Acoplamiento estructural y acoplamiento lógico

Daniel Bertarioni Chaves, Yelko Carvajal Mora y Antonio González Torres
Facultad de Tecnologías de la Información
ULACIT, San José, Costa Rica
{dbertarionic617, ycarvajalm858}@ulacit.ed.cr y
{agonzalez}@ulacit.ac.cr

***Resumen*—El acoplamiento es una métrica que se utiliza como indicador de la calidad del diseño de un software y de la dificultad para darle mantenimiento. Sin embargo, el alto nivel de acoplamiento en un sistema no siempre conlleva inconvenientes, si los elementos acoplados se encuentran en un nivel avanzado de madurez. En este contexto, la combinación del acoplamiento indirecto con el acoplamiento lógico puede ser de gran utilidad para identificar componentes que, a pesar de su alto acoplamiento, no representan un problema de mantenimiento, debido a que su frecuencia de cambio es baja y se encuentran en un estado de madurez que muestra estabilidad. Como consecuencia, este trabajo de investigación propone un método para identificar las cadenas de acoplamiento indirecto, determinar el acoplamiento lógico de sus elementos y combinar ambos tipos de acoplamiento, con el fin de brindar detalles sobre la estabilidad de los componentes y el cambio conjunto de estos.**

***Index Terms*—**

I. Introducción

Las características de los sistemas de software se representan en la interfaz de usuario utilizando opciones de menú o botones. Estos elementos de interfaz están enlazados a un método que invoca a una o varias funciones, las cuales pueden estar encadenadas a otros métodos. La secuencia de llamadas de las funciones forman cadenas de acoplamiento indirecto, y todas juntas conforman un grafo. Estas dependencias entre elementos no son visibles a simple vista y los programadores requieren examinar los componentes del sistema para generar las cadenas de acoplamiento indirecto de forma manual.

El acoplamiento indirecto es la relación transitiva entre dos elementos, A y C, de tal forma que si A está directamente acoplado a B y B está acoplado de forma directa a C, entonces A y C están acoplados de forma indirecta [1]. Esto es trascendental en el paradigma de programación orientado a objetos, porque la reutilización de código a partir de pequeños componentes que se relacionan e interactúan es fundamental. Los sistemas desarrollados bajo este modelo evitan usar grandes elementos monolíticos que operan de forma aislada y autosuficiente.

Los altos niveles de acoplamiento son indicadores de la calidad del diseño de los sistemas [2], y sirven como parámetros para la predicción de fallas del software. El alto acoplamiento entre elementos se asocia con dificultades para realizar el mantenimiento de los programas. Sin embargo, esto no es un inconveniente si los elementos fuertemente acoplados se encuentran en una fase avanzada de madurez debido a que sufren pocos cambios y son estables [3].

En tanto, el acoplamiento lógico es la dependencia entre componentes, de acuerdo con su coevolución. Los grupos de elementos acoplados de forma lógica se crean de acuerdo con la probabilidad de que al realizar cambios en un ítem de la agrupación, los demás también deban ser modificados. La identificación de los grupos de elementos que están acoplados de forma lógica es más difícil de percibir que los métodos acoplados indirectamente.

El acoplamiento lógico permite determinar los componentes que tienen alguna relación no estructural, pero que requieren ser modificados de forma conjunta cuando se cambia alguna funcionalidad del sistema. Reconocer estos grupos conlleva crear una lista de los elementos que cambiaron en cada *commit* del software y comparar la frecuencia de los que han cambiado juntos. Así, la dificultad de este proceso se incrementa de acuerdo con el historial y la cantidad de elementos del programa.

