Détection des préoccupations transversales au niveau architectural

Fairouz DAHI, Nora BOUNOUR

Laboratoire LRI, Département d'Informatique, Université Badji-Mokhtar
BP. 12, 23000, Annaba, ALGÉRIE
fairouz_dahi@yahoo.fr, nora_bounour@yahoo.fr

RÉSUMÉ. L'existence des préoccupations transversales enchevêtrées ou dispersées, rend complexe la compréhension et l'évolution du code source orienté objet. Le paradigme objet ne prend pas en charge ces préoccupations transversales. Pour remédier à ces insuffisances, l'adoption industrielle du paradigme orienté aspect a donné naissance à la recherche d'approches et d'outils supportant la migration orientée aspect des systèmes logiciels déjà implémentés. Cette migration nécessite l'identification des préoccupations transversales.

Nous nous intéressons, dans cet article, à l'identification des préoccupations transversales au niveau architectural. Le modèle architectural considéré est matérialisé par les diagrammes UML de classes et de séquences. Nous proposons une nouvelle approche qui se base sur l'analyse des concepts formels.

ABSTRACT. The existence of crosscutting concerns tangled or scattered, complicates the understanding and evolution of object-oriented source code. The object oriented paradigm doesn't support these crosscutting concerns. To remedy to these shortcomings, the adoption of industrial aspect-oriented paradigm has led to research approaches and tools supporting aspect-oriented migration of legacy systems. This migration requires the identification of crosscutting concerns.

We focus in this paper on the identification of crosscutting concerns at the architectural level. The architectural model considered is represented by class and sequence UML diagrams. We propose a new approach based on the formal concepts analysis.

MOTS-CLÉS : Analyse des concepts formels, Analyse d'ordre de transmissions des messages, Architecture logicielle, Aspect mining, Préoccupations transversales, Rétro-ingénierie, UML.

KEYWORDS: Formal concept analysis, Message transmissions order analysis, Software architecture, Aspect mining, Crosscutting concerns, Reverse engineering, UML.

1. Introduction

Toutes les études en génie logiciel ont montré que la maintenance et l'évolution sont les phases les plus longues et les plus coûteuses dans le cycle de vie du logiciel [9]. Cela

est du à l'impératif de satisfaire de nouvelles exigences des utilisateurs (commerciales, technologiques, etc.), ou modifier celles existantes, afin d'éviter la dégradation de l'efficacité du logiciel. A cet effet, la maintenance des logiciels a fait l'objet d'une grande attention ces dernières années. Elle est utilisée pour apporter des modifications sur le logiciel déjà implémenté. Mais auparavant, on doit faciliter le processus de compréhension des logiciels existants, afin d'en améliorer la maintenance.

En général, les préoccupations transversales peuvent se caractériser par la dispersion (Scattering) ou l'enchevêtrement (Tangling). La dispersion se produit quand le code d'une même fonctionnalité est distribué à travers le système, tandis que l'enchevêtrement, c'est quand deux ou plusieurs fonctionnalités sont contenues dans une même autre. Le paradigme orienté objet ne prend pas en charge ces préoccupations transversales. Le paradigme orienté aspect est une alternative pour supporter la structure améliorée du logiciel, en fournissant une bonne modularisation des préoccupations transversales.

Afin de bénéficier de l'orienté aspect, en migrant les systèmes objet existants (legacy system) vers ce paradigme [4], il est nécessaire de développer des approches et des outils qui identifient et séparent les préoccupations transversales. L'identification des préoccupations transversales au niveau code source souffre de certaines limites [1, 5]. Le code source est lié à une plateforme spécifique. En outre, sa compréhension demeure difficile, du fait de sa complexité et sa taille. Ainsi, il existe un manque d'utilisation de l'information sémantique par les approches existantes, notamment la détection des deux symptômes de la transversabilité (enchevêtrement et dispersion) à la fois.

Afin de dépasser ces insuffisances, nous proposons dans cet article une nouvelle approche d'identification des préoccupations transversales au niveau architectural. L'architecture du logiciel constitue un niveau plus abstrait, indépendant de la technologie d'implémentation. Nous la matérialisons par les diagrammes UML de classes et celui de séquence [14]. Ces derniers peuvent être générés facilement à partir du code source du système [10, 11, 12].

