REVISTA COLOMBIANA DE COMPUTACIÓN
Volumen 11, número 1
Págs. 69 - 84

AMII: un prototipo para descubrir Crosscutting Concerns en Sistemas Legados

Franco Herrera, Mirta Miranda, Fernanda Oyarzo, Karina Rama y Sandra Casas*

Fecha de Recibido: 14/01/2010 Fecha de Aprobación: 02/03/2010

Resumen

El mantenimiento y evolución de sistemas legados es complejo y costoso. Surge la necesidad de contar con herramientas y técnicas que ayuden a los desarrolladores en estas tareas. Identificar crosscutting concerns (CCC) y transformar los mismos en aspectos se considera un aporte que puede solucionar en parte éstos problemas. El proceso de identificar CCC en sistemas legados se conoce como Minería de Aspectos (Aspect Mining). Este trabajo presenta el prototipo AMII que analiza el código fuente Java y genera una estructura semejante a un índice invertido, la cual registra ciertas características y propiedades del código con el propósito de posibilitar la identificación de CCC. El trabajo incluye una experiencia inicial sobre la técnica y la herramienta.

Palabras clave: *Minería de Aspectos, sistemas legados, programación orientada a aspectos..*

Abstract

The maintenance and evolution of legacy systems is complex and costly. The need arises for tools and techniques that help developers in these tasks. Identifying crosscutting concerns (CCC) and transform them in aspects that can be considered a contribution partial solution to these problems. The process of identifying CCC legacy systems is known as Mining Aspects. This paper presents a prototype AMII that analyzes Java source code and creates a structure similar to an inverted index, which records certain characteristics and properties of the code with the purpose of permit the identification of CCC. The work includes a first experience about the technique and tool.

Keywords: *Aspect mining, Legacy Systems, Aspect oriented programing.*

* Unidad Académica Río Gallegos Universidad Nacional de la Patagonia Austral, Lisandro de la Torre 1060 Río Gallegos CP 9400 Argentina, lis@uarg.unpa.edu.ar

1 Introducción

La Programación Orientada a Aspectos (POA) [1] es un enfoque que ha sido propuesto para la implementación de CCC en forma separada y aislada [2] [3], para lo cual se emplea una nueva abstracción denominada aspecto. En forma asociada el enfoque proporciona mecanismos de composición originales que posibilitan la separación implícita de los aspectos de las unidades funcionales (clases o componentes). Se ha demostrado que las implementaciones POA resultan más reutilizables, mantenibles y evolucionables [4].

Un sistema legado es el término que se suele emplear para referiste a los sistemas software que a pesar de ser antiguos suministran servicios esenciales. Normalmente son aplicaciones complejas, de difícil mantenimiento y que por el grado de criticidad y coste de modernización, continúan activos. Por falta de documentación y debido a la salida del personal técnico que participó originalmente en su desarrollo los sistemas legados pueden presentar problemas como: dificultad de comprensión de las reglas de negocio en ellos implementadas, desconocimiento de las razones que llevaron a determinadas decisiones, problemas en la estructuración de los módulos de código, diversidad de estilos de programación, obsolescencia de las herramientas de desarrollo, etc. Surge la necesidad de contar con herramientas y técnicas que ayuden a los desarrolladores a disminuir los costos y esfuerzos que implican las tareas de mantenimiento y evolución. Por estas razones, identificar CCC y transformar los mismos en aspectos se considera un aporte que puede solucionar en parte los problemas de mantenimiento y evolución de sistemas legados. Kellens, Mens y Tonella [5] definen "Minería de Aspectos" como la actividad de descubrir aquellos CCC que potencialmente podrían convertirse en aspectos, desde el código fuente y/o desde el comportamiento del sistema de software. Las técnicas de minería de aspectos toman como entrada el código de un sistema legado y de manera automática, o semiautomática, generan un conjunto de aspectos candidatos. Teniendo en cuenta que el propósito de la Minería de Aspectos es asistir con información a la posterior transformación de CCC en aspectos (refactorización) es relevante analizar toda información que optimice, mejore y reduzca los esfuerzos de refactorización. Precisamente una de las desventajas de las técnicas de minería de aspectos es que no asisten eficientemente a la posterior refactorización del código, como lo reportan algunos autores [6] [7].

Este trabajo presenta el prototipo AMII. AMII analiza el código fuente Java y genera una estructura semejante a un índice invertido [8], la cual registra ciertas características y propiedades del código con el propósito de posibilitar la identificación de CCC. El objetivo es proporcionar información que resulte útil para mejorar el posterior proceso de refactorización.

