h e g

Haute école de gestion
Genève

Etude de la visualisation de code en lien avec les ontologies métier

Travail de Bachelor réalisé en vue de l'obtention du Bachelor HES

par :

Onur ERDOGAN

Conseiller au travail de Bachelor :

Philippe DUGERDIL, professeur HES

Carouge, 07.09.2015

Haute École de Gestion de Genève (HEG-GE)

Filière informatique de gestion

Hes·so GENÈVE
Haute Ecole Spécialisée de Suisse occidentale

Déclaration

Ce travail de Bachelor est réalisé dans le cadre de l'examen final de la Haute école de gestion de Genève, en vue de l'obtention du titre Bachelor of Science.

L'étudiant atteste que son travail a été vérifié par un logiciel de détection de plagiat.

L'étudiant accepte, le cas échéant, la clause de confidentialité. L'utilisation des conclusions et recommandations formulées dans le travail de Bachelor, sans préjuger de leur valeur, n'engage ni la responsabilité de l'auteur, ni celle du conseiller au travail de Bachelor, du juré et de la HEG.

« J'atteste avoir réalisé seul le présent travail, sans avoir utilisé des sources autres que celles citées dans la bibliographie. »

Fait à Genève, le 07.09.2015

Onur Erdogan

Remerciements

Je souhaiterai avant tout remercier la Haute Ecole de Gestion ainsi que les professeurs m'ayant soutenue durant ces deux années de formations.

Je tiens, tout particulièrement, à remercier Monsieur Philippe Dugerdil qui m'a suivi et conseillé tout au long de ce travail.

Je tiens également à remercier Charlène Léa Youyou m'ayant soutenu pendant la réalisation de mon mémoire et qui a relu avec attention mon travail.

Enfin, j'aimerais remercier mes proches, ma famille et mes amis, qui ont su me soutenir durant la réalisation de mon mémoire.

Résumé

La maintenance logicielle est une tâche importante du software engineering. Celle-ci représente entre 60% à 90% du coût total du logiciel. Une des tâches les plus difficiles de la maintenance est la compréhension de l'application. Des récentes études démontrent que les développeurs passent entre 60 et 80% de leur temps à comprendre le fonctionnement du logiciel avant de faire une quelconque opération de maintenance. Cette phase est nécessaire, car pour réaliser une opération de maintenance, il faut comprendre le code source et le fonctionnement de celui-ci.

Il existe actuellement plusieurs manières d'analyses de code, dynamiques et statiques liées au reverse engineering qui permettent de fournir des visualisations du logiciel. En revanche, celles-ci ne permettent pas de faire la liaison entre l'ontologie métier et le code source du logiciel, par conséquent, la tâche de localiser le code source associé à l'ontologie métier revient au développeur. En effet, la liaison entre l'ontologie métier et le code source permet de comprendre le logiciel en localisant les différents concepts métiers implémentés dans le code source.

Le but de mon travail consiste à explorer les techniques existantes de visualisations de code associées aux ontologies métiers, afin d'identifier un modèle prometteur et à implémenter un prototype.

Table des matières

Liste des tableaux

Liste des figures

1. Introduction

La maintenance logicielle est une tâche importante du software engineering. Celle-ci représente entre 60% à 90% du coût total du logiciel. Une des tâches les plus difficiles de la maintenance est la compréhension de l'application. Des récentes études démontrent que les développeurs passent 40% de leur temps à comprendre le fonctionnement du logiciel avant de faire une quelconque opération de maintenance

Actuellement, aucunes recherches existantes sur la visualisation de code ne proposent une représentation graphique se basant sur l'ontologie métier aidant ou améliorant la compréhension. La plus part des travaux de recherches ne définissent pas ce que représente la compréhension, en partant de là il est difficile de présenter une approche.

Qu'est-ce que la compréhension ?

Nous avons étudié la définition suivante « *La compréhension, c'est faire le lien entre ce que l'on sait déjà et ce que l'on observe* ».

Il est important de définir la notion de compréhension pour établir une approche cohérente. En effet, lors d'une maintenance logicielle, il est nécessaire d'avoir une compréhension du système, pour obtenir cette connaissance, il faut établir le lien entre les éléments du domaine métier (besoins métiers) et ceux du domaine système (code source).

L'ontologie métier se situe au niveau des besoins métiers, donc l'objectif est de faire le lien entre le code source et l'ontologie métier. L'ontologie métier peut être représentée sous forme d'arbre avec différents niveaux d'héritages et de composition. A l'aide de ces propriétés nous allons pouvoir effectuer une simplification de la représentation des concepts métiers afin d'améliorer la compréhension.

Pour finir, la réalisation de ce travail se basera sur une technique d'analyse dynamique du code utilisant une trace d'exécution, le but étant d'identifier l'intervention des différentes ontologies et la création d'une représentation graphique aidant à la compréhension.