Le diagramme de séquence est un modèle conceptuel, constituant une version proche de la version finale qui représente l'ensemble des fonctionnalités du système. En d'autre terme, les objets et les messages du diagramme de séquences d'un système reflètent, respectivement, les objets et les invocations des méthodes implémentées de ce système.

Nous proposons une approche hybride à deux passes. En une première passe, nous utilisons l'analyse des concepts formels, qui est une méthode de regroupement conceptuelle, afin de regrouper les fonctionnalités dispersées. Dans une deuxième passe, nous analysons l'ordre de transmissions de messages, afin de détecter les fonctionnalités enchevêtrées. Nous nous proposons ensuite d'effectuer une fusion des résultats de ces deux analyses, afin de filtrer l'ensemble des aspects candidats obtenus, et garder ceux jugés pertinents.

La section 2 décrit notre approche d'identification des préoccupations dispersées et enchevêtrées, ainsi que le filtrage de ces préoccupations. Un ensemble des travaux connexes feront l'objet de la section 3. Cette dernière inclut une conclusion qui nous permettra, en dernier lieu, d'ouvrir des perspectives pour des développements futurs.

2. Vers une nouvelle approche d'identification de la transversabilité au niveau architectural

Nous considérons l'architecture logicielle décrite via les diagrammes de classes et de séquence UML. Ce dernier est devenu une référence incontournable dans le domaine du génie logiciel. Sa richesse et sa puissance d'expression le rendent également éligible pour la modélisation [8].

Nous allons utiliser le diagramme de classes, afin de cerner l'ensemble des méthodes qui existent dans le système, et le diagramme de séquence pour détecter la transversabilité, car ce diagramme fournit des informations sémantiques sur les relations de transmissions de messages entre les objets du système, et l'ordre chronologique d'exécution de ses tâches. Nous allons analyser le diagramme de séquence par l'analyse de concepts formels (ACF). L'ACF nous permet de détecter la dispersion dans les diagrammes de séquence.

3.1. Principe de L'ACF

L'ACF ramène les données d'une relation binaire, et exploite les propriétés de la correspondance, afin d'en isoler une hiérarchie de concepts, dits formels [6]. Trois principaux éléments interviennent dans l'ACF: le contexte formel, le concept et le treillis des concepts. Le contexte de l'ACF est représenté par le triplet *(O, A, R)*, où *O* désigne l'ensemble d'objets, *A* est l'ensemble des attributs, et *R* décrit la relation qui relie les objets à leurs attributs. Un concept est la paire *(X, Y)*, où *X* est un ensemble d'objets, et *Y* les attributs de ces objets. Un concept associe un ensemble maximal d'objets, à l'ensemble d'attributs que ces objets partagent. Le treillis de concepts (treillis de Galois) est une hiérarchie de concepts. Ces derniers sont ainsi organisés : un attribut, présent dans un concept, est hérité par tous les sous-concepts, et inversement pour les objets.

2.2. Détection de la dispersion

Les interactions (messages transmis) entre les différents objets du diagramme de séquence reflètent les invocations de méthodes du système. Ces méthodes existent dans le diagramme de classes UML de ce système. Un diagramme de séquence capture le

comportement d'un seul scénario de cas d'utilisation, pour cette raison, nous utilisons l'ensemble des diagrammes de séquences du système. Nous utilisons l'ACF afin de détecter les fonctionnalités dispersées, en exploitant les messages transmis entre les objets dans les diagrammes de séquence. Pour le contexte *(O, A, R)* de l'ACF, nous utilisons l'ensemble des objets *O* pour représenter les objets du système qui existent dans les diagrammes de séquence, et l'ensemble des attributs *A* pour représenter toutes les méthodes du diagramme de classes, invoquées par ces objets (méthodes qui correspondent aux messages dans les diagrammes de séquence). La relation binaire *R* désigne la relation d'invocation entre les objets et leurs méthodes, représentées par la transmission des messages dans les diagrammes de séquence.

