Título: Análisis de Identificadores para Abstraer conceptos del Dominio del Problema.

Autor: Javier Azcurra Marilungo.

Dirección:

- **Director:** Dr. Mario Marcelo Berón.
- **Co-Director:** Dr. Germán Antonio Montejano.

Objetivos: Los objetivos principales de este trabajo final de Licenciatura son:

- Extraer identificadores de programas escritos en lenguaje JAVA.
- Analizar los identificadores extraídos.
- Construir en JAVA una herramienta que implemente los ítems anteriores.
- Evidenciar una aproximación que relacione la salida de un programa con los elementos involucrados que producen dicha salida.

Antecedentes:

La Comprensión de Programas (CP) [7, 6, 5, 33] 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 [37, 29, 9]. La construcción de esta relación representa el principal desafío en el contexto de la CP.

Una solución posible al desafío previamente mencionado consiste en construir una representación para cada dominio, luego se procede a vincular ambas representaciones a través de una estrategia de vinculación. La representación de ambos dominios, se construye en base a la información estática y dinámica que se extrae de los mismos. En la actualidad se conocen distintas técnicas y herramientas que llevan a cabo esta tarea [1, 3].

La información dinámica se extrae del sistema en tiempo de ejecución. Las técnicas encargadas de extraer este tipo de información, insertan sentencias nuevas en el código fuente del programa. Esta inserción no altera la funcionalidad original del programa. La finalidad de las nuevas sentencias es registrar las actividades realizadas durante la ejecución del programa.

Las sentencias nuevas insertadas pueden simplemente imprimir las secciones que se están ejecutando. Más complejo aún, se pueden agregar un conjunto de sentencias nuevas que reemplacen a otro conjunto (sin alterar la funcionalidad original) para hacer un estudio más preciso en ciertas partes del código.

El volumen de información extraída puede ser de gran magnitud. Es por esto, que el encargado de hacer el análisis dinámico debe conocer con claridad los lugares estratégicos donde colocar las sentencias.

La instrumentación de código es una de las técnicas más conocidas del análisis dinámico[3]. Esta técnica, inserta sentencias para crear trazas de ejecución en distintas partes del código. Las trazas clarifican la ejecución del programa mejorando la comprensión del mismo.

El análisis estático es tan importante como el análisis dinámico. El análisis estático a diferencia del dinámico, no requiere ejecutar el sistema.

Para realizar el análisis estático, se extrae información presente en el código utilizando técnicas de compilación tradicionales. Estas técnicas realizan un análisis sintáctico y semántico en el código fuente.

Para llevar a cabo estos análisis se recorre el árbol de sintaxis abstracta (ASA)[1]. En la fase de compilación, el ASA se construye a medida que se analiza el código.

Una forma de construir el ASA es por comprensión[1], esta forma se logra en la etapa del análisis sintáctico durante el proceso de compilación del código. Es la más sencilla y se utiliza en tareas de chequeos de tipos, generación de código, construcción de grafo de funciones, entre otras.

Otra forma de elaborar el ASA es por extensión[1], esta forma se usa para hacer análisis más complejos en el código. También extrae mayor cantidad de información en tiempo de compilación. Algunos ejemplos de tareas que se pueden realizar son, extracción de identificadores, extracción de funciones y tipos y demás objetos presentes en el código.

El ASA por extensión también se usa para resolver conflictos a nivel semántico, como es el caso de borrar referencias a variables redundantes, detectar funciones o estructuras a través del uso de punteros.

El correspondiente trabajo hará más énfasis en la extracción de información estática. Cabe destacar que la dinámica también es importante, sin embargo su análisis requiere de otro tipo de aproximaciones que escapan a los objetivos de este trabajo.

Como bien se explico en párrafos precedentes, se pueden extraer distintos objetos del código fuente de forma estática usando técnicas de compilación, y sin la necesidad de ejecutar el código.