Este artículo se estructura de la siguiente manera: en la Sección 2 se expone con más detalles los trabajos relacionados con esta propuesta; en la Sección 3 se describe la estructura del índice invertido y su construcción; en la Sección 4 se presentan las experiencias realizadas; en la Sección 5 se compara AMII con otras herramientas y en la Sección 6 se exponen las conclusiones y trabajo futuro.

2 Descrubrimiento de Aspectos

La Minería de Aspectos [5] es la actividad de descubrir CCC que potencialmente pueden convertirse en aspectos. Estos concerns pueden, posiblemente, ser mejor modularizados aplicando soluciones orientadas a aspectos o podrían documentarse con propósitos de una mejor comprensión de un programa. Desde el punto de vista de comprensión de un programa, los CCC son un problema por dos razones. Primero, descubrir o entender la implementación de este tipo de concern es difícil ya que el mismo no está localizado en un único módulo sino que está disperso (scattered) a través de muchos módulos. Segundo, entender la implementación de los módulos en sí mismo se vuelve más complicado, ya que el código de los diferentes concerns se mezcla (tangled) con la funcionalidad principal de esos módulos. Por lo tanto, la utilización de la tecnología orientada a aspectos sobre sistemas legados, favorecerá la separación de estos CCC del código base. La consecuencia será un sistema más fácil de entender, mantener y evolucionar.

Sin embargo, la aplicación manual de las técnicas orientadas a aspectos sobre sistemas legados es un proceso difícil y que tiende a introducir errores. Debido al gran tamaño de estos sistemas, la complejidad de su implementación, la falta de documentación y de conocimiento de los mismos, existe la necesidad de herramientas y técnicas que ayuden a los ingenieros de software en la localización o documentación de CCC.

El estudio y desarrollo de tales técnicas es el objetivo de la minería de aspectos y refactorización aspectual. Mientras que la minería de aspectos se ocupa de descubrir los potenciales CCC, la refactorización aspectual es la actividad de transformar esos aspectos potenciales en aspectos reales del sistema de software.

Existen tres tipos de enfoques para el descubrimiento de aspectos [5]:

- Técnicas de descubrimiento de aspectos tempranos: implica descubrir aspectos durante etapas tempranas del ciclo de vida del software, tales como requerimientos, análisis de dominio o diseño de la arquitectura. Aunque, en el contexto de sistemas legados,

donde los documentos de requerimientos y de la arquitectura no están actualizados, estas técnicas no pueden ser aplicadas.

- Visualizadores (browsers) específicos: que ayudan a los desarrolladores a navegar manualmente el código fuente de un sistema para explorar los CCC.
- Técnicas de minería de aspectos: automatizan el proceso de descubrimiento de aspectos y proponen a su usuario uno o más aspectos candidatos. Este tipo de técnicas razonan acerca del código fuente del sistema o sobre los datos obtenidos mediante la ejecución o manipulación del código.

Finalmente, en base a la clasificación anterior Kellens, Mens y Tonella [5] proponen la siguiente definición de minería de aspectos:

La "Minería de Aspectos es la actividad de descubrir aquellos CCC que potencialmente podrían convertirse en aspectos, desde el código fuente y/o desde el comportamiento del sistema de software. Se refieren a tales concerns como "aspectos candidatos".

El espacio entre secciones debe ser de 2 líneas en blanco. En caso de referencias bibliográficas, se debe seguir el formato del paquete Bibtex, como se presenta al final de este archivo, donde se referencia a [1] y [2]. Como usted puede ver, cada citación corresponde a un numero encerrado en paréntesis cuadrados.

Las técnicas de minería de aspectos se clasifican principalmente en estáticas o dinámicas. Las técnicas dinámicas se centran en el análisis del comportamiento de los sistemas, en base a la observación de trazas de ejecución, que surgen ya sea de escenarios o de casos de uso. Algunos trabajos [8] [9] [10] [11], aplican estrategias como analisys formal concept, clustering y detección de reglas de asociación, búsqueda recurrente de patrones, etc.

Las técnicas estáticas se enfocan en analizar el código fuente, en la Tabla 1 se resumen algunos de estos trabajos.