Nous construisons une matrice binaire *M*, ses lignes contiennent les noms des objets, et ses colonnes contiennent les noms des méthodes, tel que : si l'objet *i* transmet le message qui invoque la méthode *j*, alors *M[i,j]=1*, sinon *M[i, j]=0*. Dans un concept *(X, Y)*, *X* est un ensemble des objets appelants, et *Y* est l'ensemble des méthodes invoquées par ces objets. Le treillis de concepts regroupe les méthodes par invocations. Il nous aide à déterminer les méthodes (les tâches du système) dont l'invocation est dispersée.

Les aspects candidats sont tous les attributs (méthodes invoquées) de tous les concepts du treillis, qui ont le nombre de leurs objets supérieur à un certain seuil (métrique fan-in >1 [7]), car la refactorisation orientée aspect d'une méthode invoquée peu de fois n'améliore pas la structure du système (la modification de la manière de son invocation n'est pas coûteuse). Pour la même raison, une méthode dont l'invocation n'est pas dispersée, c'est-à-dire elle est invoquée par le même objet ou par l'objet qui la contient, n'est pas un aspect candidat.

2.3. Détection de l'enchevêtrement

L'ordre des messages transmis entre les objets dans les diagrammes de séquence reflète l'ordre chronologique d'exécution des tâches du système. Il est utile donc d'analyser l'ordre chronologique de transmissions des messages dans les diagrammes de séquence, afin de détecter les préoccupations enchevêtrées.

L'enchevêtrement est détecté lors de l'existence d'un modèle récurrent d'invocation des méthodes de type avant « *before* », après « *after* » ou autour « *around* » (le type concerne le moment d'exécution de la consigne d'aspect [4]) comme suit :

– Si, dans les diagrammes de séquence, le message qui invoque la méthode *A* est transmis toujours avant celui qui invoque la méthode *B*, il y aura un aspect de type *before*, dont la méthode *A* est sa consigne, et *B* son point de coupure.

- Si le message qui invoque la méthode *A* est transmis toujours après celui qui invoque la méthode *B*, il existe un aspect de type *after*, dont la méthode *A* représente sa consigne, et la méthode *B* représente son point de coupure.
- Si le message qui invoque la méthode *A* est transmis toujours avant et après celui qui invoque la méthode *B*, il existe donc un aspect de type *around*, dont la méthode *B* représente son point de coupure, et la méthode *A* représente sa consigne.
- Si le message qui invoque la méthode *A* est transmis toujours avant celui qui invoque la méthode *B*, et ce dernier est transmis toujours après le message qui invoque la méthode A, l'aspect *before* dont *A* représente la consigne et l'aspect *after* dont *B* représente la consigne sont symétriques. En d'autre terme, nous disons qu'il existe un aspect *before* (*A* est sa consigne et *B* son point de coupure) ou il existe un aspect *after* (*B* est sa consigne et *A* son point de coupure).
- Si le message qui invoque la méthode *A* est transmis toujours avant celui qui invoque la méthode *B* et /ou *C*, alors il y a un aspect *before*, dont sa consigne est la méthode *A*, et ses points de coupures sont les méthodes *B* et *C*. Les points de coupures peuvent, par conséquent, être un ensemble de méthodes invoquées.

Notre analyse d'ordre d'invocations de méthodes facilite la refactorisation orientée aspect [3], en précisant la consigne de l'aspect et ses points de coupure.

2.4. Filtrage des aspects candidats obtenus

La dispersion et l'enchevêtrement représentent les deux symptômes indiquant l'existence des préoccupations transversales. La détection d'un symptôme sans l'autre affaiblit la précision.

Prenons un petit exemple représentant un ensemble de messages transmis entre les objets d'un système dans le diagramme de séquence, en respectant l'ordre de transmissions. Soit {*F, A, B, C, D, E, T*} l'ensemble d'objets, et {*s, x, y, z, a, b, r*} l'ensemble de méthodes existant dans le diagramme de classes de l'exemple, et qui sont invoquées par les messages transmis dans les diagrammes de séquence du même exemple. La séquence des transmissions des messages est :

{ F.s, A.x, B.y , C.z , A.a, D.b, E.x, C.r, E.s, A.x, T.y, A.a, D.s }

En analysant cet exemple avec l'ACF, nous obtiendrons les résultats suivants :

- La méthode *x* a deux invocations dispersées (invoquée par *A* et *E*).
- La méthode *y* a deux invocations dispersées.
- La méthode *s* a trois invocations dispersées.

