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.
- Extraer comentarios y literales de programas escritos en lenguaje JAVA.
- Analizar los identificadores capturados con ayuda de la información extraída en el ítem anterior.
- 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) [9, 8, 7, 36] 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 [41, 33, 11, 3]. La construcción de esta relación representa el principal desafío en el contexto de la CP.

Una posible solución 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. Para construir la representación del dominio del problema y la del dominio del programa, se necesita extraer información estática y dinámica propia de ambos dominios. En la actualidad se conocen distintas técnicas y herramientas que llevan a cabo esta tarea [1, 5].

La información dinámica se extrae del sistema en tiempo de ejecución [5, 15]. 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 [5]. 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 [15, 1].

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 estático 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 [14, 10].

Los identificadores normalmente están conformados con abreviaturas en su nombramiento. Estudios realizados [12, 28, 24, 18] 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 palabras/frases dentro del código: comentarios, literales strings, documentación [22]. En caso de que estas fuentes escaseen, una alternativa viable es consultar fuentes de palabras externas al código como es el caso de los diccionarios con palabras en lenguaje natural, entre otros.

En la actualidad existen técnicas/herramientas que extraen, dividen y expanden identificadores [24, 18, 13, 20]. Sin embargo, no abundan herramientas que integren todas estas tareas. A su vez, tampoco existen muchas 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 códigos fuentes escritos en lenguaje JAVA.

Luego se va investigar técnicas conocidas de división y de expansión de abreviaturas pertenecientes a identificadores. Finalmente, se procederá a implementar una herramienta que integre las técnicas de análisis de identificadores antedichas junto al analizador sintáctico explicado en el párrafo anterior. De esta manera, se obtendrá una herramienta que extrae y analiza información estática contenida detrás de los identificadores. Extraer y analizar este tipo de información propia del dominio del problema [14, 10, 18] es un aporte importante para 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: Elaboración del analizador sintáctico que extrae identificadores de los códigos escritos en JAVA.**

  Se construirá un analizador sintáctico (parser) encargado de extraer identificadores, comentarios y literales desde los códigos escritos en JAVA. Estos comentarios y literales ayudarán al análisis de identificadores. Para construir este parser, se investigarán herramientas que ayude a la construcción y se elegirá la más adecuada. Se llevarán a cabo las pruebas pertinentes.

- **Tercera Etapa: Construcción de la herramienta que analiza identificadores e integra el parser elaborado en la etapa previa.**

  Finalmente, en la última etapa se construye una herramienta que contendrá al analizador sintáctico elaborado en la etapa anterior y también implementa algunas de las técnicas de análisis de identificadores estudiadas en la primer etapa. La finalidad de esta herramienta es extraer y analizar información presente en identificadores ubicados en códigos JAVA. Esta información extraída y analizada, se considera propia 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 ayuden a construir un Analizador Sintáctico (parser). Preferentemente se escogerán aquellas que empleen gramáticas de atributos [1].
4. Estudiar la gramática de JAVA e implementar el Analizador Sintáctico (parser) utilizando alguna herramienta investigada del item anterior.
5. Implementar una herramienta en JAVA que contenga distintas técnicas de análisis de identificadores investigados en el ítem 2. Esta herramienta también contendrá el parser elaborado en 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 ayuden a construir Analizadores Sintácticos.				X				
4	Estudiar la gramática de JAVA y construir el Analizador Sintáctico (parser).				X				
5	Construir una herramienta que contenga el parser de la etapa anterior e implemente distintas técnicas de análisis de identificadores vistas en el ítem 1.					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] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition).* Addison Wesley, August 2006.

[2] Alfred V. Aho, Jeffrey D. Ullman, and John E. Hopcroft. *Data structures and algorithms / Alfred V. Aho, John E. Hopcroft, Jeffrey D. Ullman.* Addison-Wesley Reading, Mass, 1983.

[3] José Luís Albanes, Mario Berón, Pedro Henriques, and Maria João Pereira. Estrategias para relacionar el dominio del problema con el dominio del programa para la comprensión de programas. *XIII Workshop de Investigadores en Ciencias de la Computación*, pages 449–453, 2011.

[4] G. Antoniol, G. Canfora, G. Casazza, A. De Lucia, and E. Merlo. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on*, 28(10):970–983, 2002.

[5] Thomas Ball. The concept of dynamic analysis. In Oscar Nierstrasz and Michel Lemoine, editors, *ESEC / SIGSOFT FSE*, volume 1687 of *Lecture Notes in Computer Science*, page 216–234. Springer, 1999.

[6] K. Bennett and V. Rajlich. Software maintenance and evolution: a roadmap. In *ICSE - Future of SE Track*, page 73–87, 2000.

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

[8] Mario Berón, Pedro Henriques, Maria João Pereira, and Roberto Uzal. Program inspection to interconnect behavioral and operational view for program comprehension. University of York, 2007.

[9] Mario Berón, Daniel Eduardo Riesco, Germán Antonio Montejano, Pedro Rangel Henriques, and Maria J Pereira. Estrategias para facilitar la comprensión de programas. In *XII Workshop de Investigadores en Ciencias de la Computación*, 2010.