Un elemento que abunda en los códigos son los identificadores. Estudios realizados sobre 2.7 millones de lineas de código escritas en JAVA, indican que más del 70 % de los caracteres del código fuente forman parte de un identificador [28, 8].

Los identificadores normalmente están conformados con abreviaturas en su nombramiento. Estudios realizados [10, 26, 21, 15] arrojan que detrás de las abreviaturas se encuentra oculta información importante del dominio del problema.

Por lo antedicho, construir herramientas que puedan extraer y analizar identificadores desde los códigos JAVA es un gran aporte al área de CP.

Una forma de analizar identificadores consiste en desplegar la información oculta de sus

abreviaturas. Para lograr este cometido, primero se necesita dividir las distintas abreviaturas que el identificador compone. Por ejemplo: inp_fl_sys → inp fl sys. Luego se someten a un proceso de expansión. Por ejemplo: inp fl sys → input file system.

Para realizar los pasos de división y expansión se recurren a fuentes de información informal dentro del código: comentarios, literales strings, documentación. En caso de que estas fuentes escaseen, se pueden utilizar fuentes externas al código como es el caso de los diccionarios de palabras en lenguaje natural.

En la actualidad existen técnicas/herramientas que extraen, dividen y expanden identificadores. Sin embargo, no abundan herramientas que integren todas estas tareas. A su vez, tampoco abundan herramientas que permita utilizar distintas políticas de división y expansión de abreviaturas. De esta forma, se podría variar los resultados obtenidos y determinar el más adecuado.

En base a lo descripto en los párrafos anteriores, en el presente trabajo final de licenciatura se elaborará un analizador sintáctico en lenguaje JAVA que extrae los identificadores, comentarios y literales encontrados en distintos código fuentes escritos en JAVA.

Luego se va investigar técnicas conocidas de división y de expansión de abreviaturas pertenecientes a identificadores, algunas de ellas serán implementadas.

Finalmente, se procederá a implementar una herramienta que integre el analizador y las técnicas de análisis de identificadores nombradas en el párrafo anterior. Esta herramienta mostrará la importancia de la información estática contenida detrás de los identificadores. Extraer esta información permite encarar uno de los desafíos principales de la CP: vincular el Dominio del Problema con el Dominio del Programa.

Metodología:

La metodología a utilizar en este trabajo final, esta enmarcada por 3 etapas principales:

- **Primera Etapa: Investigación de conceptos orientados a la Comprensión de Programas y a la extracción de información estática.**

  En esta etapa se incorporan los conocimientos necesarios para llevar adelante este trabajo final. Se investigará sobre comprensión de programas y las áreas que contiene. De todas estas áreas, se hará más hincapié en la extracción de información estática en base al análisis de identificadores presentes en los códigos. Este análisis, servirá para la elaboración de técnicas de comprensión de sistemas.

- **Segunda Etapa: Realización del analizador sintáctico que extrae identificadores de los códigos escritos en JAVA.**

  Se construirá un analizador sintáctico (parser) encargado de extraer identificadores desde los códigos escritos en JAVA. También, se construirá un analizador lexicográfico (lexer) que extraiga los eventuales comentarios presentes en los códigos. Estos comentarios ayudarán al análisis de identificadores. Para ambas construcciones, se utilizará una herramienta que facilite esta tarea. Se llevarán a cabo las pruebas pertinentes.

- **Tercera Etapa: Construcción de la herramienta que analiza identificadores e integra los analizadores realizados en la etapa previa.**

  Finalmente, en esta etapa se construye una herramienta que contendrá el lexer y el parser elaborado en la etapa anterior. A su vez, implementa algunas técnicas de análisis de identificadores estudiadas en la primer etapa. La finalidad de esta herramienta es extraer información estática en base a los identificadores analizados. Esta información extraída, se considera del Dominio del Problema y es un aporte a la reconstrucción de la relación entre: el Dominio del Problema y el Dominio del Programa.

