Exploration de la redistribution des classes d'un package par Analyse Formelle de Concepts

Lala Madiha Hakik
Lirmm, France et Univ Hassan I, Maroc
lalamadihah@gmail.com

Marianne Huchard
Lirmm, Université Montpellier 2 et CNRS, France
huchard@lirmm.fr

Rachid El Harti
Université Hassan I, Settat, Maroc
relharti@gmail.com

Abdelhak-Djamel Seriai
Lirmm, Université Montpellier 2 et CNRS, France
Abdelhak.Seriai@lirmm.fr

Résumé

Du fait de l'évolution inévitable des logiciels, leur structure, qui se compose principalement d'abstractions modulaires telles que les packages s'érode plus ou moins lentement. Lutter contre cette érosion et remodulariser le logiciel est une question qui se décline sous plusieurs formes. Dans cet article nous nous intéressons à l'exploration de la redistribution des classes d'un package vers d'autres packages. Nous utilisons une approche basée sur des techniques d'Analyse Formelle de Concepts pour déterminer les packages receveurs des classes redistribuées.

Abstract

During software evolution, the software structure, which mainly consists of modular abstractions (such as packages) erodes more or less slowly. Modernizing the software structure with remodularization approaches, is an important issue with several variants. In this paper we explore the issue of redistributing classes of a package to other packages. We use an approach based on Formal Concept Analysis to determine the packages that receive the redistributed classes.

Mots clés : Remodularisation logicielle, Analyse de concepts formels
Key words : Software Modularization, Formal Concept Analysis

1 Introduction

Les grands systèmes logiciels basés sur des approches à objets sont composés de classes regroupées dans des packages, formant une structure modulaire. Les relations de dépendances entre classes dans un même package (dépendances internes) et entre classes de différents packages (dépendances externes) engendrent une complexité qui rend difficile la compréhension et la maintenance du système. De plus la structure modulaire tend à se dégrader au fil du temps, rendant nécessaire une intervention experte pour sa modernisation.

Dans cet article, nous étudions une déclinaison particulière du problème présenté par H. Abdeen et al.[2, 1] qui consiste à redistribuer des classes d'un package vers des packages existants. Ce package est peut-être un package de très petite taille et l'on veut rééquilibrer les tailles des packages dans le système, ou bien il a été créé artificiellement pour contenir des classes ajoutées au système au fil de l'eau et le concepteur considère qu'il n'a pas de cohérence sémantique. Dans un cadre de restructuration d'architecture ou d'évolution, ce peut être aussi lié à la disparition d'un nœud dans une architecture distribuée ou à une recentralisation et/ou

un refactoring pour des raisons de maintenance. Nous explorons une solution utilisant l'Analyse Formelle de Concepts (AFC) et illustrons notre proposition par un exemple théorique.

La section 2 présente notre exemple, puis nous décrivons l'approche dans la section 3. Des travaux connexes sont présentés en section 4, puis nous concluons à la section 5.

2 Illustration

Cette section présente la problématique de la remodularisation d'architectures logicielles sur un exemple que nous utiliserons tout au long de l'article. L'architecture présentée figure 1 se compose de cinq packages, A, B, C, D et E. Des dépendances internes ou externes aux packages relient les classes : elles correspondent par exemple à l'appel d'une méthode ou à l'utilisation d'un type. Les dépendances internes de A, B, C et D ne sont pas présentées.