2. Définition du problème

2.1 Objectif

Nous savons qu'un logiciel évolue au cours du temps pour s'adapter au mieux aux besoins de de l'environnement réel sous peine de devenir progressivement mois utile dans cet environnement, l'évolution de celui-ci implique une augmentation de sa complexité.

Il existe deux lois de Lehman [1]dans le cadre de l'évolution logiciel :

- Loi du changement continue « *Un programme utilisé dans un environnement réel doit être modifié ou devenir progressivement mois utile dans cet environnement* ».
- Loi de la complexité croissante « *A mesure qu'un programme change et évolue, sa structure à devenir plus complexe. Des ressources supplémentaires doivent alors être engagées pour préserver et simplifier cette structure* ».

Par la suite, la maintenance logicielle est une tâche importante du software engineering, elle permet l'évolution logicielle. Pour ce faire, elle implique la compréhension du logiciel avant une quelconque opération de maintenance.

L'objectif de ce travail est d'explorer une nouvelle approche pouvant aider la compréhension en se basant sur les ontologies métier pour la création d'une représentation graphique avec différents niveaux d'abstractions.

Une ontologie métier est une représentation d'un ensemble de concept métier ayant un sens permettant de décrire les concepts du domaine.

[1] LEHMAN, M. M., BELADY, L. A. – A model of large program development. IBM Syst. Journal 15(3) Septembre 1976.

2.2 Compréhension

Il s'agit d'un raisonnement psychologique qui nécessite une réflexion et l'utilisation de connaissances acquises pour analyser l'élément de manière adéquat. La compréhension[2] permet de faire le lien entre ce que l'on sait déjà et ce que l'on observe.

Dans le cas du code, ce que l'on sait déjà, ce sont les justifications métiers qui ont été développées. Ce que l'on observe c'est la structure du code. Par conséquent, comprendre le code revient à faire le lien les spécifications métiers et le code qui est exécuté selon un certain niveau de granularité.

De plus, grâce à la visualisation de code en lien avec les ontologies métier nous allons pouvoir identifier les concepts qui sont impliqués dans chaque segment[3]. En effet, nous verrons tout au long de la trace[4] l'enchaînement de ceux-ci qui seront représentés sous forme hiérarchique. Ainsi, on pourra faire le lien entre le code et ce qu'il représente dans le cadre métier.

Par ailleurs, il serait intéressant d'intégrer par la suite une fonctionnalité de navigation dans le code pour identifier directement le code qui est implémentée l'ontologie.

---

[2] BARON, Jonathan. Thinking and Deciding. Cambridge : Cambridge. ISBN 9780511464874. MASON, Richard. Understanding Understanding. New-York : Suny. ISBN 0791458717.

[3] Segment : représentation d'une partie de la trace obtenue à l'aide d'une technique de segmentation.

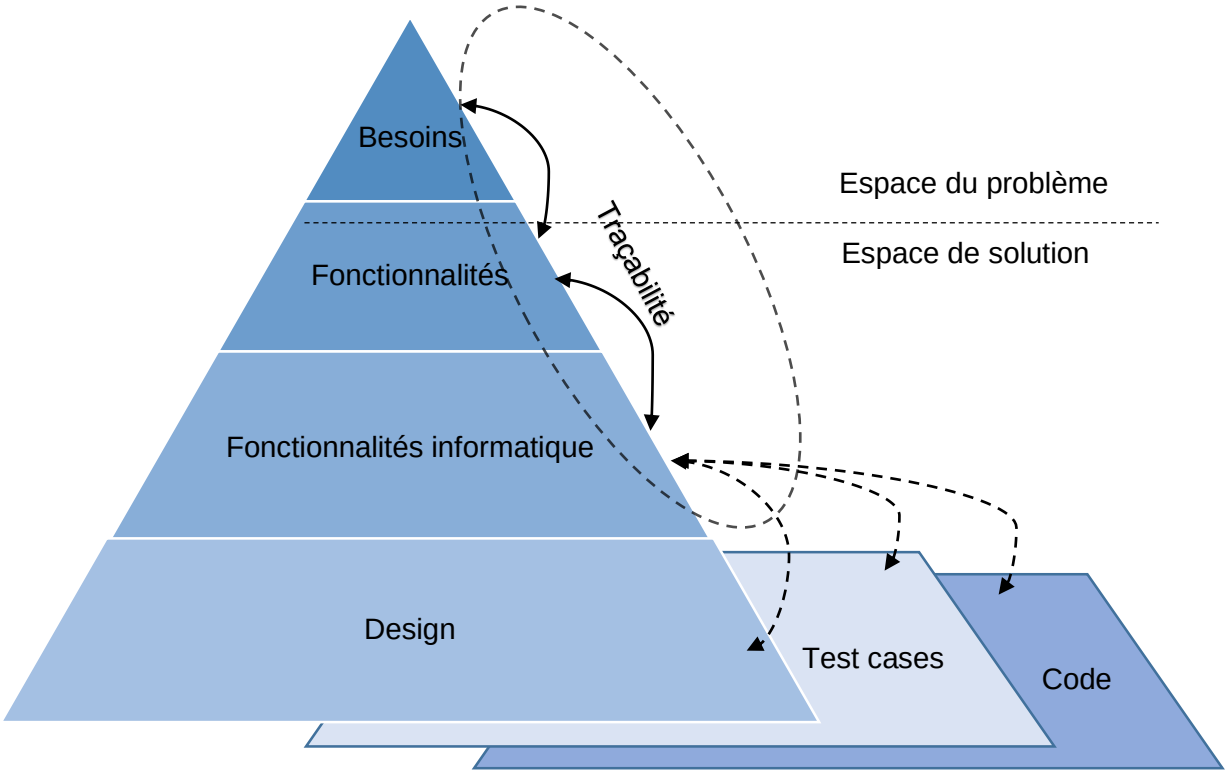
[4] Trace : représentation d'un enregistrement d'évènements liés à la réalisation d'un use case avec ses appels de méthodes lors de son exécution dans un logiciel.