La combinación del acoplamiento indirecto y el acoplamiento lógico puede ser de gran utilidad para identificar los componentes de las cadenas de acoplamiento indirecto que se encuentra en una etapa de madurez. Sin embargo, tanto los elementos que conforman el acoplamiento indirecto y como el lógico no se pueden observar a simple vista. La identificación del acoplamiento indirecto requiere extraer las cadenas que conforman las relaciones entre elementos usando análisis estático o dinámico en los programas, mientras que el determinar los grupos de elementos acoplados lógicamente se realiza a partir de los patrones de cambio conjunto de los componentes en el tiempo.

El tiempo de ejecución que requiere extraer las cadenas de acoplamiento indirecto y determinar los grupos de elementos acoplados de forma lógica es proporcional, en el primer caso, a la complejidad interna de los sistemas; y en el segundo, al tamaño del sistema y el tiempo durante el cual ha evolucionado. El método utilizado para extraer el acoplamiento indirecto a nivel de métodos es el propuesto por Navas-Sú y González-Torres [1], y el procedimiento para obtener el acoplamiento lógico entre los componentes es parte del trabajo realizado en esta investigación.

Como consecuencia, este trabajo de investigación propone un procedimiento para identificar los grupos de métodos acoplados lógicamente en una cadena de acoplamiento indirecto, así como para medir la intensidad de los cambios que se les están realizando a lo largo del tiempo. Lo que se pretende es determinar la estabilidad de los métodos fuertemente acoplados de acuerdo con la frecuencia de los cambios que se les realizan, y establecer cuáles están siendo modificados

de forma conjunta. Con esto se busca facilitar las tareas de mantenimiento y reingeniería del código fuente.

II. Antecedentes

Edward Yourdon y Larry Constantine definen el acoplamiento como el nivel de dependencia entre los elementos de un sistema, de forma que entre más alto es el valor del acoplamiento, menor es la independencia de los elementos [4]. La relación de dependencia aumenta las probabilidades de errores inesperados y la dificultad para entender el funcionamiento del código fuente. Esto compromete la mantenibilidad, debido a que realizar cambios en un elemento específico tiene mayor complejidad.

Posteriormente, Chidamber y Kemerer incorporaron la definición del acoplamiento a su propuesta de métricas para el diseño de un sistema orientado a objetos [5]. En la actualidad, el acoplamiento se clasifica en estructural, lógico, dinámico y semántico. En el presente trabajo, el acoplamiento estructural y lógico son de especial interés y se explican a continuación.

La medición del acoplamiento estructural está asociada al número de elementos que intervienen durante la realización de una tarea específica, lo cual implica que un módulo fuertemente acoplado está relacionado a un gran número de componentes, debido a la interconexión entre estos [5]. La identificación de las relaciones entre los componentes permite que los desarrolladores determinen los métodos para realizar cambios y efectuar pruebas una vez que han realizado las modificaciones. Estas relaciones se clasifican en acoplamiento de datos, común, control, sello y contenido.

Este tipo de acoplamiento puede ser directo o indirecto, y es ilustrado en la figura 1. Las clases *User* y *Song* tienen una relación de acoplamiento directo, de forma similar a las clases *Song* y *Artist*; mientras que las clases *User* y *Artist* tienen una relación transitiva por medio de *Song*, por lo que tienen una relación de acoplamiento indirecto: el método *getRecommendations* de la clase *User* depende del procesamiento que realiza el método *getRelatedSongs* de la clase *Artist*, el cual es invocado a través del método *getRecommendedSongs* de la clase *Song*. El acoplamiento indirecto implica que un cambio en el componente independiente afecta al elemento dependiente, por lo que en el ejemplo de la figura 1 los cambios que se realicen en el método *getRelatedSongs* de la clase *Artist* afectan el funcionamiento del método *getRecommendations* de la clase *User*.

Las secuencias de acoplamiento indirecto de un sistema conforman un grafo dirigido. Cada cadena inicia con un elemento que solo tiene relaciones salientes y termina en un componente que no llama a otros ítems. Los elementos que forman una cadena también pueden ser parte de otras cadenas. La principal contribución del acoplamiento indirecto es exponer el impacto que tiene la modificación de un elemento en otros.

