CAPITULO I

Introducción

1. Mantenimiento de Sistemas de Software

Los sistemas de software exitosos están destinados a sufrir incesantes modificaciones con el objetivo de mantenerse vigentes en el mercado. Los diseños de estos sistemas se ven desbordados por los nuevos requisitos funcionales surgidos en la etapa de mantenimiento, deteriorándose con el transcurso del tiempo. Incluir nuevas funcionalidades o realizar modificaciones a las ya existentes presupone un alto costo cuando estos eventuales – y asiduos escenarios en sistemas exitosos – no han sido contemplados en las etapas tempranas del proceso de desarrollo. Es así que resulta de suma importancia que el atributo de calidad de modificabilidad se vea reflejado claramente en estos sistemas [3].

2. Separación de Concerns

Se define a un concern como cualquier cuestión de interés en un sistema de software. La separación de estos desde la etapa de diseño es una forma de obtener sistemas más flexibles, que no sufran en demasía el impacto generado por requerimientos cambiantes. Realizar una separación exitosa de los concerns de importancia en un sistema de software aumenta la calidad del mismo, reduciendo los costos de producción y mantenimiento y facilitando su evolución, asegurando una competitividad prolongada en el mercado [2].

Conseguir aislar los concerns de un sistema de software no es una tarea sencilla. En la práctica resulta casi imposible llevar a cabo en forma completa esta separación. Este

fenómeno es conocido como la “tiranía de la descomposición dominante” [1.1] No importa cuán bien una aplicación se descomponga en unidades modulares, siempre existirán concerns que atraviesan dicha descomposición. Estos se denominan crosscutting concerns y se encuentran diseminados por diferentes módulos del sistema, generando un impacto negativo en la calidad del mismo: la comprensibilidad del código se ofusca, la adaptabilidad disminuye, y la evolución del sistema se torna ardua y costosa.

3. Programación Orientada a Aspectos (AOP, Aspect Oriented Programming)

Con el objetivo de resolver la problemática del aislamiento exitoso de concerns surge un nuevo paradigma, la programación orientada a aspectos (AOP) [2]. AOP provee mecanismos para separar la funcionalidad central de un sistema de software de los concerns que atraviesan sus módulos. El código correspondiente a los crosscutting concerns es encapsulado en un nuevo constructor llamado aspecto. Esta nueva estructuración permite liberar a los módulos centrales del sistema de código que no corresponden a la lógica del negocio, y centralizar este código en los aspectos.

Dado que el paradigma introducido es bastante joven, el principal desafío reside en la migración de sistemas legados. Es decir, en tomar sistemas orientados a objetos – ya consolidados – y encontrar su equivalente orientado a aspectos. Esta migración permitirá mejorar la comprensión de los sistemas y la mantenibilidad y extensibilidad de los mismos.

En el proceso de migración se definen dos actividades principales, aspect mining y aspect refactoring [1.2]:

- Aspect mining: es la actividad de descubrir aquellos crosscutting concerns desde el código fuente o las trazas de ejecución de una aplicación que podrían ser encapsulados como aspectos del nuevo sistema.

- Aspect refactoring: es la actividad de transformar los aspectos candidatos identificados en el código orientado a objetos en aspectos reales en el código fuente.

Como resultado del estudio de estas actividades, surge un conjunto de técnicas que facilitan su puesta en práctica. Debido a la dimensión y complejidad de los sistemas legados y a la documentación incompleta sobre los mismos, la ejecución manual de estas técnicas de migración se torna un proceso propenso a errores, dificultoso, y muy costoso. Es así que surge la necesidad de contar con herramientas que asistan al programador en las etapas del proceso de migración aspectual.

4. Aspect Mining Tool (AMT)

El objetivo del presente trabajo es el desarrollo de una herramienta capaz de asistir en la actividad de aspect mining. Para la implementación de la misma se utiliza un motor de inferencia que permite la identificación de crosscutting concerns de manera semi-automática. La herramienta se sustenta de la implementación de algoritmos estáticos de aspect mining para conseguir los resultados deseados. La combinación entre el motor de inferencia y el conocimiento sobre las técnicas de aspect mining constituyen un sistema experto en el dominio en cuestión. [1.3]

La herramienta desarrollada puede analizar proyectos codificados en lenguaje Java, examinando sus elementos en busca de seeds candidatos. En el proceso de identificación de seeds se pueden distinguir tres etapas fundamentales: la traducción de la información necesaria del proyecto a hechos lógicos, el procesamiento de estos para la obtención de resultados, y la presentación de estos últimos al usuario de la herramienta (Fig. I - 1).