- La méthode *a* est invoquée seulement par l'objet *A*, donc elle n'est pas considérée comme aspect candidat.
- Les méthodes {z, *b, r*} sont invoquées une seule fois, et par conséquent, elles ne sont pas considérées comme des aspects candidats.

Nous constatons que les aspects candidats obtenus par l'ACF sont les méthodes ***{x, y, s}***. L'utilisation unique de l'analyse des concepts formels pour détecter les préoccupations transversales, semble être insuffisante, car elle ne prend pas en compte le symptôme de l'enchevêtrement. En outre, elle ne fait pas la distinction entre une consigne et un point de coupure, car elle considère les deux comme des aspects candidats s'ils ont des diverses invocations. Or, une méthode invoquée plusieurs fois peut être un point de coupure, et non pas la consigne de l'aspect.

Analysons maintenant le même exemple avec l'analyse d'ordre d'invocations. Les méthodes *{x, y}* constituent un modèle récurrent de type *before*, car la méthode *y* est invoquée après l'invocation de la méthode *x.* Par conséquent, le seul aspect candidat obtenu par l'analyse d'ordre d'invocations est de type *before*, dont la méthode *x* est sa consigne, et ses points de coupure sont les méthodes {*y, r*}.

Une méthode dont son invocation est dispersée (en plusieurs fois dans des contextes différents), mais n'appartenant pas à un modèle récurrent de type *before*, *after* ou *around*, n'est pas considérée, par l'analyse de l'ordre d'invocations, comme un aspect. Or, la modification de la manière d'invoquer cette méthode entraine des changements coûteux dans les objets qui l'invoquent. Par conséquent, l'analyse d'ordre d'invocations ne prend pas en compte le symptôme de la dispersion. Concernant l'exemple précédent, la méthode *s* n'est pas considérée par l'analyse d'ordre d'invocations comme un aspect, or c'est une méthode dispersée, et la modification de son invocation entraine des modifications couteuses dans les objets {*F, E, D*}. Par conséquent, la méthode *s* est un aspect candidat qui n'a pas été détecté par l'analyse d'ordre d'invocations.

Les deux analyses présentées dans les sections 2.2 et 2.3 semblent être complémentaires. L'utilisation de l'analyse des concepts formels et celle d'ordre d'invocations, afin de détecter les préoccupations dispersées et celles enchevêtrées, respectivement, augmente la précision, et diminue les faux aspects.

En filtrant les aspects candidats obtenus par les deux approches : analyse des concepts formels et analyse d'ordre d'invocations, afin d'éliminer les faux aspects et garder ceux les plus pertinents, nous constatons que la méthode *y* est un faux aspect obtenu par l'ACF, et la méthode *s* est un aspect qui n'a pas été détecté par l'analyse d'ordre d'invocations. Les aspects candidats obtenus seront l'aspect dont la consigne est la méthode ***s,*** et l'aspect dont la méthode ***x*** est sa consigne et ses points de coupure sont les méthodes {*y, r*}.

3. Travaux connexes et Conclusion

L'analyse des concepts formels (ACF) a été appliquée dans le domaine d'aspect mining, afin d'identifier les aspects candidats au niveau code source. L'ACF des noms de classes et de méthodes [6] vise à regrouper ces dernières selon leurs sous-chaines communes, or les noms peuvent ne pas être significatifs (dépendent du programmeur) et l'approche détecte seulement la dispersion comme symptôme. L'ACF des traces d'exécution [13] regroupe les méthodes selon les cas d'utilisation qui les invoquent pendant la génération des traces d'exécution. Les traces d'exécution peuvent être volumineuses et l'approche détecte seulement le symptôme d'enchevêtrement. L'ACF des relations d'appels des méthodes, proposée dans notre travail antérieur [2], exploite les relations d'appels entre les méthodes, et regroupe ensuite ces dernières par classes appelantes. Cette approche ne détecte que la dispersion.