Autores/Trabajo	Estrategia	Tool	Descubre
Bruntnik [12][13]	Detección de clones basado en análisis de tokens (análisis léxico).	-	Fragmentos de código repetido.
Marin, Van Deursen y Moonen [14]	Análisis de fan-in.	FINT	Métodos repetidos.
Moldovan and Serban [15]	Análisis de fan-in y clustering de métodos.	-	Métodos repetidos
Shepherd, Gibson y Pollock [16]	Grafo de dependencia de programa (cada sentencia es un nodo y las aristas son el control). Comparación de los grafos.	Ophir	Métodos repetidos
Shepherd y Pollock [17]	Clustering jerárquico acumulativo (agrupar métodos relacionados).	AMAV	Métodos repetidos
Zhan y Jacobsen [18]	Random walks – grafo de acoplamiento (modelo de Markov) Aplica medidas de popularidad y significancia (page rank)	PAM	Métodos repetidos Fragmentos de código repetido

Tabla 1. Técnicas de Minería de Aspectos estáticas

Estas técnicas analizan el código fuente en cierta forma "a ciegas", en el sentido que cualquier método/clase es un potencial CCC. Esto en general provoca que entre los resultados aparezcan potenciales CCC que no son tales, lo que se denomina "falsos positivos". Algunas técnicas incluyen operaciones de filtrado posteriores o establecen una priorización mediante el cálculo de alguna métrica; de esta forma se intenta reducir la cantidad de falsos positivos que genera la técnica de minería de aspectos. De todas formas, implican una mayor intervención e interpretación del desarrollador.

3 AMII

AMII (Aspect Mining Inverted Index) es un prototipo que analiza el código fuente Java con el propósito de asistir a la identificación de CCC. La estrategia que emplea se basa en generar una estructura de tablas que registran propiedades y características del código que permiten identificar patrones y situaciones comunes y recurrentes.

La estructura subyacente detrás de las tablas responde al esquema de un índice invertido. Los índices invertidos [8] son la estructura de datos más elemental para la recuperación de palabras. Se usan ampliamente en el área Information Retrieval (IR) y su principal ventaja es que resuelven búsquedas a un mejor costo que otras estructuras. Un índice invertido en su versión más clásica, consta de dos elementos:

- Vocabulario: conjunto de términos distintos del texto.
- Posteo: para cada término, la lista de documentos donde aparece.

3.1 El índice invertido de CCC

La estrategia consiste en construir una estructura de índice invertido. En este caso, se consideran como palabras a indexar las invocaciones a métodos y el manejo de excepciones. El espacio de búsqueda se reduce significativamente si los métodos invocados corresponden a clases que a priori se suponen que no forman parte del dominio, denominadas clases soporte [19]. Un documento es el código fuente de una clase Java. La estructura completa del índice incluye cuatro estructuras: vocabulario, LDPI (lista de posteo de invocaciones) y LDPE (lista de posteo de excepciones) y Enlaces.

La construcción y generación de la estructura completa se realizar a partir del pre-procesamiento del código fuente de todas las clases. Básicamente se analiza el texto fuente buscando palabras válidas (invocaciones a métodos de clases soporte y excepciones). Primero se

construyen LDPI y LDPE. Cuando este proceso ha finalizado, a partir de LPDI y LDPE se construye el vocabulario, por último se generan vistas y consultas que integran el código fuente. A continuación se describe cada estructura.

3.1.1 LDPI

Dado un subconjunto de 1 o más clases en un sistema, que se establecen como objetivo de análisis, por cada invocación i, AMII procede obteniendo la siguiente información:

i = (MCC, C, L, R, #l, [D, E, MI, O, #b])
dónde:

- MCC (método y clase contenedor de la invocación)
- C (categoría de la invocación)
- L (lugar de la invocación)
- R (tipo de referencia).
- #l (número de línea)
- D (distancia)
- E (enlace)
- MI (método invocado)
- (orden)
- #B (número de bloque)

Una invocación a un método de una clase soporte se presenta siempre en el cuerpo del método de otra clase, esta propiedad es representada por el campo MCC. Una invocación se puede presentar de diferentes formas en el texto fuente, es necesario distinguir y clasificar en categorías cada invocación. Una clasificación primaria establece las categorías de la Tabla 2. Esta información posibilita la identificación de patrones en la invocación de los métodos. Esta propiedad se representa por el campo C.

Categoría	Descripción
Invocación Simple	la invocación es independiente de las demás sentencias del método.
Bloque de Invocaciones	cuando un grupo de invocaciones a la misma clase soporte se hallan contiguas en el método.
Invocación Condicional	la invocación es parte de una estructura de selección.
Invocación Iterativa	la invocación es parte de una estructura iterativa.
Invocación en Excepción	la invocación se presenta en un bloque de tratamiento de excepciones.
Lanzamiento	el CCC es una excepción y la invocación se refiere a su lanzamiento.
Invocación única	la invocación es la única sentencia en el método contenedor.
Combinación	dos o más categorías se dan en forma simultánea.