El acoplamiento lógico se basa en identificar los objetos que cambian o coevolucionan simultáneamente en el tiempo. Esto es ilustrado en la figura 2, en la cual se puede observar que los archivos *Song* y *Artist* han cambiado en los *Commit 1*, *Commit 3* y *Commit 4*, en tanto que el archivo *Playlist* cambió en dos

Figura 1. Acoplamiento indirecto.

ocasiones junto con los archivos *Song* y *Artist* en *Commit 1* y *Commit 4*. Por lo tanto, el nivel de acoplamiento entre *Song* y *Artist* es más alto en relación con *Playlist*, y la probabilidad de que alguno de los dos primeros archivos sea modificado cuando cualquiera de ambos es cambiado es más alta.

La información necesaria para la extracción del acoplamiento lógico se obtiene de los *commit* almacenados por los sistemas de control de versiones, como *Subversion* o *Git*.

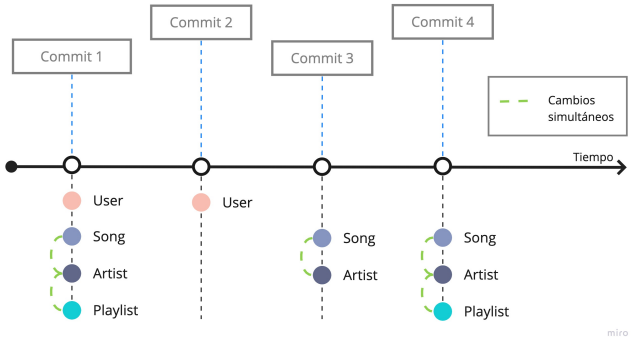

Figura 2. Acoplamiento lógico.

III. Combinación de acoplamiento estructural y lógico

El proceso para la identificación de los grupos de métodos de una cadena de acoplamiento indirecto que están acoplados lógicamente se muestra en la figura 3 y se describe a continuación:

Extracción del acoplamiento indirecto: La extracción de las cadenas de elementos acoplados de forma indirecta

