Capítulo 5

Conclusiones y Trabajos Futuros

5.1. Conclusiones

En este último capítulo, se describen las conclusiones obtenidas luego de la realización de este trabajo final, más adelante se proponen trabajos futuros a realizar.

Las conclusiones que se llevaron a cabo, están relacionadas a:

- La Investigación sobre el Análisis de Identificadores.
- La Construcción de la Herramienta Identifier Analyzer (IDA).
- Los Casos de Estudio probados en IDA.

A continuación, se desarrollan cada uno de estos ítems.

5.1.1. La Investigación sobre el Análisis de Identificadores

Luego de haber realizado un extenso estudio sobre la temática de análisis de identificadores (el cual fue descripto en el capítulo 3), se arribaron a las conclusiones que se describen a continuación.

Si bien existen diferentes técnicas de división de identificadores (GentTest, Dynamic Time Warping, Identifier Name Tokeniser Tool, entre otros) los algoritmos más populares son Greedy y Samurai. El primero porque es sencillo de implementar, y además es muy utilizado como medida de comparación con técnicas más avanzadas [**?**, **?**, **?**]. El segundo porque es un algoritmo bastante automatizado y según el autor [**?**] tiene un mejor desempeño que la mayoría de las técnicas conocidas que separan ids.

En lo que respecta a estrategias de expansión de ids, la técnica básica de expansión es muy sencilla y trae como consecuencias expansiones incorrectas. Esto ocurre debido a que las abreviaturas con pocas letras, generalmente se expanden por medio de diccionarios con palabras en lenguaje natural. Estos diccionarios, normalmente contienen amplias variedades de palabras que no están muy relacionadas al dominio del problema y tampoco a las ciencias de la computación. Siguiendo con las técnicas investigadas, el algoritmo de expansión Automatically Mining Abbreviation Expansions in Programs (AMAP) [**?**] contiene mejoras notables con respecto al algoritmo de Expansión Básico. Con la estrategia AMAP, se expanden abreviaturas de manera más precisa y sin utilizar amplios diccionarios, lo que conlleva a tener resultados más efectivos.

Algunas técnicas de análisis de identificadores emplean diccionarios/listas para realizar su trabajo. Las más comunes son las que primero se construyeron, este es el caso de Identifier Restructurer, el algoritmo Greedy y el algoritmo de Expansión Básico (ver capítulo 3). Los mismos utilizan diccionarios/listas extensos lo que conlleva a un gasto alto en espacio y a una precisión baja de aciertos en sus resultados. A medida que se avanzó en la investigación, se pudo comprobar la existencia de técnicas más nuevas como Samurai y AMAP que disminuyen el espacio utilizado y aumentan la efectividad de sus tareas. La forma en que consiguen esto, es explotando más los recursos propios del sistema, y no apoyarse tanto en recursos externos. Esto a su vez permite, que las técnicas Samurai y AMAP sean más escalables a nuevas tecnologías y a nuevos vocablos en el ámbito de la ciencias de la computación.

5.1.2. La Construcción de la Herramienta IDA

Luego de haber construido la herramienta Identifier Analyzer (IDA) (que fue descripta en el capítulo 4), se puede concluir que, implementar técnicas de análisis de identificadores requiere de la aplicación de diferentes conocimientos allegados a áreas dentro de las ciencias de la computación, tales como:

Lenguajes de Programación: Se utilizaron conceptos de lenguajes de programación, relacionados a la sintaxis y la semántica. Se hizo hincapié en el lenguaje JAVA, y de como los nombres de los ids impactan en la comprensión de los sistemas.

Base de Datos: Se adquirieron cocimientos para seleccionar un motor de base de datos adecuado. El mismo, debe mantener y gestionar los diccionarios/listados de palabras para que las técnicas que utiliza IDA, funcionen lo más eficientemente posible.