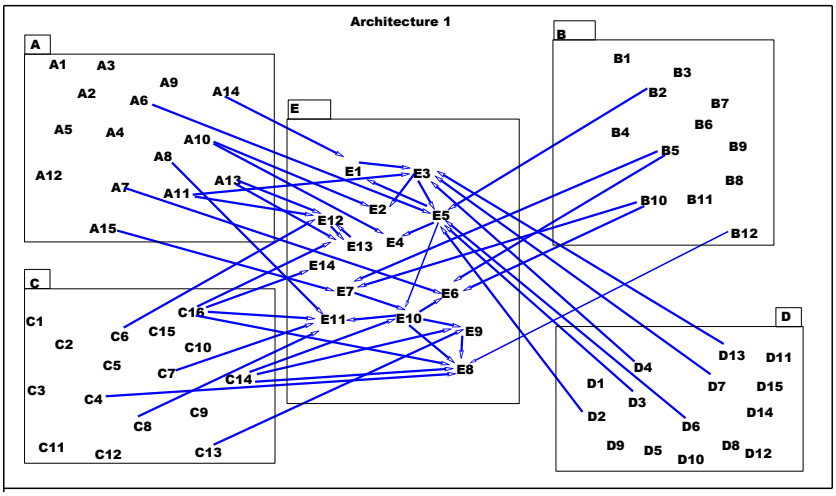

FIGURE 1 – Une architecture initiale composée de classes et de packages

Nous nous intéressons à la redistribution des classes de E vers les autres packages par une méthode exploratoire lors de laquelle des propositions de redistribution sont présentées à un expert. Ces propositions se basent sur l'idée que l'expert, tout en vérifiant la sémantique des classes, pourrait chercher à augmenter la cohésion (au sens du couplage des classes dans un package) et à diminuer le couplage entre classes de packages différents. Pour ce faire, nous pensons qu'il est judicieux de favoriser les deux tendances suivantes :

– les classes d'un package attirent vers elles les classes de E qu'elles utilisent,
– des classes de E interconnectées sont de préférence redistribuées dans le même package.

Nous pensons que l'Analyse Formelle de Concepts (AFC) peut nous apporter des pistes intéressantes pour traiter ce problème car cette technique permet de grouper des classes connectées de manière identique. Nous ne cherchons pas ici à proposer une *meilleure* solution, mais à présenter à un expert différentes solutions hiérarchisées.

3 Approche proposée

L'Analyse Formelle de Concepts (AFC) [6] est une technique d'analyse de données qui permet de grouper des entités possédant des caractéristiques communes. Un concept est un ensemble maximal d'entités (extension du concept) partageant un ensemble maximal de caractéristiques (intension du concept). L'AFC est utilisée en génie logiciel pour traiter des problèmes assez variés [11].

Configurations Dans le cadre de notre problème, nous avons étudié cinq configurations différentes d'analyse avec l'AFC. Nous présentons sur deux d'entre elles l'approche d'exploration. Une configuration d'analyse avec l'AFC consiste à définir un contexte formel C : l'ensemble O des entités étudiées (ou *objets formels*), l'ensemble A des caractéristiques (ou *attributs formels*) et la relation $R \subseteq O \times A$.

Le premier contexte formel associe à une classe c du package E les packages qui accèdent à cette classe c (voir figure 2, partie gauche).

Contexte (Contexte formel C2)**.**
O2 est l'ensemble des classes de E en relation avec l'extérieur
A2 est l'ensemble des packages A, B, C, D (qui ont une relation vers une classe de E)
R2 est la relation "est cible d'un accès externe depuis"
(e,p) ∈ R2 si e est cible d'un accès depuis p, par exemple (E2,A) ∈ R2

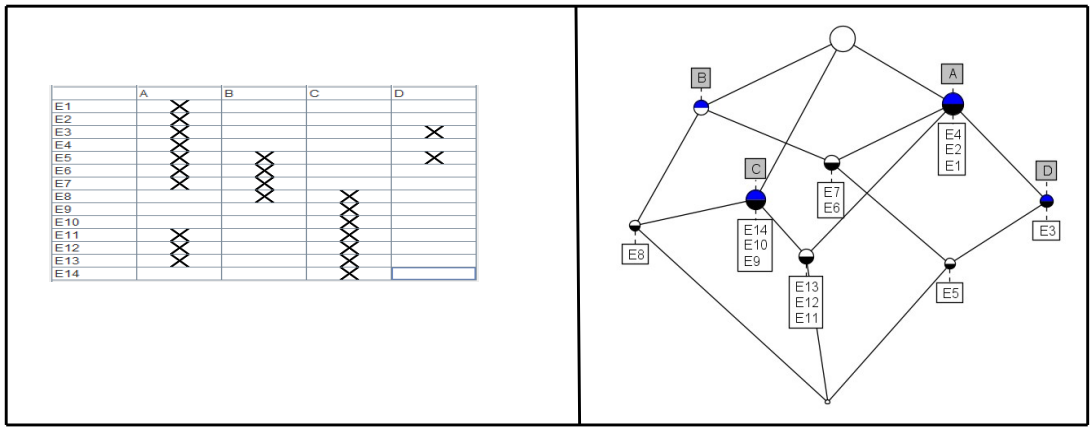

FIGURE 2 – Contexte formel C2 et treillis T(C2) -Architecture 1-.