Al ejecutar un análisis sobre un proyecto, el código de este es recorrido por un parser. El parser representa la estructura sintáctica del mismo en forma de árbol, haciendo uso de la noción de AST (Abstract Sintax Tree o árbol sintáctico abstracto) [5]. El propósito de este recorrido es generar un conjunto de hechos, que representan información estática

del proyecto, que sirvan como entrada para los algoritmos de aspect mining conocidos por el sistema experto. Una vez procesada esta información, se obtienen como resultado los seeds candidatos del sistema. Con la posterior intervención del programador se pueden extraer del código del proyecto los crosscutting concerns a refactorizar.

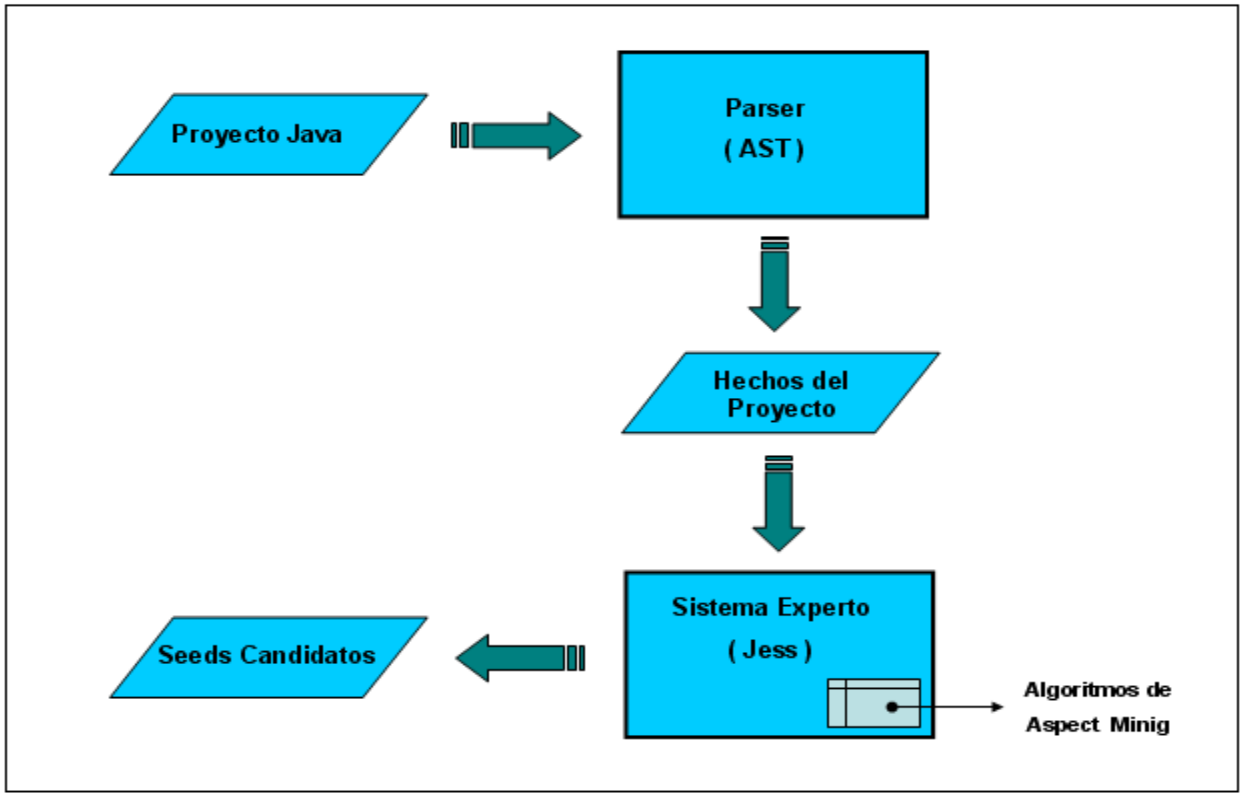

Fig. I - 1. Proceso de extracción de crosscutting concerns.