Sistemas Operativos: Dado que IDA emplea un programa externo para embellecer el código de entrada, se necesitó investigar como realizar las llamadas a programas por línea de comandos, según el Sistema Operativo que se utilice.

Ingeniería del Software: Se implementaron técnicas de ingeniería inversa, IDA recibe un código de entrada, y retorna una tabla descriptiva con los ids analizados, para lograrlo se adquirieron conocimientos sobre técnicas para construir un Analizador Sintáctico, con el objeto de capturar ids, comentarios y literales.

La selección del lenguaje Java para la implementación de la herramienta IDA resulto apropiada porque dicho lenguaje posee un conjunto amplio de librerías útiles. Algunas de las librerías escritas en JAVA, que se incorporaron en IDA son: ANTLR[1], OpenCloud[2], HSQLDB[3].

[1] ANother Tool for Language Recognition. http://www.antlr.org
[2] http://opencloud.mcavallo.org
[3] Hyper SQL Data Base. http://www.hsqldb.org

La interfaz gráfica de IDA fue construida de manera tal de que sea simple de usar. Para lograr tal objetivo fue necesario la implementación de estrategias de interacción hombre-máquina, y también, de la aplicación de técnicas de visualización tanto textuales, como gráficas. Las estrategias antes mencionadas posibilitaron que la herramienta IDA tenga un cuadro con el código leído desde el archivo, este código se resalta con color para destacar los distintos elementos que lo componen, en sintonía con la interacción con el usuario. Además se requirió llevar a cabo, una correcta distribución de las tablas, ventanas, botones, menús y todos los componentes visuales que la herramienta IDA tiene (ver capítulo 4).

5.1.3. Los Casos de Estudio probados en IDA

Los casos de estudios presentados en este trabajo final, son programas que han sido implementados por otros programadores y por lo tanto son susceptibles a tareas de comprensión. En el caso que se requieran hacer futuras modificaciones, se sabe como funcionan.

Los resultados que arrojaron la ejecución de estos casos de estudio en IDA, indican que la técnica Samurai fue más efectiva que Greedy. Una causa de esto, se debe a que Greedy es una técnica que siempre tiende a dividir a los ids en mayor proporción que lo deseado, y esto se reflejó en los resultados de los casos de estudio. La mayoría de los resultados incorrectos de Greedy fueron causados por el exceso de separaciones. Otro aspecto que influyó en los resultados, es que Greedy basa sus criterios de división, solo en la información provista por diccionarios/listados de palabras predefinidos, sin tener en cuenta la información provista en el código (comentarios y literales), que es propia del dominio del problema.

Las expansiones de las abreviaturas de los ids, si se observan los casos fallidos de expansión, los mismos fueron causados por dos motivos principales: cuando la división previa no se efectuó correctamente, y cuando no se encontraron palabras candidatas completas dentro de los comentarios y literales ubicados en el código del programa. En muy pocos casos hubo acierto en la expansión, cuando la fuente de búsqueda era el diccionario de pala-

bras en lenguaje natural. Esto apoya la teoría de que estos diccionarios son imprecisos cuando se trata de analizar ids.

A través de los casos de estudio, se pretendió mostrar que IDA es útil a la hora de comprender un sistema por medio del análisis de los ids. Como se describió en capítulos anteriores, el principal objetivo de la CP es relacionar el Dominio del Programa y el Dominio del Problema. En los tres casos de estudio presentados en el capítulo anterior, la información obtenida producto de las expansiones acertadas de los ids, revela que es propia del domino del problema. Por ende, IDA sirve como aporte a la CP y como punto de partida para desarrollar nuevas herramientas, que faciliten el entendimiento de los ids en los códigos de los sistemas de software.

Habiendo descripto las conclusiones pertinentes, en la próxima sección se proponen algunos trabajos futuros a realizar.