Le second contexte formel permet d'affiner les résultats et de redistribuer vers le même package deux classes qui seraient interconnectées dans E). Il associe à une classe du package E une autre classe qui lui est connectée (voir figure 3, partie gauche).

Contexte (Contexte formel C5)**.**
O5 est l'ensemble des classes de E en relation avec l'extérieur
A5 = O5 : classes de E en relation avec l'extérieur
R5 est la relation "est connectée à"
(e1,e2) ∈ R5 s'il y a une flèche de e2 vers e1 ou de e1 vers e2, par exemple (E4, E5) et (E5, E4) appartiennent à R2.

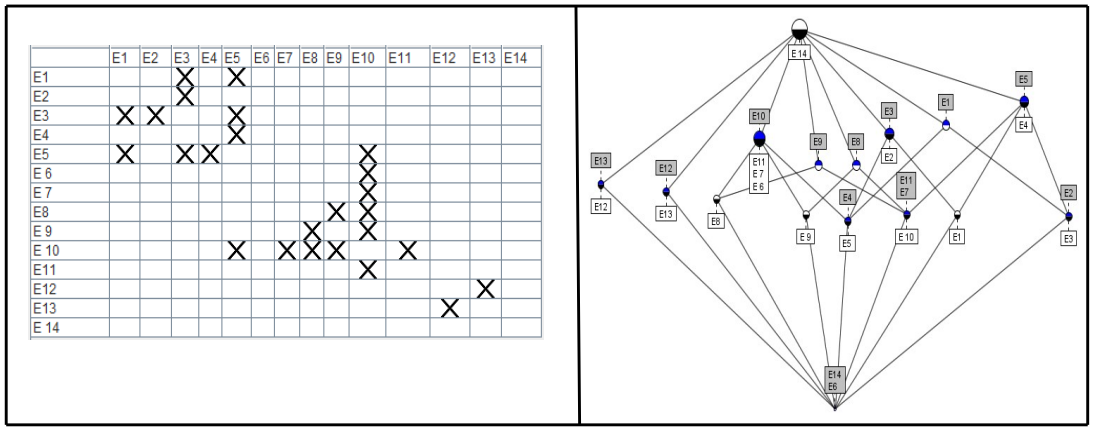

FIGURE 3 – Contexte formel C5 et treillis T(C5) -Architecture 1-.

Les treillis de concepts sont des structures de classification qui exposent les concepts (leurs nœuds) et les relient par spécialisation. Par exemple, le treillis de concepts T(C2) associé au contexte C2 (voir figure 2, droite), contient huit concepts en dehors du top et du bottom. La partie grisée des étiquettes (partie haute) correspond à l'intension simplifiée du concept, tandis que la partie blanche des étiquettes (partie basse) correspond à l'extension simplifiée. Les étiquettes des extensions sont héritées en remontant dans le treillis tandis que les étiquettes des intensions sont héritées en descendant. Par exemple ce treillis T(C2) contient les concepts :

- ({E6,E7,E8}, {B}) tout en haut à gauche, simplifié en ({}, {B})
- ({E11,E12,E13}, {A,C}) au milieu tout en bas, simplifié en ({E11,E12,E13}, {})

Exemple d'exploration L'exploration consiste à naviguer les deux treillis T(C2) et T(C5) pour déterminer les possibilités de redistribution des classes et les présenter à un expert. Nous détaillons partiellement un exemple d'analyse pour expliquer le principe. Les treillis font apparaître des structurations dans les connections. Le treillis T(C5) peut se décomposer en trois gros blocs dans lesquels nous allons choisir des concepts.

1. Analyse du concept ({E1, E3, E4}, {E5}) de la droite de T(C5) : l'extension du concept est dans l'extension du concept (simplifié) ({E1, E2, E4}, {A}) de T(C2), et E5 est aussi connectée à A, l'expert peut choisir de mettre les trois classes E1, E3, E4 dans A.
2. Analyse du concept ({E3}, {E2, E5}) de la droite de T(C5) : les trois classes sont dans l'extension complète du concept d'intension {A} de T(C2), l'expert peut encore choisir de les mettre dans A. Le sous-système {E1, E2, E3, E4, E5} peut être placé dans A.
3. Analyse des concepts de droite ({E12}, {E13}) et ({E13}, {E12}) de T(C5). Dans T(C2), {E12, E13} est dans l'extension du concept d'intension {A, C} qui nous indique les deux solutions possibles. l'expert peut choisir de mettre ensemble E12 et E13 dans A ou dans C, mais il évitera de mettre E12 dans A et E13 dans C. Cela amènera aux deux architectures possibles de la figure 4.
4. Dans le centre très entremêlé de T(C5), l'expert choisit un concept du bas ({E10},{E5, E7, E8, E9, E11}). L'analyse de T(C2) montre que la majorité de ces classes est attirée dans C.