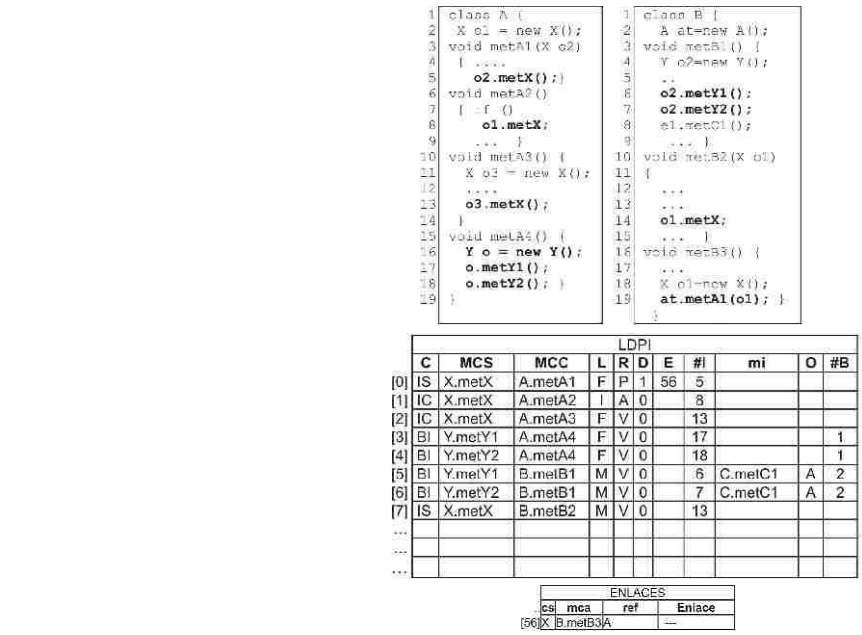
Tabla 2. Categorías de Invocación a Métodos

Se determina si la invocación se encuentra al inicio, al final o al medio del código del cuerpo del método contenedor cuando éste está compuesto de varias líneas de código. Cuando una invocación a un método soporte no se encuentra al inicio o al fin de un método, se dice que se encuentra al medio. Si una invocación se encuentra al medio, puede ocurrir que esté precedida o sucedida por otras invocaciones. Esta información permite descubrir los join-point y posibles patrones para determinar los futuros métodos de composición (after-before-etc.). Si una invocación está al inicio y al final de un método, es la única sentencia del método. Esta propiedad se representa por el campo L.

Una invocación a un método de una clase soporte surge desde una instancia (objeto) particular de dicha clase o bien es un método de clase (static). En el primer caso, el origen (declaración) de la instancia es una variable local declarada en el método que contiene la invocación o es un atributo declarado de la clase que contiene la invocación o es un parámetro declarado en la lista de argumentos del método que contiene la invocación. Si el CCC resulta ser una variable local o un atributo, o la invocación a un método estático, la refactorización a aplicar se refiere a la clase que los contiene. Muy distinto es, si el CCC es un parámetro lo que implica que el CCC afecta más de una clase y su futura refactorización es más compleja. Esta información sirve para conocer a priori las clases a refactorizar por cada CCC. Esta propiedad se representa por medio del campo R.

Otra información surge del valor que tomen algunos de estos campos, como es el caso de: D, E, MI, O y #B. D y E se emplean cuando R es parámetro. E referencia una entrada en la tabla Enlaces que registra el método y clase que genera la invocación con el parámetro CCC. Los enlaces se mantienen hasta alcanzar el método en el cual la referencia es un atributo o una variable local. Esta cadena de enlaces genera una distancia (D) entre el origen de la referencia y el uso analizado. MI y O se emplean cuando L es medio e indican la relación con otra invocación. MI es un método invocado y O es antes o después. #B se emplea cuando categoría de la invocación es un bloque de invocaciones y se refiere a un número asignado por el analizador que permite agrupar las distintas sentencias de los bloques de invocaciones.

La Figura 1 presenta un ejemplo. Las clases X e Y son clases soporte. El análisis estático de la clase A genera cinco entradas en LDPI. Las invocaciones al método metX de X se han registrado en las posiciones 0, 1 y 2 respectivamente, mientras que las invocaciones a metY1 y metY2 de la clase Y se han registrado en las posiciones 3 y 4.



```
1  class A {
2   X o1 = new X();
3  void metA1(X o2)
4   { ....
5     o2.metX();}
6  void metA2()
7   { if ()
8     o1.metX;
9     ...  }
10 void metA3() {
11   X o3 = new X();
12   ....
13   o3.metX();
14  }
15 void metA4() {
16   Y o = new Y();
17   o.metY1();
18   o.metY2(); }
19 }
```

```
1  class B {
2    A at=new A();
3  void metB1() {
4    Y o2=new Y();
5    ..
6    o2.metY1();
7    o2.metY2();
8    e1.metC1();
9     ... }
10 void metB2(X o1)
11 {
12   ...
13   ...
14   o1.metX;
15   ...  }
16 void metB3() {
17   ...
18   X o1=new X();
19   at.metA1(o1); }
   }
```

