [4, 5].

Los *Identificadores* generalmente están compuestos por más de una palabra en forma de abreviatura. Los nombres de los identificadores se basan en función de la idiosincrasia del programador y esto representa un problema para el lector del código [10, 7]. Una aproximación a la solución de este problema es tomar los identificadores, aplicarles técnicas de división o separación y luego técnicas de expansión a las abreviaturas para transformar las mismas en palabras completas. Para lograr esto, primero se descompone al identificador en las distintas palabras abreviadas que lo componen, luego se toma cada abreviatura y se expande a la palabra completa. Este no es un proceso sencillo ya que las abreviaturas representan palabras en lenguaje natural, el cual es ambiguo y puede generar controversia en la conversión.

Existen a su vez, otros componentes en el código del programa que ayudan a lograr una mejor comprensión de los identificadores abreviados y del sis-

tema: los comentarios y los literales.

Sin duda, los comentarios tienen como principal finalidad ayudar a entender un segmento de código [8]. Dicho de otra manera, son una fuente importante de información de los conceptos del Dominio del Problema. Por esta razón, se puede ver a los comentarios como una herramienta natural para entender el significado de los identificadores de un código, como así también el funcionamiento del sistema en sí.

Otra manera de entender la semántica de un identificador es analizando los literales o constantes strings. Estos representan un valor constante formado por secuencias de caracteres. Ellos son generalmente utilizados en la muestra de carteles por pantalla, y comúnmente se almacenan en variables de tipo *string* (como es el caso de los programas escritos en Java).

Tanto los literales como los comentarios están escritos en lenguaje natural, por lo tanto es un desafío su correcta interpretación.

Detrás de la información informal se oculta información relevante del Dominio del Problema. Esta información es muy importante porque facilita la reconstrucción de la relación del Dominio del Problema con el Dominio del Programa [11, 6].

4. Resultados y Objetivos

Los resultados obtenidos hasta el momento concernientes al *Análisis de Identificadores* son los siguientes:

- Se investigaron herramientas de construcción de Analizadores Léxicos y Analizadores Sintácticos que emplean la teoría asociada a las gramáticas de atributos. De la investigación mencionada previamente, se determinó que la herramienta $ANTLR$[2] es la más adecuada para extraer eficazmente los identificadores y toda la información relevante asociada a ellos que facilite su análisis.

- Se construyó un analizador sintáctico del lenguaje Java que permite extraer los identificadores, comentarios y literales encontrados en el código fuente del sistema de estudio.

[2]http://www.antlr.org/

- Se implementó una herramienta (Figura 1) que utiliza el analizador mencionado en el ítem previo. La herramienta permite visualizar los atributos (ambiente, tipo de identificador, número de línea,etc) de cada objeto y la parte del código donde se encuentra ubicados. La herramienta fue usada con distintos casos de prueba y se observó que la misma cumplía con las expectativas.

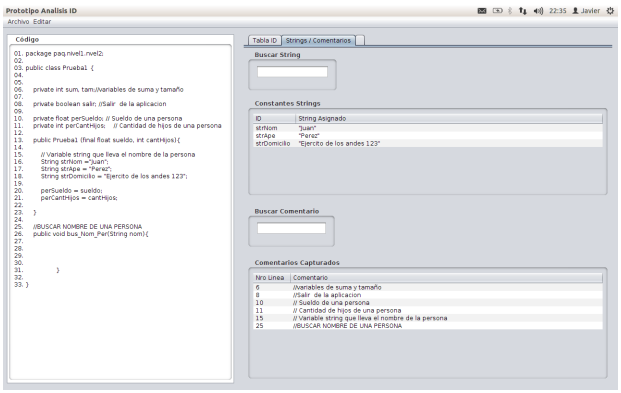

Figura 1: Vista del Sistema

- Se estudiaron las técnicas de división de identificadores Greedy [9] y Samurai [7]. En la primera la partición se basa en controlar la paridad de las abreviaturas con un diccionario de sufijos y prefijos a partir de una lista de abreviaciones conocidas. El segundo usa una función que mide la frecuencia de aparición de palabras en comentarios o literales, las palabras con mayor frecuencia son serias representantes de una abreviatura que forma parte de un identificador.

- Se estudiaron técnicas de expansión de identificadores. Dichas técnicas llevan a disponer de listas de palabras formadas de los comentarios y literales capturados, (como así también una lista de palabras del diccionario en español). La expansión o "reemplazo" de la abreviatura a su correspondiente palabra se realiza cuando la primeras letras son coincidentes y el resto de las letras de la abreviatura se encuentran en la palabra siguiendo el mismo orden [10]. Ejemplo: $fnct \Leftrightarrow function$

Como trabajo futuro a corto plazo se espera:

- Incorporar a la herramienta ya construida las técnicas de división/expansión de identificadores previamente mencionadas [7, 9].
- Agregar a la herramienta técnicas de analisis de comentarios y literales, con el propósito de incrementar conceptos en el código fuente.
- Construir visualizaciones de software basadas en la información obtenida con el análisis de la información informal.

5. Formación de Recursos Humanos

Las tareas llevadas a cabo en la actual línea de investigación son dedicadas a la realización de diferentes tesis correspondiente a la Licenciatura en Ciencias de la Computación. Se proyecta a corto plazo con la continuación de esta investigación la realización de tesis de maestría o bien tesis doctorales. Es importante destacar que el equipo Argentino y el equipo Portugués se ocupan en conjunto a incorporar nuevos integrantes al Grupo de Procesamiento de Lenguajes/Ingeniería Reversa con el fin de fortalecer los vínculos de investigación.

Referencias

[1] Mario M. Berón, Daniel Riesco, and Germán Montejano. Estrategias para facilitar la comprensión de programas. San Luis, Argentina, April 2010.

[2] L. C. Briand. The experimental paradigm in reverse engineering: Role, challenges, and limitations. In *WCRE '06: Proceedings of the 13th Working Conference on Reverse Engineering (WCRE 2006)*, pages 3–8, Washington, DC, USA, 2006. IEEE Computer Society.

[3] R. Brook. Using a behavioral theory of program comprehension in software engineering. *Proceedings of the 3rd international conference on Software engineering*, pages 196–201, 1978.

[4] D. Cruz, P. Henriques, and J. Pinto. Code analysis: Past and present. *Proceeding of the Third International Workshop on Foundations and Techniques for Open Source Software Certification*, 00:1–10, 2009.

[5] T. Eisenbarth, R. Koschke, and D. Simon. Aiding program comprehension by static and dynamic feature analysis. In *ICSM'01: Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, page 602, Washington, DC, USA, 2001.

[6] D.W. Embley. Toward semantic understanding: an approach based on information extraction ontologies. *Proceedings of the fifteenth conference on Australasian database-Volume 27*, pages 3–12, 2004.

[7] E. Enslen, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. 2009.

[8] J. L. Freitas. Comment analysis for program comprehension. OCTUBRE 2011.

[9] D. Lawrie, H. Feild, and D. Binkley. Identifier splitting: A study of two techniques. 2006.

[10] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifier. 2007.

[11] H. Lieberman and C. Fry. Bridging the gulf between code and behavior in programming. In *ACM Conference on Computers and Human Interface*, Denver, Colorado, April 1994.

[12] M. A. Storey, F. D. Fracchia, and Müller. Cognitive design elements to support the construction of a mental model during software exploration.

[13] M. A. Storey, K. Wong, and H. A. Müller. How do program understanding tools affect how programmers understand programs? *Sci. Comput. Program.*, 36(2-3):183–207, 2000.

[14] A. Von Mayrhauser and A. M. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, 1995.