Los enfoques que se desarrollan en el trabajo analizan la información estática del sistema, es decir, la proveniente del código fuente. Se implementan cinco algoritmos: análisis de Fan-in, análisis de Métodos Únicos, análisis de Relaciones de Ejecución, análisis de Métodos Redirectores y un último algoritmo, denominado Sinergia, que une a los 3 primeros.

Para la implementación y ejecución de los algoritmos de aspect mining se utiliza Jess [1.3, 1.4], un motor de reglas para el lenguaje Java, desarrollado en este mismo lenguaje en los laboratorios Sandia. La elección de este motor se fundamenta básicamente en la naturaleza del problema. La facilidad de representar los algoritmos como un sistema

experto proviene de la posibilidad de escribir conjuntos de reglas de manera independiente. Esta característica también permite el enriquecimiento del sistema sin realizar mayores modificaciones. Si se desea agregar conocimiento al sistema experto – agregado de algoritmos de aspect mining o refinamiento de los ya implementados – basta con escribir nuevas reglas o modificar las existentes. A su vez, la velocidad de procesamiento constituye un punto muy importante a causa de la magnitud de la información entrante. Jess usa una versión optimizada del algoritmo Rete [6]. El mismo sacrifica espacio para ganar velocidad.

Health Watcher (HW) [111] fue el sistema elegido para realizar las pruebas de la herramienta. HW es una aplicación desarrollada acorde a una arquitectura por capas utilizando tecnología J2EE. El mismo es seleccionado debido a que es un sistema real y suficientemente complejo e involucra un gran número de concerns clásicos, como por ejemplo sistemas usables, concurrentes, persistentes y distribuidos.

Los enfoques seleccionados para realizar las pruebas fueron Sinergia y Métodos Redireccionadores. Se realizaron dos corridas del primer enfoque variando sus parámetros de entrada y se logró identificar un 50% y 66.66% de los concerns existentes en el sistema con una precisión de 75% y 33.33% respectivamente. El análisis de Métodos Redireccioandores identificó el 16.66% de los concerns con una precisión del 100%, y adicionó al conjunto de concerns identificados clases adapters diseminadas en la capa de negocio del código.

5. Organización del Documento

El presente documento se organiza de la siguiente manera:

- En el capitulo 2, se introduce la problemática relacionada al mantenimiento de aplicaciones. Se presenta el paradigma de Programación Orientada a Aspectos (AOP) y el proceso de migración de sistemas legados.
- En el capitulo 3, se presentan y explican las técnicas de Aspect Mining de interés para este trabajo.

- El capitulo 4 constituye un marco teórico sobre los Sistemas Expertos Basados en Reglas.

- La propuesta del trabajo de tesis es presentada y desarrollada en el capítulo 5, donde se explican los algoritmos desarrollados como sistemas expertos y la herramienta en cuestión.

- El capitulo 6 muestra la experiencia del uso de la herramienta sobre un caso de estudio, analizando los resultados obtenidos.

- El 7mo y último capítulo muestra las conclusiones y trabajos futuros obtenidos del desarrollo del proyecto.

Referencias

[1.1] B Hannemann, J., Kiczales, G.: Design Pattern Implementation in Java and Aspectj. In: 17th ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, pp. 161--173. ACM Press, New York, USA (2002)

[1.2] Kellens, A., Mens, K. A survey of aspect mining tools and techniques. Technical report, INGI 2005-07, Universite catholique de Louvain, Belgium (2005)

[1.3] Friedman-Hill E. (2003). "Jess in Action". Manning Publications, ISBN 1-930110-89-8.

[4] http://www.jessrules.com/

[3] Khalid Al-Jasser, Peter Schachte, Ed Kazmierczak, "Suitability of Object and Aspect Oriented Languages for Software Maintenance," aswec, pp.117-128, 2007 Australian Software Engineering Conference (ASWEC'07), 2007 .

[2] [Laadad 2003] Ramnivas Laadad. "AspectJ in Action". ©2003 by Manning Publications Co. All rights reserved.

[5] Joel Jones. Abstract Syntax Tree Implementation Idioms. The 10th Conference on Pattern Languages of Programs, Sep. 8th-12th, 2003.

[111] Soares, S.; Borba, P.; Laureano, E.: Distribution and Persistence as Aspects. In: Software Practice and Experience, Wiley, vol. 36 (7), (2006) 711-759.