LDPI

	C	MCS	MCC	L	R	D	E	#I	mi	O	#B
[0]	IS	X.metX	A.metA1	F	P	1	56	5			
[1]	IC	X.metX	A.metA2	I	A	0		8			
[2]	IC	X.metX	A.metA3	F	V	0		13			
[3]	BI	Y.metY1	A.metA4	F	V	0		17			1
[4]	BI	Y.metY2	A.metA4	F	V	0		18			1
[5]	BI	Y.metY1	B.metB1	M	V	0		6	C.metC1	A	2
[6]	BI	Y.metY2	B.metB1	M	V	0		7	C.metC1	A	2
[7]	IS	X.metX	B.metB2	M	V	0		13			
...											
...											
...											

ENLACES

	cs	mca	ref	Enlace
[56]	X	B.metB3	A	---

Fig 1. Entradas en LDPI generadas a partir del análisis de las clases A y B

El análisis de la clase B ha generado 3 entradas en LDPI. Las invocaciones a los métodos metY1 y metY2 de la clase Y se han registrado en las posiciones 5 y 6, mientras que la invocación al método metX de X se ha registrado en la posición 7.

En cada caso se ha determinado la categoría de la invocación (IS: invocación simple, IC: invocación condicional, etc.), el lugar de la invocación (I: inicio, F: final, M: medio y U: única sentencia) y el tipo de referencia (parámetro, atributo o variable local). La entrada de la posición 0 esta enlazada con la entrada de la posición 56 de la tabla Enlaces. Esto es debido a que la referencia es un parámetro. En la posición 56 la misma instancia es un atributo en la clase B y la invocación a A.metA1 surge del método B.metB3.

3.1.2 LDPE

En la lista de posteo de excepciones se registran las excepciones halladas en el código fuente. En este caso se registra información similar a la que se almacena en LDPI (el nombre de la clase excepción, nombre del método de la clase clave que contiene el manejador, etc.).

3.1.3 Vocabulario

El vocabulario de invocaciones es una tabla que resume las entradas de LDPI a nivel método de clase soporte. El vocabulario permite hallar

posibles patrones en las invocaciones y determinar así los métodos de composición y declaración de los pointcuts de los CCC como aspectos. La estructura básica del vocabulario está compuesta por los siguientes elementos:

- Nombre del método soporte;
- Total de invocaciones al método descubiertas;
- Total de invocaciones al inicio, fin, medio;
- Total de invocaciones como variable local, parámetro, atributo, estática;
- Total de invocaciones por categoría;

3.2 Diseño e Implementación

La arquitectura de AMII se compone de tres capas: a) capa núcleo, b) capa persistencia y c) capa interfaz. La capa núcleo consiste en un conjunto de unidades cuya responsabilidad es la construcción y generación del índice invertido. En la Figura 2 se presenta un diagrama de clases resumido de las clases más importante que intervienen en este proceso.

AMII controla todo el proceso, aunque el trabajo más pesado es llevado a cabo por SCAN. Cada clase a analizar primero es convertida a un formato que contiene sólo la información que se considera relevante, esta responsabilidad es llevada a cabo por ClassFormatter. Las clases formateadas son examinadas por SCAN que analiza cada método de las clases recolectando información del código. Por cada método SCAN genera una instancia de SMethod. Los objetos SMethod contienen información de su contenido (clase contenedora, parámetros, número de línea de inicio y número de línea de fin, variables locales, invocaciones, tipo de retorno, instrucciones condicionales e iterativas, manejadores de excepciones, etc.). Luego, se analizan el conjunto de SMethod obtenidos para construir LDPE y LDPI, que están representadas por las clases InvocationPostingList y ExceptionPostingList. Por último, se genera un objeto Vocabulary, que representa al vocabulario.