5. L'expert examine le concept ({E8},{E9, E10}) de T(C5). Son intension est dans l'extension du concept d'intension {C} ce qui tend à mettre aussi la classe E8 dans C.

La figure 4 montre deux résultats possibles. Les concepts de T(C5) nous ont renseigné sur la cohésion interne au package E, tandis que la structure de redistribution des classes de E est consultée dans T(C2) et nous informe sur le couplage potentiel.

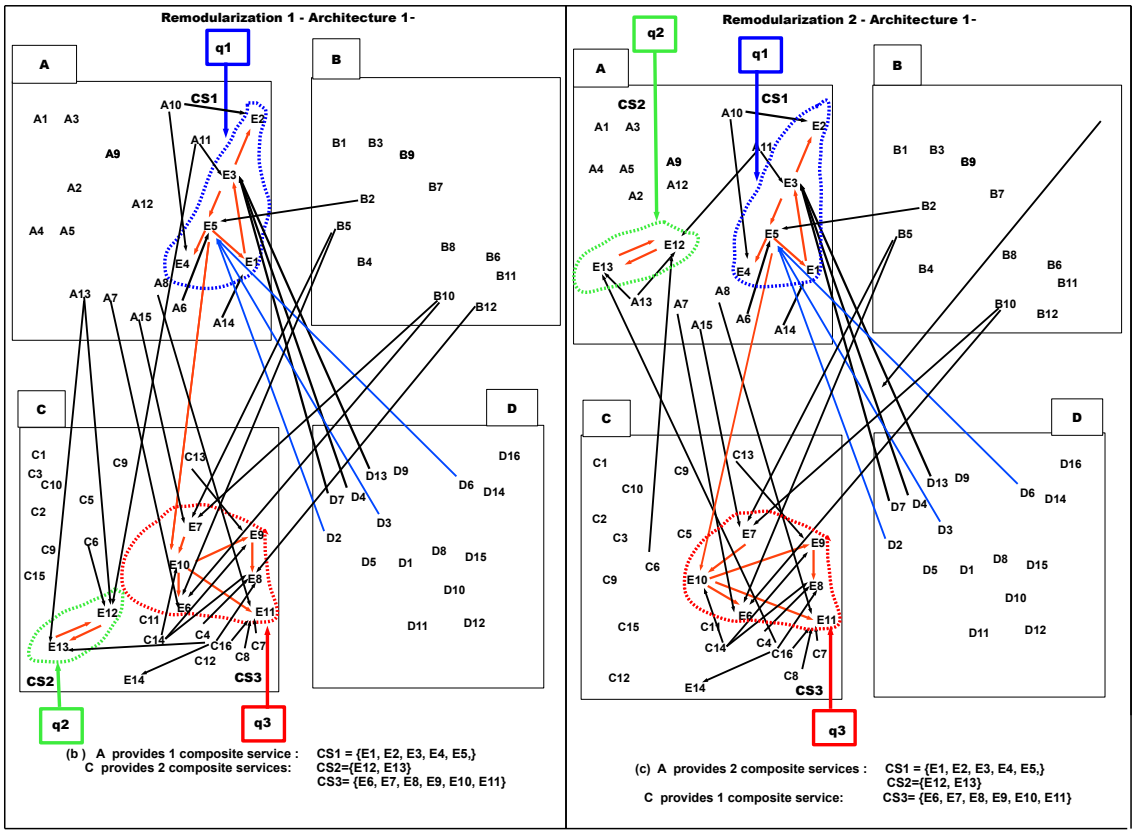

FIGURE 4 – Deux possibilités de remodularisation

4 Travaux connexes

Différentes approches automatisées ont été proposées pour restructurer des systèmes à objets. Nous en citons trois : les algorithmes de regroupement, les algorithmes basés sur des méta-heuristiques et ceux basés sur l'AFC. Les premiers visent à restructurer un système par la répartition de certains de ses éléments (e.g classes, méthodes, attributs) dans des groupes de telle sorte que les éléments d'un groupe sont plus similaires entre eux qu'avec les éléments des autres groupes [3, 7, 5]. Les approches de restructuration basées sur des algorithmes méta-heuristiques [9, 8] sont généralement des algorithmes stochastiques itératifs, qui progressent vers un optimum global d'une fonction en évaluant une certaine fonction objectif (e.g caractéristiques ou métriques de qualité). Enfin les approches basées sur l'AFC [10, 12] fournissent une