5.2. Trabajos Futuros

En esta sección se describen propuestas vinculadas a trabajos futuros de la herramienta IDA. Se tomará como punto de partida el actual estado de desarrollo de IDA y en función de este estado, se proponen mejoras y/o expansiones. Los trabajos futuros propuestos son:

- Ampliar la Captura del Analizador Sintáctico.
- Implementar otro Algoritmo de Expansión.
- Expandir Identificadores en el Código.
- Acoplar a Entornos de Desarrollos.

A continuación, se describen cada uno de ellos.

5.2.1. Ampliar la Captura del Analizador Sintáctico

El Analizador Sintáctico (AS) actual que posee la herramienta IDA no es el “ideal”, debido a que no se capturan todos los ids dentro del código, solo

se extraen los ids en su punto de declaración. Esto quiere decir, que todos los ids que se declaran son capturados (ya sean locales dentro de una función, dentro de una estructura de control (if, while, etc.) o una variable de clase), pero no se extraen las ocurrencias, es decir, el uso del id. Desarrollar un AS que también capture las ocurrencias de los ids, implica un gran esfuerzo debido a la complejidad del mismo. Esta conclusión, se determinó mediante la experiencia obtenida producto de la construcción del actual AS de IDA. Sin embargo, para el caso de los comentarios y literales, el AS los extrae en forma completa.

En función de lo antedicho, una propuesta a futuro es ampliar el AS para que capture todos los ids presentes en los archivos JAVA. Con esto, se brindará al usuario más y mejor información estática asociada a los ids. Por otro lado, se perfeccionarán las técnicas de análisis de ids. Sin duda, la técnica más beneficiada en este sentido será Samurai, más precisamente en las tablas de frecuencias de aparición de palabras, a continuación se explican más detalles al respecto.

Las dos tablas de frecuencias de aparición de palabras que son utilizadas por el Algoritmo Samurai en la función score, son la tabla de frecuencias local y la tabla de frecuencias global (ver capítulo 3 - sección **??**). En la tabla local se considera la frecuencia de aparición de palabras en el código de estudio actual, mientras que la global, se asocia a un conjunto grande de programas externos. A continuación, se explican las mejoras que recibiría cada tabla, si el AS se amplia:

Tabla de Frecuencia Local: Intuitivamente, cada palabra en esta tabla que esté asociada a un id[1], tendrá la frecuencia de aparición local mucho más precisa dado que se capturan todas las ocurrencias del id.

Tabla de Frecuencia Global: Esta tabla originalmente fue construida por los autores, a partir del análisis de 9000 programas JAVA (ver capítulo 3 - sección **??**), y la misma no está disponible. Por lo tanto, se tomó la iniciativa de construir una aproximación. Esta aproximación

[1]Producto de la separación Hardword (easyCase se obtienen las palabras easy case, por ejemplo).

se efectuó ejecutando el mismo AS que se utiliza en IDA, pero para 20 programas JAVA tomados como muestra. Una vez capturadas las palabras provistas por el AS de ids, comentarios y literales, se calculó la frecuencia de aparición de cada una. Teniendo en cuenta que este AS se va ampliar para que capture todos los ids del código. Si se corre nuevamente el AS para el mismo lote de 20 programas antedicho, se mejorará aun más la precisión de la tabla de frecuencias globales construida, ya que la cantidad de palabras vinculadas a ids serán más precisas. A su vez, como mejora extra, se puede correr el AS para una cantidad de programas superior a 20 y de esta manera tener mayor variedad de palabras.