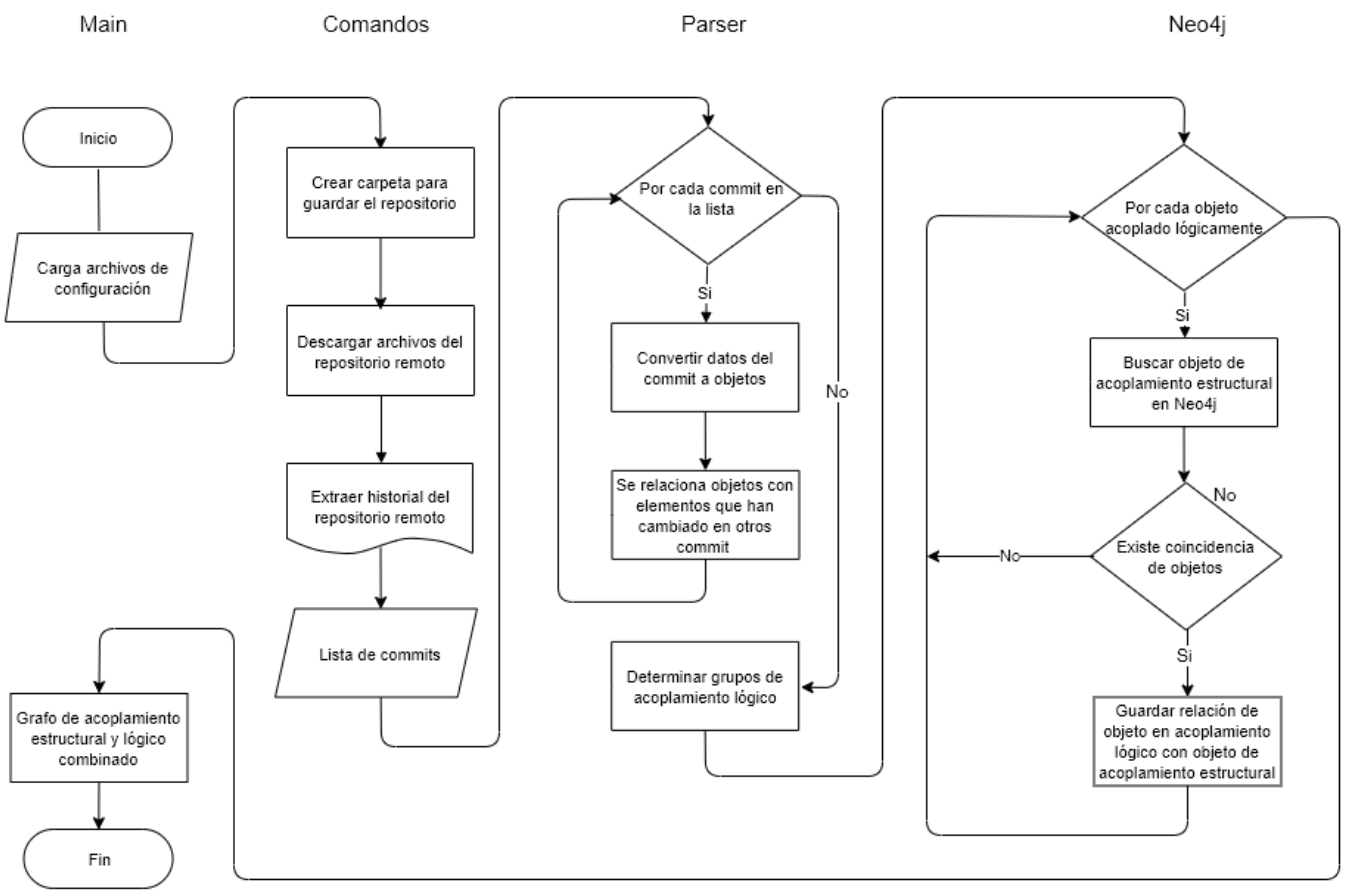

Figura 3. Diagrama de flujo del proceso.

se realizó utilizando el método de Navas-Sú y González-Torres [1]. Las cadenas de acoplamiento se almacenan en una base de datos gestionada por Neo4J.

Identificación del acoplamiento lógico: La identificación de los grupos de componentes acoplados lógicamente se realiza siguiendo el proceso descrito en la figura 3. El reconocimiento de estos grupos contempla dos etapas: descarga de información del repositorio del sistema de control de versiones (e.g., Git, Subversión y TFS) y cálculo del acoplamiento lógico.

Descarga de información del repositorio: Los pasos que se llevan a cabo son los siguientes:

1. Conexión a un repositorio usando una ruta y permisos especificados previamente.
2. Creación de una carpeta para guardar el repositorio.
3. Descargar los archivos del repositorio remoto.
4. Extracción del historial de los archivos del repositorio remoto y almacenamiento en el archivo *log.txt*.
5. Creación de la lista de *commits*.

Cálculo del acoplamiento lógico: Requiere efectuar los siguientes pasos por cada elemento en la lista de *commit*:

1. Se convierten los datos de cada *commit* en un objeto. Para esto se realiza el recorrido del archivo *log.txt* y por cada *commit* se crea un objeto.
2. Se relacionan los objetos con otros elementos a los que se han efectuado cambios en *commits* previos.

Combinación del acoplamiento indirecto y el lógico: Esta fase se encarga de determinar los elementos de la cadena de acoplamiento indirecto que se encuentran acoplados de forma lógica. El proceso lee cada objeto de la lista de *commits* y busca un elemento correspondiente en la base de datos de Neo4j. Si existe una correspondencia, se realiza la combinación de ambos tipos de acoplamiento y se almacena la información con la asociación en Neo4j.