méthode algébrique de dérivation de hiérarchies d'abstractions à partir de l'ensemble d'entités d'un système. La référence [4] présente une approche générale pour l'application de l'AFC dans le domaine de la réingénierie du logiciel orientée objet. Dans notre approche, nous ajoutons la dimension d'exploration à l'utilisation de l'AFC.

5 Conclusion et discussion

Dans cet article, nous avons présenté et illustré sur un cas théorique une proposition pour explorer la redistribution des classes d'un package en se basant sur l'AFC. Il reste encore beaucoup de questions liées à cette première réflexion. Les treillis contiennent de nombreuses informations à exploiter : on peut remarquer dans T(C2) que toutes les classes de E connectées à des classes de D sont obligatoirement aussi connectées à des classes de A. Les arcs pourraient être valués pour affiner les forces d'attraction d'une classe sur une autre. Par ailleurs certaines classes du package E peuvent ne pas être connectées à l'extérieur. On pourrait dans ce cas réitérer l'analyse. La méthode pourrait être guidée et évaluée par des métriques et une technique de visualisation comme proposée dans [1].

Références

[1] Hani Abdeen. *Visualizing, Assessing and Re-Modularizing Object-Oriented Architectural Elements.* PhD thesis, Université de Lille, 2009.

[2] Hani Abdeen, Stéphane Ducasse, Houari A. Sahraoui, and Ilham Alloui. Automatic package coupling and cycle minimization. In *International Working Conference on Reverse Engineering (WCRE)*, pages 103–112, Washington, DC, USA, 2009. IEEE Computer Society Press.

[3] Fernando Brito e Abreu, Gonçalo Pereira, and Pedro Sousa. A coupling-guided cluster analysis approach to reengineer the modularity of object-oriented systems. In *Proceedings of the Conference on Software Maintenance and Reengineering*, CSMR '00, pages 13–, Washington, DC, USA, 2000. IEEE Computer Society.

[4] Gabriela Arévalo, Stéphane Ducasse, and Oscar Nierstrasz. Lessons learned in applying formal concept analysis to reverse engineering. In *Proceedings of the Third international conference on Formal Concept Analysis*, ICFCA'05, pages 95–112, Berlin, Heidelberg, 2005. Springer-Verlag.

[5] Markus Bauer and Mircea Trifu. Architecture-aware adaptive clustering of oo systems. In *Proceedings of the Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'04)*, CSMR '04, pages 3–, Washington, DC, USA, 2004. IEEE Computer Society.

[6] B. Ganter and R. Wille. *Formal Concept Analysis - Mathematical Foundations.* Springer, 1999.

[7] Brian S. Mitchell and Spiros Mancoridis. Comparing the decompositions produced by software clustering algorithms using similarity measurements. In *ICSM*, pages 744–753, 2001.

[8] Mark O'Keeffe and Mel í Cinnéide. Search-based refactoring for software maintenance. *J. Syst. Softw.*, 81(4) :502–516, April 2008.

[9] Olaf Seng, Johannes Stammel, and David Burkhart. Search-based determination of refactorings for improving the class structure of object-oriented systems. In Mike Cattolico, editor, *GECCO*, pages 1909–1916. ACM, 2006.

[10] Gregor Snelting. Software reengineering based on concept lattices. In *CSMR*, pages 3–10, 2000.

[11] Thomas Tilley, Richard Cole, Peter Becker, and Peter W. Eklund. A survey of formal concept analysis support for software engineering activities. In *Int. Conf. Formal Concept Analysis (ICFCA 2005)*, pages 250–271, 2005.

[12] Paolo Tonella. Concept analysis for module restructuring. *IEEE Trans. Software Eng.*, 27(4) :351–363, 2001.