Capítulo 4

Identifier Analyzer (IDA)

4.1. Introducción

En el capítulo anterior, se explicó la importancia de analizar ids ubicados en los códigos. Los ids ocultan información que es propia del Dominio del Problema (ver capítulo 3). Desafortunadamente, las personas ajenas al código, no comprenden a simple vista la información que los ids poseen e invierten tiempo en entender su presencia en el código. Es por esto, que las herramientas automáticas de análisis de ids son bienvenidas en el ámbito de la Comprensión de Programas (CP). Con estas herramientas se logra achicar los tiempos de comprensión de ids y revelar la información poco visible que estos contienen.

Dada la importancia que tienen las herramientas de análisis de ids, se tomó la iniciativa de desarrollar una llamada Identifier Analyzer (IDA). Esta herramienta le permite al usuario ingresar un archivo JAVA, luego IDA analiza los ids que están en el archivo, y finalmente muestra como resultado una tabla del análisis realizado. El objetivo de IDA es lograr que el usuario comprenda más rápidamente el propósito de los ids en los archivos JAVA, y de esta manera mejorar la comprensión del código analizado.

El correspondiente capítulo, está destinado a explicar los distintos módulos que IDA posee, y que proceso de ejecución debe realizar el usuario para analizar los ids. Al final de este capítulo, se describen algunos casos de es-

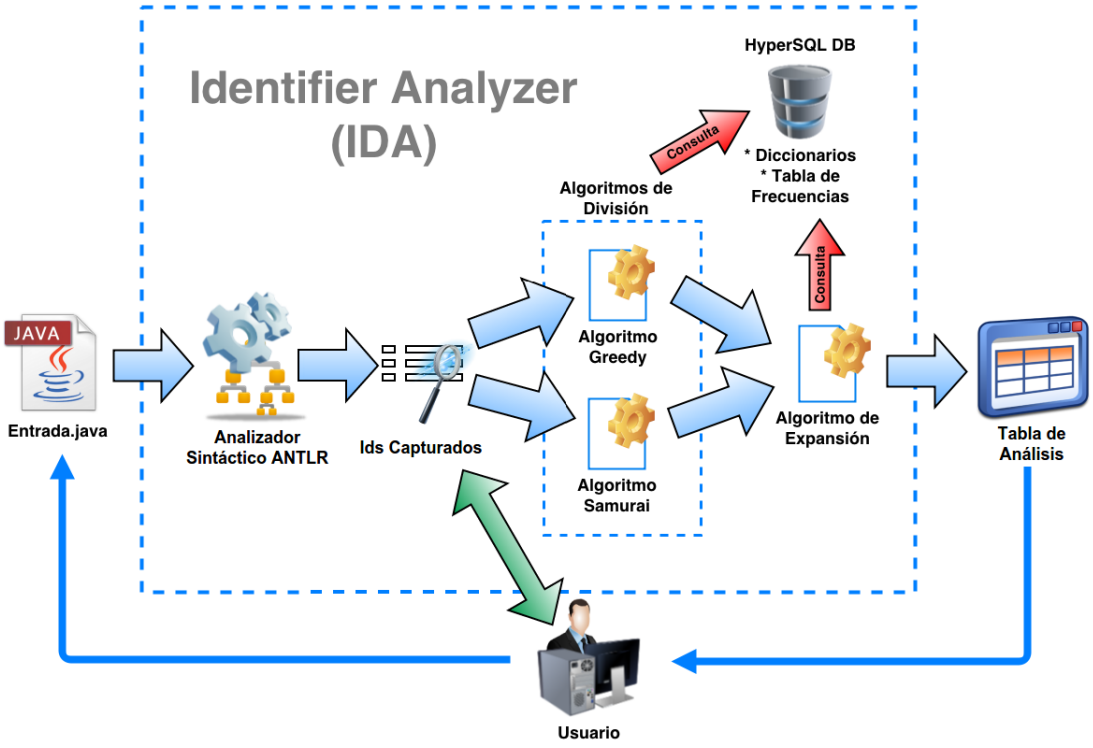

Figura 4.1: Arquitectura de IDA.

tudio que demuestran la importancia de haber construido IDA. Todas estas explicaciones, están acompañadas con capturas de pantalla y tablas que facilitarán al lector entender el funcionamiento de la herramienta IDA. Para comenzar con la descripción de IDA, en la siguiente sección se explica la arquitectura y cuales son sus componentes principales.

4.2. Arquitectura

En la figura 4.1 se puede apreciar la arquitectura de IDA. Esta arquitectura describe tres partes principales, la primera consiste en la *extracción de datos*, la segunda trata sobre la *división de ids* y la tercera sobre *expansión de ids*. A continuación se detallan cada una de ellas.

Módulo de Extracción de Datos: Este módulo recibe como entrada un archivo JAVA que ingresa el usuario (ver Figura 4.1 Entrada.java), luego

este archivo se procesa por un Analizador Sintáctico (AS) (ver Figura 4.1 Analizado Sintáctico ANTLR). El AS, extrae y almacena en estructuras internas la información estática perteneciente al código del archivo ingresado. Esta información extraída, está relacionada con ids, literales y comentarios (ver próxima sección para más detalles). El usuario a través de la interfaz de IDA, puede visualizar esta información capturada del código por medio de tablas claramente definidas (ver figura 4.1 - Flecha Verde).