[10] Dave Binkley, Marcia Davis, Dawn Lawrie, Jonathan Maletic, Christopher Morrell, and Bonita Sharif. The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, 18(2):219–276, 2013. (early access).

[11] R. Brook. A theoretical analysis of the role of documentation in the comprehensionof computer programs. *Proceedings of the 1982 conference on Human factors in computingsystems.*, pages 125–129, 1982.

[12] Bruno Caprile and Paolo Tonella. Nomen est omen: analyzing the language of function identifiers. In *Proc. Sixth Working Conf. on Reverse Engineering*, page 112–122. IEEE, October 1999.

[13] Bruno Caprile and Paolo Tonella. Restructuring program identifier names. In *Proc. Int'l Conf. on Software Maintenance*, page 97–107. IEEE, 2000.

[14] Florian DeiBenbock and Markus Pizka. Concise and consistent naming. In *IWPC*, page 97–106. IEEE Computer Society, 2005.

[15] Thomas Eisenbarth, Rainer Koschke, and Daniel Simon. Aiding program comprehension by static and dynamic feature analysis. In *ICSM*, pages 602–611, 2001.

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

[17] David W Embley. Toward semantic understanding: an approach based on information extraction ontologies. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 3–12. Australian Computer Society, Inc., 2004.

[18] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *6th IEEE International Working Conference on Mining Software Repositories.*, page 71–80. IEEE, may. 2009.

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

[20] Henry Feild, David Binkley, and Dawn Lawrie. Identifier splitting: A study of two techniques. In *Proceedings of the Mid-Atlantic Student Workshop on Programming Languages and Systems.*, pages 154–160, 2006.

[21] Fangfang Feng and W. Bruce Croft. Probabilistic techniques for phrase extraction. *Inf. Process. Manage.*, 37(2):199–220, 2001.

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

[23] Wilhelm Hasselbring, Andreas Fuhr, and Volker Riediger. First international workshop on model-driven software migration (mdsm 2011). In *CSMR '11 Proceedings of the 2011 15th European Proceedings of the 2011 15th European Conference on Software Maintenance and Reengineering (CSMR 2011)*, pages 299–300, Washington, DC, USA, März 2011. IEEE Computer Society.

[24] Emily Hill, Zachary P. Fry, Haley Boyd, Giriprasad Sridhara, Yana Novikova, Lori Pollock, and K. Vijay-Shanker. Amap: Automatically mining abbreviation expansions in programs to enhance software maintenance tools. In *Proceedings of the 5th Int'l Working Conf. on Mining Software Repositories.*, page 79–88. ACM, 2008.

[25] IEEE. Ieee standard for software maintenance. *IEEE Std 1219-1998*, page i–, 1998.

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

[27] Leah S. Larkey, Paul Ogilvie, M. Andrew Price, and Brenden Tamilio. Acrophile: an automated acronym extractor and server. In *ACM DL*, page 205–214, 2000.

[28] D. Lawrie, H. Feild, and D. Binkley. Extracting meaning from abbreviated identifiers. In *Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on*, page 213–222, Sept 2007.

[29] D. Lawrie, C. Morrell, H. Feild, and D. Binkley. What's in a name? a study of identifiers. In *Program Comprehension, 2006. ICPC 2006. 14th IEEE International Conference on.*, page 3–12, June 2006.

[30] Dawn Lawrie, Henry Feild, and David Binkley. Syntactic identifier conciseness and consistency. In *SCAM*, page 139–148. IEEE Computer Society, 2006.

[31] Dawn Lawrie, Henry Feild, and David Binkley. An empirical study of rules for well-formed identifiers. *Journal of Software Maintenance and Evolution: Research and Practice*, 19(4):205–229, 2007.

[32] Dawn Lawrie, Henry Feild, and David Binkley. Quantifying identifier quality: an analysis of trends. *Empirical Software Engineering*, 12(4):359–388, 2007.

[33] Michael P O'brien. Software comprehension–a review & research direction. *Department of Computer Science & Information Systems University of Limerick, Ireland, Technical Report.*, 2003.

[34] Roger S. Pressman. *Software Engineering - A Practitioner's Approach.* McGraw-Hill, 5 edition, 2001.

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

[36] M. Storey. Theories, methods and tools in program comprehension: past, present and future. In *Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on.*, page 181–191, May 2005.

[37] 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.

[38] P.F. Tiako. Maintenance in joint software development. In *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International.*, page 1077–1080, 2002.

[39] Tim Tiemens. Cognitive models of program comprehension, 1989.

[40] Marco Torchiano, Massimiliano Di Penta, Filippo Ricca, Andrea De Lucia, and Filippo Lanubile. Software migration projects in italian industry: Preliminary results from a state of the practice survey. In *ASE Workshops.*, page 35–42. IEEE, 2008.

[41] A von Mayrhauser and AM. Vans. Program comprehension during software maintenance and evolution. *Computer*, 28(8):44–55, Aug 1995.

Firma de alumno y asesores