IV. Visualización de resultados

En esta sección se presenta una propuesta simple de visualización que representa la combinación del acoplamiento estructural y el acoplamiento lógico. La prueba de concepto realizada es un *frontend* construido en *Angular*. El código del módulo para conectarse y descargar el código de un repositorio de Github se puede descargar de https://github.com/cincottel5/git-2-neo4j, mientras que el repositorio del *frontend* se encuentra en https://github.com/cincottel5/coupling-frontend.

La representación visual busca mostrar las dependencias entre los elementos de un sistema, y los grupos de componentes que son modificados de forma conjunta en una cadena

Figura 4. Cambio histórico de nodos.

Figura 5. Coevolución perteneciente a la misma clase.

de llamadas. Así, se usa un grafo dirigido para mostrar las dependencias entre elementos y distintos colores para definir los grupos de acoplamiento lógico.

En la figura 4 se resalta en azul un método como punto de entrada para generar la cadena de acoplamiento. El resto de nodos utilizan diferentes colores en sus bordes. Los elementos cuyo borde es de color anaranjado tienen una relación de acoplamiento lógico, aquellos con una parte de su borde en color rojo están relacionados entre sí y los que tienen parte de su borde coloreado en azul también tienen una asociación de acoplamiento lógico.

Por otra parte, la figura 5 muestra un ejemplo en el cual el método de la esquina superior derecha del proyecto NEO4J.Net.Driver (https://neo4j.com/developer/dotnet) es el punto de inicio de la cadena (grafo) de acoplamiento indirecto. Cuando se pasa el *mouse* por un nodo, los elementos que se encuentran en la misma clase se resaltan en rosado, los que están acoplados de forma lógica se pintan de blanco y el nodo final solo es llamado, pero no invoca a otros, se pone de color negro. Así la visualización muestra el acoplamiento indirecto, los grupos de acoplamiento lógico y la cohesión de los métodos. Además, la representación permite que el usuario filtre los nodos por nombre, tipo de relación o nivel de profundidad del grafo.

V. Conclusiones

La prueba de concepto efectuada permitió identificar las cadenas de acoplamiento indirecto en los sistemas, determinar el acoplamiento lógico de los elementos, correlacionar el acoplamiento indirecto con el acoplamiento lógico y representar visualmente los resultados.

Con base en la experiencia de los autores, se puede señalar que esta información puede facilitar que los desarrolladores identifiquen las dependencias estructurales y lógicas entre los elementos. Esto es de utilidad para hacer cambios durante las tareas de mantenimiento. Asimismo, los encargados de las pruebas del software pueden seguir los elementos de una cadena de acoplamiento para probar cada elemento y su integración en una funcionalidad. Adicionalmente, los detalles que proporciona la visualización pueden contribuir con la toma de decisiones y la priorización de tareas.

Las cadenas de acoplamiento indirecto combinadas con la información de acoplamiento lógico permiten determinar la estabilidad de una clase, en función de los cambios que se les realizan. Además, el método propuesto permite visibilizar las relaciones entre elementos y su patrón de evolución, y puede servir como fundamento para pronosticar futuros cambios en el código fuente, mediante la adición de análisis predictivo a la coevolución de los componentes de software.

Referencias

[1] A. G.-T. José Navas-Sú, "A method to extract indirect coupling and measure its complexity," *2018 International Conference on Information Systems and Computer Science (INCISCOS)*, 2018.

[2] M. Fowler, "Reducing coupling," *IEEE Software*, vol. 18, no. 4, pp. 102–104, July 2001.

[3] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, Nov 2009.

[4] E. Yourdon and L. L. Constantine, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, 1st ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979.

[5] C. F. K. Shyam R. Chidamber, "Towards a metric for object oriented design," *ACM SIGPLAN Notices*, vol. 26, no. 11, pp. 197 – 211, 1991.