Plan y cronograma de tareas:

1. Estudiar conceptos, técnicas y herramientas sobre la comprensión de programas: con esto se lográ conocer las características relacionadas al ámbito de la comprensión de sistemas.
2. Investigar conceptos, técnicas y herramientas relacionados al análisis de identificadores: aquí se adquieren los conceptos necesarios para extraer información del Dominio del Problema, en base a los identificadores.
3. Investigar herramientas que puedan generar un Analizador Lexicográfico (lexer) y un Analizador Sintáctico (parser). Preferentemente se escogerán aquellas que empleen gramáticas de atributos.
4. Estudiar la gramática de JAVA para construir el Analizador Lexicográfico (lexer) y Analizador Sintáctico (parser), el primero va extraer comentarios y el segundo identificadores.
5. Implementar una herramienta en JAVA que contenga distintas técnicas de análisis de identificadores investigados en el ítem 2. Esta herramienta también contendrá el lexer y el parser de la etapa anterior.
6. Escribir el informe del trabajo final. Mientras se ejecutan los ítems precedentes, se irá redactando el informe correspondiente al trabajo final.

Cronograma									
Actividad		Meses							
		1	2	3	4	5	6	7	8
1	Estudiar conceptos, técnicas y herramientas sobre la comprensión de programas.	X							
2	Investigar conceptos, técnicas y herramientas relacionados al análisis de identificadores.	X	X	X					
3	Estudiar herramientas que generen Analizadores Lexicográficos y Analizadores Sintácticos.				X				
4	Construir el Analizador Lexicográfico (lexer) y Analizador Sintáctico (parser).				X				
5	Construir una herramienta que contenga el lexer y el parser de la etapa anterior e implemente distintas técnicas de análisis de identificadores.					X	X	X	X
6	Escribir el informe del trabajo final.		X	X	X	X	X	X	X

Recursos:

El correspondiente trabajo final se llevará a cabo en la Universidad Nacional de San Luis, en el Área Programación y Metodologías de Desarrollo de Software enmarcado en los siguientes proyectos:

- *"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.
- "Quixote - Development of Problem Domain Models to Interconnect the Behavioral and Operational Views to Aid in Software Systems Comprehension" (código de proyecto: PO/09/38). Quixote es un proyecto bilateral entre la Universidade do Minho (Portugal) y la Universidad Nacional de San Luis (Argentina). Dicho proyecto fue aprobado por el Ministerio de Ciencia, Tecnología e Innovación Productiva de la Nación (MinCyT)[1], y la Fundação para a Ciência e Tecnología (FCT)[2] de Portugal.

Ambos entes soportan económicamente la realización de diferentes misiones de investigación desde Argentina a Portugal y viceversa, como así también la presentación y publicación de artículos en diferentes congresos nacionales e internacionales.

Por otro lado, el equipamiento necesario es una PC con el sistema Linux, impresora y papel para realizar las impresiones del informe. También se hará uso de internet, de material bibliográfico provisto por la biblioteca "Antonio Esteban Agüero"; y de material provisto por las librerías digitales a las que se tiene acceso desde la Universidad.

[1] www.mincyt.gov.ar/
[2] www.fct.mctes.pt/

Bibliografía

[1] A. V. Aho, R. Sethi, and J. D. Ullman. Compilers principles, techniques and tools. 2006.

[2] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *Estructuras de datos y algoritmos*. Sistemas Tecnicos de Edicion, 1988.

[3] Thoms Bell. The concept of dynamic analysis. 1999.

[4] Keith H. Bennett and Vaclav T. Rajlich. Software maintenance and evolution: a roadmap. 2000.

[5] M. Beron, P. Henriques, and R. Uzal. *Inspección de Programas para Interconectar las Vistas Comportamental y Operacional para la Comprensión de Programas.* PhD thesis, Universidade do Minho, Braga. Portugal, 2010.

[6] Mario Beron, Pedro R. Henriques, Maria J. Varanda, and Roberto Uzal. Program inspection to inter-connect the operational and behavioral views for program comprehension. 2007.