5.2.2. Implementar otro Algoritmo de Expansión

Esta propuesta consiste en implementar una nueva técnica de análisis de ids en IDA, más precisamente un nuevo algoritmo de expansión. Un algoritmo interesante es AMAP (Automatically Mining Abbreviation Expansions in Programs) descripto en el capítulo 3 - sección **??**. El mismo, observa gradualmente en el código las palabras presentes partiendo desde el lugar del id que se desea expandir. La búsqueda de palabras candidatas para expandir las abreviaturas comienza en un alcance cercano (palabras ubicadas en sentencias cercaras a la abreviatura), luego en caso de no tener éxito, el alcance se va expandiendo a los métodos (sentencias, comentarios de métodos, etc.), clases (comentarios de clase, variables de clase, etc.). En caso de ser necesario, también puede explorar palabras contenidas en el sistema completo, librerías estándar de JAVA y/o otros programas JAVA. Con estas caracterizadas, la técnica AMAP no necesita grandes diccionarios con palabras en lenguaje natural como el caso del algoritmo Básico de Expansión. Además, AMAP, a diferencia del algoritmo de expansión convencional, posee criterios de selección inteligentes que permiten elegir la mejor expansión para una abreviatura, ante muchas posibilidades de expansión. Esta propuesta para la herramienta IDA brindaría al usuario mejores expansiones a los ids y una nueva opción en lo que respecta al análisis de ids.

```
public Person(int doc, String name, int age, String gen) {
    setDocPrs(doc);
    setNm(name);
    setAg(age);
    setGndr(gen);
}

public void setDocPrs(int value) {
    this.docPrs = value;
}
```

```
public Person(int document,String name,int age,String gender){
    set_document_person(document);
    set_name(name);
    set_age(age);
    set_gender(gender);
}

public void set_document_person(int value) {
    this.document_person = value;
}
```

Figura 5.1: Comparación de un código, antes y después de expandir los ids.

5.2.3. Expandir Identificadores en el Código

Para ubicarse en el contexto de esta mejora, IDA tiene un panel (Panel de Elementos Capturados - ver capítulo 4) en donde se visualiza el código del archivo ingresado para que el usuario lo visualice.

Una propuesta de mejora en la herramienta IDA, consiste en traducir los ids que se muestran en esta visualización del código. Esta traducción implica reemplazar cada id ubicado en el código por la expansión que fue llevada a cabo, dado que hay dos tipos de expansiones por cada id (expansión desde Greedy/Samurai), lo que se permitirá es que el usuario pueda elegir entre ambas alternativas la que mejor le parezca. De esta forma, se obtendrá un código más legible y ayudará a comprenderlo más fácilmente. En la figura 5.1 se explaya esta idea, aquí se compara un código, antes y después de expandir los ids.

Luego el nuevo código con los ids expandidos se podrá guardar en un nuevo archivo de salida JAVA, este nuevo archivo será funcionalmente equivalente al archivo ingresado, pero tendrá los ids expandidos. Esta idea de traducción y creación de un nuevo archivo, fue tomada de una de las características que posee la herramienta Identifier Restructuring cuyos autores

son Tonella y Caprile (ver capitulo 3 - sección **??**), ya que esta herramienta realiza una traducción similar de ids generando un código más comprensivo.

5.2.4. Acoplar a entornos de desarrollos

Una interesante extensión futura para la herramienta IDA, consiste en adaptarla como extensión (plugin) para un entorno de desarrollo integrado, como es el caso de NetBeans o Eclipse. Esto permitiría que el usuario abra un proyecto JAVA desde estos entornos, e inmediatamente con IDA expanda los ids para mejorar la comprensión. Esta propuesta, en parte es similar a una de la características de la herramienta Identifier Dictionary (IDD), que fue desarrollada por Deissenboeck y Pizka (ver capítulo 3 - sección **??**). La herramienta IDD es un plugin de eclipse que al compilar un proyecto en JAVA, automáticamente captura y enumera dentro de una tabla los ids presentes en el proyecto, luego el usuario puede renombrar cada id desde esta tabla a una forma más comprensiva.

Para la herramienta IDA se propone construir un plugin similar al de IDD, en donde se enumeran los ids en una tabla, pero el renombre de ids en IDA a diferencia de IDD es más automático, ya que IDA expande los ids por medio de las estrategias que tiene implementadas. El usuario solo deberá intervenir para determinar que expansión es la más adecuada, entre los distintos resultados que se obtengan, producto de las diferentes estrategias de análisis de ids ejecutadas.