Módulo de División de Ids: Una vez completada la extracción de información, el proceso continua en el módulo de división de ids. Aquí se encuentran implementados 2 algoritmos de división; uno es el Algoritmo Greedy y el otro es el Algoritmo Samurai ambos explicados en el capítulo anterior. Estos algoritmos reciben como entrada la información capturada en el módulo de extracción de datos (ids, comentarios, literales), luego estos algoritmos se encargan de dividir los ids del archivo JAVA (ver figura 4.1 Algoritmos de División). Los resultados de las divisiones se almacenan en estructuras internas que serán utilizadas en próximo módulo. Cabe recordar que estos algoritmos de división necesitan datos externos para funcionar, uno es el diccionario de palabras (en caso de Greedy) y el otro es lista de frecuencias globales de aparición de palabras (en caso de Samurai). Estos datos externos se encuentran almacenados en una base de datos embebida (parte superior de la figura 4.1 - Flechas Rojas).

Módulo de Expansión de Ids: La tercera y última parte, tiene implementado el Algoritmo de Expansión Básico de abreviaturas que fue explicado en el capítulo anterior. Este algoritmo toma como entrada los ids divididos en la etapa anterior, tanto de Greedy como de Samurai. Luego, el Algoritmo de Expansión procede a expandir los ids obtenidos por estos 2 algoritmos, dando como resultado ids expandidos desde Greedy y desde Samurai (ver figura 4.1 Algoritmo de Expansión). El Algoritmo de Expansión también necesita de un diccionario de palabras, por eso se realizan consultas a la base de datos embebida (parte superior de la figura 4.1 - Flechas Rojas). Finalmente, los resultados de las divisiones y las expansiones de los ids, son exhibidas en una tabla (ver figura 4.1 Tabla de análisis).

4.3. Analizador Sintáctico

Como se explicó en la sección previa, cuando el usuario ingresa un archivo JAVA, IDA utiliza un Analizador Sintáctico (AS) que examina y extrae información estática presente en el archivo ingresado. Esta información está compuesta por identificadores, comentarios y literales.

La construcción de este AS se llevó a cabo, primero investigando herramientas encargadas de construir AS. Se dio preferencia a aquellas que emplean la teoría asociada a las gramáticas de atributos [1]. De la investigación previamente descripta, se determinó que la herramienta *ANTLR*[1] era la que mejor se ajustaba a las necesidades antes planteadas. Esta herramienta permite agregar acciones semánticas (escritas en JAVA) para el calculo de los atributos, en una gramática de lenguaje JAVA[2]. Estas acciones semánticas deben estar correctamente insertadas en la gramática para, por ejemplo, implementar estructuras de datos y algoritmos que capturan los ids utilizados en un programa [2]. Una vez insertadas estas acciones, ANTLR se encarga de leer la gramática y generar el AS adicionando acciones que fueron programadas. De esta manera, se obtiene un AS que recolecta ids mientras examina el código. A su vez a estas acciones semanticas, se le agregan otras acciones que extraen comentarios y literales strings. Estos elementos son necesarios ya que sirven para los algoritmos de análisis de ids que serán explicados en próximas secciones.

4.4. Base de Datos Embebida

Dado que IDA necesita de diccionarios y listas de palabras para poder llevar a cabo sus tareas, los mismos están almacenadas dentro de IDA en una base de datos embebida. Esta base de datos utiliza una tecnología llamada HSQLDB[3] y al estar desarrollada en JAVA permite una correcta integración con IDA. Otra ventaja que tiene esta tecnología es que responde rápidamente

[1] ANother Tool for Language Recognition. http://www.antlr.org
[2] Especificaciones de la gramática JAVA en http://docs.oracle.com/javase/specs/jls/se7/jls7.pdf
[3] Hyper SQL Data Base. http://www.hsqldb.org

las consultas.

A continuación, se describen los diccionarios/listas de palabras que están alojadas en la base de datos explicada en el párrafo anterior. También se nombran los algoritmos implementados en IDA que consultan cada una.

Diccionario en Inglés (ispell): Contiene palabras en Inglés que pertenecen a la lista de Palabras de *Ispell*[1]. Se utiliza en el Algoritmo de Greedy y en el Algoritmo de Expansión (ver capítulo 3).

Lista de Palabras Excluyentes (stop-list): Esta compuesta con palabras que son poco importantes o irrelevantes en el análisis de ids. Se utiliza en el Algoritmo de Greedy y en el Algoritmo de Expansión (ver capítulo 3).

Lista de Abreviaturas y Acrónimos Conocidas: Contiene abreviaturas comunes del idioma Inglés y Acrónimos conocidos de programación (gif, jpg, txt). Se emplea en el Algoritmo Greedy (ver capítulo 3).

Lista de Prefijos y Sufijos Conocidos: Posee Sufijos y Prefijos conocidos en Inglés, esta lista fue confeccionada por el autor del Algoritmo Samurai (ver capítulo 3). Se consulta solo en dicho algoritmo.

Frecuencias Globales de Palabras: Lista de palabras, junto con su frecuencia de aparición. Esta lista fue construida por el autor del Algoritmo Samurai. Se emplea solo en dicho algoritmo, más precisamente en la función de *Scoring* (ver capítulo 3).

Cabe destacar que las listas y diccionarios que fueron descriptos poseen palabras que pertenecen al idioma Inglés, dado que los autores así lo determinaron. Por lo tanto, para que la herramienta IDA analice correctamente los ids, se recomienda ingresar en IDA archivos JAVA con comentarios, literales e ids acordes a la lengua Inglesa.

Habiendo descripto los principales módulos de la herramienta, en la próxima sección se explicará el proceso que debe seguir el usuario para analizar ids a través de IDA.

[1]Comando de Linux. http://wordlist.aspell.net

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] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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).

[9] 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.

[10] 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.

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

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

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

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

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

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

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

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

[32] 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.

[33] 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.

[34] 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.

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

[36] 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.

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