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

[8] Dave Binkley, Marcia Davis, Dawn Lawrie, and Bonita Sharif. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 2013.

[9] R. Brook. A theoretical analysis of the role of documentation in the comprehension of computer programs. *Proceedings of the 1982 conference on Human factors in computing systems*, pages 125–129, 1982.

[10] Bruno Caprile and Paolo Tonella. Nomen est omen: Analyzing the language of function identifiers. IEEE Computer Society, 1999.

[11] Bruno Caprile and Paolo Tonella. Restructuring program identifier names. IEEE Computer Society, 2000.

[12] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding program comprehension by static and dynamic feature analysis. page 602, 2001.

[13] Ramez A. Elmasri and Shankrant B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 3rd edition, 1999.

[14] David W. Embley. Toward semantic understanding: an approach based on information extraction ontologies. Australian Computer Society, Inc. Darlinghurst, Australia, Australia, 2004.

[15] Eric Enslen, Emily Hill, Lori Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. 2009.

[16] Henry Feild, David Binkley, and Dawn Lawrie. An empirical comparison of techniques for extracting concept abbreviations from identifiers. In *In Proceedings of IASTED International Conference on Software Engineering and Applications (SEA 2006)*, 2006.

[17] Henry Feild, David Binkley, and Dawn Lawrie. Identifier splitting: A study of two techniques. April 2006.

[18] Fangfang Feng and W. Bruce Croft. Probabilistic techniques for phrase extraction. *Inf. Process. Manage.*, 37:199–220, 2001.

[19] José Luís Freitas, Daniela da Cruz, and Pedro Rangel Henriques. The role of comments on program comprehension. 2008.

[20] Wilhelm Hasselbring, Andreas Fuhr, and Volker Riediger. First international workshop on model-driven software migration. Marzo 2011.

[21] Emily Hill, Zachary P. Fry, and Haley Boyd. Automatically mining abbreviation expansions in programs to enhance software maintenance tools. 2008.

[22] IEEE. *IEEE Standard for Software Maintenance, IEEE Std 1219-1998*, volume 2. IEEE Press, 1999.

[23] IEEE. *Standard Glossary of Software Engineering Terminology 610.12-1990*, volume 1. IEEE Press, 1999.

[24] Dawn Lawrie, Henry Feild, and David Binkley. Quantifying identifier quality: an analysis of trends. Kluwer Academic Publishers-Plenum Publishers., 2006.

[25] Dawn Lawrie, Henry Feild, and David Binkley. Syntactic identifier conciseness and consistency. 2006.

[26] Dawn Lawrie, Henry Feild, and David Binkley. Extracting meaning from abbreviated identifier. 2007.

[27] Dawn Lawrie, Christopher Morrell, Henry Feild, and David Binkley. What's in a name? a study of identifiers. 2006.

[28] Pizka M and DeiBenbock F. Concise and consistent naming. *Software Quality Control*, 2005.

[29] Michael P. O'Brien. Software comprehension – a review and research direction. 2003.

[30] Roger S. Pressman. *Ingeniería del Software un enfoque práctico*. 2002.

[31] T.A. Standish. *Data structure techniques*. Computer Sciences. Addison-Wesley, 1980.

[32] M. A. Storey, F. D. Fracchia, and H. A. Müller. Cognitive design elements to support the construction of a mental model during software exploration. *The Journal of Systems & Software*, 44(3):171–185, 1999.

[33] Margaret Anne Storey. Theories, methods and tools in program comprehension: Past, present and future. 2005.

[34] Pierre F. Tiako. Maintenance in joint software development. 2002.

[35] Tim Tiemens. Cognitive models of program comprehension. 1989.

[36] Marco Torchiano and Massimiliano Di Penta. Software migration projects in italian industry: Preliminary results from a state of the practice survey. 2007.

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

Firma de alumno y asesores