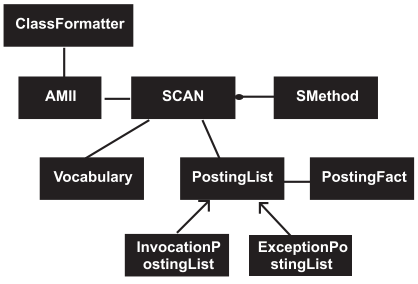


Fig 2. Diagrama de Clases Resumida de la Capa Núcleo

Finalizado el proceso la capa de persistencia es responsable de almacenar la estructura en una base de datos (HSQLDB) y la capa de interfaz es responsable de la visualización de LDPE, LDPI, el vocabulario y diversas consultas sobre los datos registrados. En la Fig. 3 se visualiza la LDPI desde una ventana de AMII.

Fig. 3. Interfaz de AMII

4 Exprerimentos

Se realizó una experiencia sobre un software de escala pequeña pero real, del cuales no se dispone ninguna documentación excepto el código y cuyos desarrolladores no pertenecen el equipo de investigación. Estas condiciones resultan apropiadas para probar la funcionalidad de AMII.

MiPitanza (http://mipitanza.sourceforge.net/) es una guía personal de restaurantes para teléfonos móviles habilitados con J2ME (MIDP 1.0 y 2.0) que no requiere servidor. MiPitanza se compone de 3 paquetes: Main, Language y UI; y un total de 16 clases. De éstas se han considerado 9 como clases soporte, más otras 10 de los paquetes Standard o importados y excepciones.

El análisis de AMII ha producido 88 entradas en LDPI y 23 entradas en LDPE. El vocabulario se muestra en la Tabla 4 del Apéndice. El método setCurrent de la clase Display es el más invocado, ya sea al inicio o final de métodos y en dos ocasiones es la única sentencia del método contenedor. La referencia es estática o un atributo de clase. En cuanto a la categoría de invocación siempre se presenta como invocación simple. Este primer análisis parece indicar que es el método más repetido en el sistema, además de mostrar un patrón de comportamiento similar.

Con la información registrada en LDPI se ha generado una consulta que por cada clase soporte calcula las clases afectadas. Los resultados de esta consulta se presentan en la Tabla 3.

Clase Soporte	TI	clases afectadas
Display	19	6
Alert	18	1
DataInputStream	8	1
DataOutputStream	7	1
RecordStored	6	1
UIFilter	4	1
ByteArrayInputStream	3	1
InvalidRecordException	3	2
UIEditDetails	3	1
UIList	3	1
ByteArrayOutputStream	2	1
UIMainMenu	2	1
UIShowDetails	2	1
Connector	1	1
System	1	1
TextMessage	1	1
UIAbout	1	1

Tabla 3. Total de llamadas y clases afectadas

Aquí las clases Display y Alert tienen un número similar de invocaciones (TI) pero claramente Display se esparce más que Alert. Mientras todas las invocaciones a la clase Alert se mantienen encapsuladas en una única clase, Display corta seis clases del sistema. A través de consultas adicionales a LDPI también se determina que Display sólo afecta clases que representan componentes gráficos. Por otro lado, las invocaciones a clases que conforman la interfaz gráfica (UIList, UIAbout, UIFilter, etc.) aunque sólo afectan a una clase, todas están contenidas en la misma clase, lo que implica que 32 invocaciones (de las 69 restantes descubiertas por AMII) están inmersas en el mismo módulo. Esta evidencia se presenta de manera menos clara pero 51 invocaciones corresponden al concern de interfaz gráfica. Así esta serie de consultas, reforzadas con el análisis de código (en forma no integrada) permite concluir que el CCC de interfaz gráfica se mezcla y esparce en esta simple aplicación, y que su refactorización con aspectos mejoraría la modularización del sistema. La interfaz gráfica es considerada un clásico CCC.

5 Discusión

A continuación se plantean algunas diferencias y similitudes entre AMII y otras técnicas de minería de aspectos.

AMII emplea una técnica que en esencia localiza métodos repetidos y esparcidos en los diferentes módulos de un sistema, es decir ayuda a identificar el síntoma de scattering. Asimismo se complementa por medio de consultas a las tablas generadas que permiten analizar la dispersión de las clases. Al realizar la experiencia indicada en la sección anterior, se identifica como requerimiento la integración del código fuente, para mejorar la funcionalidad de la misma.

La herramienta FINT (Fan-in Tool) [14] que esta basada en la métrica fan-in, sólo examina la dispersión a nivel método, aplica filtros arbitrarios y el lugar de la invocación (inicio o final) debe ser analizado manualmente, aspectos que AMII resuelve automáticamente.