Outre l'ACF, l'approche dans [15] a exploité les invocations dispersées et enchevêtrées des méthodes. Initialement, l'auteur a identifié les préoccupations transversales statiques grâce aux relations statiques liées à des déclarations dispersées et enchevêtrées des méthodes et des types (générées à partir des relations d'héritage, implémentation…) [16]. Avec ces relations statiques, l'auteur définit ensuite les préoccupations du système. Lorsqu'une méthode d'une préoccupation (identifiée précédemment) est invoquée par plusieurs méthodes appartenant à différentes préoccupations, la préoccupation invoquée, dans ce cas, est dynamiquement dispersée dans les préoccupations associées aux appelantes (préoccupation transversale dynamique). Egalement, lorsqu'une méthode appelle d'autres méthodes appartenant à différentes préoccupations, alors les comportements de telles préoccupations sont enchevêtrées ensembles dans la méthode appelée. Cette approche n'est pas complète, car les préoccupations transversales qui n'ont pas une structure statique ne seront pas prises en compte.

Dans ce papier, Nous avons proposé une approche d'identification des préoccupations transversales au niveau architectural. Nous visons la détection des deux symptômes de la transversabilité, par l'utilisation d'une approche hybride combinant les deux analyses : l'analyse des concepts formels et l'analyse d'ordre d'invocations des méthodes, et de filtrer ensuite l'ensemble des aspects candidats obtenus par ces deux analyses, afin de garder ceux jugés pertinents.

En se basant sur les diagrammes de séquence, nous exploitons d'une part les relations d'invocations des méthodes et d'autre part l'ordre chronologique d'exécution des tâches du système. Il nous a été possible de dépasser les limites des approches existantes de l'identification des préoccupations transversales au niveau code source, par l'utilisation des informations sémantiques du système et la détection des deux symptômes de la transversabilité, à un niveau plus abstrait que le code source.

La perspective ouverte par notre travail se situe au niveau du développement d'un outil mettant en œuvre notre approche, afin de la valider et l'évaluer.

4. Bibliographies

[1] Dahi, F., Bounour, N., *Etude critique des approches de découverte d'aspect à travers le cycle du développement de logiciels*, 2nd International Conference on Complex Systems, Jijel, Algeria, 2011.

[2] Dahi, F., Bounour, N., *Identification d'aspects par l'analyse des concepts formels*, 1st International Conference on Information Systems and Technologies, Tebessa, Algeria, 2011.

[3] Deursen, A., Marin, M.,Moonen, L., *Aspect Mining and Refactoring*, In Proceedings First International Workshop on REFactoring: Achievements, Challenges, 2003.

[4] Pawlak, R. ,Retaillé, J.P., Seinturier, L., *Programmation orientée aspect pour Java/J2EE*, Eyrolles, 2004.

[5] Mens , K., Kellens, A., Krinke, J., *Pitfalls in Aspect mining* , Proceeding WCRE'08 Proceedings of theA15th Working Conference on Reverse Engineering, 2008 .

[6] Tourw´e, T. , Mens, K., *Mining aspectual views using formal concept analysis*, In 4th IEEE International Workshop on Source Code Analysis and Manipulation, pp. 97–106, 2004.

[7] Marin, M., Deursen, A., Moonen, L., *Identifying aspects using fan-in analysis*, In Proceedings of the 11th Working Conference on Reverse Engineering, 2004.

[8] Siau K., Cao, Q., *Unified Modeling Language*: *A Complexity Analysis*, journal of database management, Vol. 12, 2001.

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

[10] http://bouml.free.fr/screenshots.html

[11] http://staruml.sourceforge.net/en/

[12] http://www.umlgraph.org/ver.html

[13] Tonella, P., Ceccato, M., *Aspect mining through the formal concept analysis of execution traces*, In Proceedings of the 11th Working Conference on Reverse Engineering, 2004.

[14] Su, Y., Li, F., Hu, S., Chen, P., *Aspect-oriented software reverse engineering*, Journal of Shanghai University, Vol. 10, number 5, 2006.

[15] Bernardi, M.L., Lucca, G.A., *Identifying the Crosscutting among Concerns by Methods' Calls Analysis*, Springer, Vol. 257, pp. 147-158, 2011.

[16] Bernardi, M.L., Lucca, G.A., *Analysing Object Type Hierarchies to Identify Crosscutting Concerns*, Springer-Verlag Berlin Heidelberg , pp. 216–224, 2009.