AMII analiza las invocaciones de manera tal de encontrar sus características, una de ellas es el lugar de la invocación relativa al método que contiene a la misma, en este sentido cuando esta al inicio y al final, significa que es la única sentencia. Esto concuerda con Gybels y Kellens [20] que observaron que muchos CCC son implementados como un único método que es invocado desde muchos lugares en el código, pero además AMII proporciona información sobre la naturaleza de la referencia empleada en la invocación (atributo, parámetro, variable)

AMAV [17] y FindConcept [21] emplean como principal estrategia de análisis la técnica de procesamiento de lenguaje natural, aunque no se puede comparar su eficiencia en términos de resultados, si se puede asegurar que ambas requieren mayor intervención del desarrollador que AMII.

Como ha sido observado [7] una única técnica de minería de aspectos no hallará todo tipo y clase de CCC, dado que el problema es demasiado complejo. Se observa que últimamente se está trabajando en la combinación de técnicas que de manera unificada mejoren los resultados que logran en forma individual. Esto se rescata en [22][23][24]. Por estos motivos, más allá que AMII pueda resultar en cierto punto o contexto más adecuada que otras herramientas, y viceversa, la conclusión no puede ser totalmente aseverada. En todo caso, AMII será más adecuada en cuanto más posibilidades admita, para lo cual debe ser complementada con otros métodos. De acuerdo a esto, la información obtenida y generada por AMII puede resultar de gran utilidad a otras técnicas.

6 Conclusiones

Este trabajo presenta el prototipo AMII que registra distintas características y propiedades del código fuente Java en índices invertidos mediante análisis estático con el objeto de identificar CCC para su posterior refactorización. La información registrada en el índice invertido permite razonar al desarrollar respecto de los métodos más invocados sino también sobre ciertos patrones que se dan en sus llamadas (lugar, categoría, referencia, etc.). La información recogida en el índice invertido puede ser recuperada y relacionada mediante consultas, pero permite ser analizada mediante otras técnicas de minería de aspectos como también empleada para la aplicación de métricas específicas.

La principal limitación de la técnica es que requiere por parte de los desarrolladores un mínimo de conocimiento previo respecto de que clases considera "clases soporte", pero a la vez esta restricción reduce la cantidad de resultados no relevantes o falsos positivos. De todas formas, no constituye una restricción para AMII, puesto que todas las clases de una aplicación pueden ser analizadas por la herramienta.

El trabajo futuro está orientado a: diseñar e implementar un visualizador de consultas y código integrado; articulación e integración de los resultados con métodos de refactorización y realizar experimentos con software de mayor complejidad y tamaño.

Referencias

Este trabajo está parcialmente financiado por la Universidad Nacional de la Patagonia Austral (Argentina).

[1] G. Kiczales. et al. "Aspect-oriented programming." In ECOOP'97—Object- Oriented Programming, 11th European Conference. LNCS 1241, pp 220- 242, 1997.

[2] W. Hürsch. y C. Lopes. "Separation of Concerns". Northeastern University, TR NU-CCS-95-03, Boston 1995.

[3] P Tarr.; H. et al. "N Degrees of Separation: Multi-Dimensional Separation of Concerns." ICSE 1999 Conference Proceedings. pp 107-119, 1999.

[4] R. Laddad. "AspectJ in Action", Manning Publications Co. 2003.

[5] A Kellens; K. Mens y P. Tonella. "A Survey of Automated Code-Level Aspect Mining Techniques" Transactions on Aspect-Oriented Software Development IV – VOL 4640 – 2007- pp 143-162.

[6] M. Storzer, U Eibauer y S. Schoeffmann. "Aspect Mining for Aspect Refactoring: An Experience Report". Towards Evaluation of Aspect Mining, France, at ECOOP 2006.

[7] K. Mens, K. Kellens y Krinke J. "Pitfalls in Aspect Mining" In Proceedings of the 2008 15th Working Conference on Reverse Engineering – Bélgica.

[8] S. Breu. y J. Krinke. "Aspect Mining Using Event Traces". In Proc. International Conference on Automated Software Engineering (ASE) (2004) pp 310–315.

[9] L. He. y H. Bai. "Aspect Mining using Clustering and Association Rule Method". International Journal of Computer Science and Network Security, 6(2A):247–251. (2006).

[10] P. Tonella. y P. Ceccato. "Aspect Mining through the Formal Concept Analysis of Execution Traces". In Proceedings of the IEEE Eleventh Working Conference on Reverse Engineering (2004), pp 112–121.

[11] E. Abait; S.Vidal y C. Marcos. "Dynamic Analysis and Association Rules for Aspects Identification". II Latin American Workshop on Aspect-Oriented Software Development LA-WASP´2008, Campinas Brasil. (2008).

[12] M. Bruntink. "Aspect mining using clone class metrics". In Proceedings of the 2004 Workshop on Aspect Reverse Engineering (2004).

[13] M Bruntink et al. "On the use of clone detection for identifying crosscutting concern code". IEEE Trans. Softw. Eng., vol. 31, no. 10. (2005) pp. 804-818.

[14] M. Marin; A. Van Deursen y L.Moonen. "Identifying Aspects Using Fan-in Analysis". In Proceedings of the 11th Working Conference on Reverse Engineering (2004), pages 132–141. IEEE Computer Society.

[15] G. Moldovan y G. Serban. “Aspect Mining using a Vector-Space Model Based Clustering Approach”. In Proceedings of Linking Aspect Technology and Evolution Workshop (2006).

[16] D.Shepherd; E.Gibson y L. Pollock. “Design and evaluation of an automated aspect mining tool”. In Software Engineering Research and Practice, H. R.Arabnia, H. Reza, H. Eds. CSREA Press (2004), pp. 601-607.

[17] D.Shepherd y L. Pollock. “Interfaces, Aspects, and Views”. In Proceedings of Linking Aspect Technology and Evolution Workshop (2005).

[18] C. Zhang y H. Jacobsen. “Efficiently mining crosscutting concerns through random walks”. In AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development. New York, NY, USA: ACM Press (2007) pp. 226-238.

[19] M. Lorenz y J. Kidd. “Object-Oriented Software Metrics: A Practical Guide”. P T R Prentice Hall, Prentice-Hall, Inc. A Pearson Education Company, 146 pages (1994).

[20] K. Gybels y A.Kellens. "Experiences with identifying aspects in smalltalk using 'unique methods'," in Linking Aspect Technology and Evolution (LATE), collocated with Aspect-Oriented Software Development, 2005.

[21] D. Shepherd et al. “Using natural language program analysis to locate and understand action-oriented concerns”, In AOSD '07: Proceedings of the 6th International Conference on Aspect-Oriented Software Development (2007), pp. 212-224.

[22] D. Shepherd et al., "Timna: a framework for automatically combining aspect mining analyses," in ASE '05: Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. New York, NY, USA: ACM, 2005, pp. 184-193.

[23] G. Cojocar y G. Serban., "On some criteria for comparing aspect mining techniques," in LATE '07: Proceedings of the 3rd workshop on Linking aspect technology and evolution. New York, NY, USA: ACM, 2007.

[24] M. Marin; et al. "An integrated crosscutting concern migration strategy and its automated application to JHotDraw," Automated Software Engineering, vol. 26, no. 2, pp. 323-356, 2009.

APENDICE

Tabla 4: Vocabulario generado del software MiPitanza

		Lugar				Referencia			Categoría			
método clase soporte	**TI**	**U**	**I**	**F**	**M**	**V**	**A**	**S**	**IS**	**IC**	**BI**	**BIC**
Display.setCurrent	18	2	7	9			6	12	18			
DataInputStream.readUTF	5				5		5					5
DataOutputStream.writeUTF	5				5		5				5	
Alert.addCommand	3		2	2			3				3	
Alert.Alert	3		1		2		3		1		2	
Alert.setString	3		2		1		3				3	
Alert.setTimeout	3		2		1		3				3	
Alert.setType	3		2		1		3				3	
ByteArrayInputStream.reset	3		1		2		3			3		
InvalidRecordIDException	3			3						3		
RecordStored.getStore	3		2		1	3			2			1
Alert.setCommandListener	2		1	1			2				2	
DataInputStream.reset	2				2		2					2
UIEditDetails.edit	2	1		1			2		2			
UIFilter.show	2	2					2		2			
UIFilter.UIFilter	2	1			1		2		2			
UIList.show	2	2					2		2			
UIMainMenu.UIMainMenu	2	1			1		2		2			
Alert.setTitle	1		1				1				1	
ByteArrayOutputStream.reset	1				1		1		1			
ByteArrayOutputStream.toByteArray	1				1		1		1			
Connector.open	1		1					1	1			
DataInputStream.readInt	1				1		1					1
DataOutputStream.flush	1				1		1				1	
DataOutputStream.writeInt	1				1		1				1	
Display.getDisplay	1				1		1		1			
System.gc	1		1					1	1			
TextMessage.setPayLoadText	1				1	1			1			
UIAbout.UIAbout	1				1		1		1			
UIEditDetails.UIEditDetails	1				1		1		1			
UIList.UIList	1				1		1		1			
UIShowDetails.show	1	1					1		1			
UIShowDetails.UIShowDetails	1	1					1		1			
RecordStored.addRecord	1				1	1				1		
RecordStored.deleteRecord	1		1			1						1
RecordStored.setRecord	1				1	1				1		