2.3 Compréhension logicielle

La compréhension dans le cadre d'un logiciel intervient lors des opérations de maintenance. Pour réaliser une opération de maintenance, il est important de pouvoir assurer la traçabilité entre l'implémentation des fonctionnalités au niveau du code source et les besoins métiers auxquelles elles répondent. La traçabilité d'un logiciel permet de faire la liaison entre les besoins et les fonctionnalités informatique implémentés, comme nous pouvons l'observer sur la Figure 1.

La traçabilité est donc la compréhension du logiciel.

Figure 1 : Pyramide de traçabilité d'une application



Dean Leffingwell – Features, Uses Cases, Requirements, Oh My ! Rational Software 2001

Vu dans le cadre du cours du module 625-1 Génie logiciel - de Monsieur Philipe Dugerdil

3. Etat de l’art

3.1 Applications existantes pour la visualisation de logiciel

Il existe plusieurs logiciels permettant d’effectuer de la visualisation d’application, mais ces derniers sont soit pour la plus part peu clair et difficilement utilisables, soit ils ne sont plus d’actualité. Pour certains Eclipse inclut déjà la fonctionnalité.

Ces logiciels permettent de faire principalement des exploitations visuelles du code sans utiliser les techniques de visualisations métier. Ils nous aident en rien à faire la relation entre les éléments du code et du métier.

Aucun de ces logiciels ne permet d’observer une représentation de l’ontologie et de ses différents concepts. En effet, ces logiciels exploitent principalement les fréquences d’utilisations d’une classe, d’une méthode ou une représentation visuelle des différentes classes entres-elles, mais aucunement l’ontologie. Par conséquent, cela revient à exploiter le code sous différentes formes visuelles, sans savoir ce qu’il représente d’un point de vue métier ni de faire le lien avec la partie métier, tandis que notre objectif est de réussir à faire le lien, pour comprendre et ensuite apporter des modifications, si nécessaire, au bon endroit.

Enfin, aucune des applications existantes observées dans le cadre de ce travail ne permet d’effectuer le lien entre le domaine métier et le code. Il s’agit d’un problème récurrent à toutes ces applications.

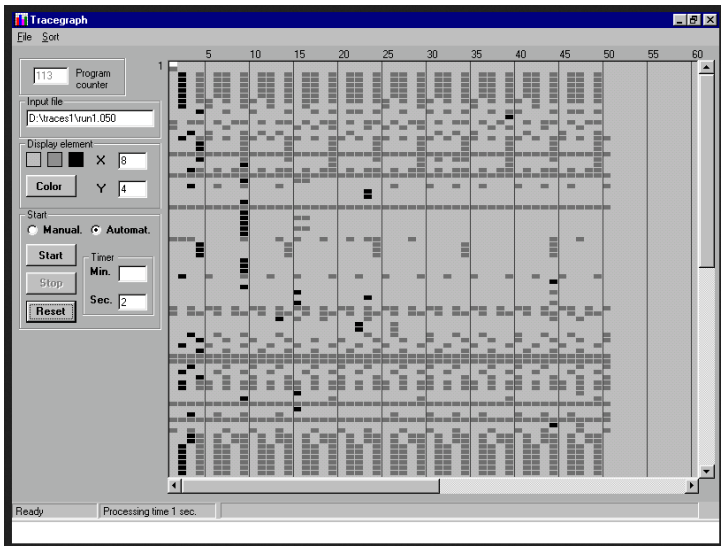
3.1.1 TraceGraph

TraceGraph [5]se base sur une trace pour fournit un visuel du fonctionnement du programme pour distinguer facilement les changements lors de l'exécution.

Cet outil est peu clair et difficilement utilisable.

Ce logiciel ne fournit aucun résultat représentant les ontologies et les concepts impliqués. Nous pouvons observer des blocs sans obtenir d'informations aidant à la compréhension. En aucun cas nous pouvons observer le problème métier auquel répond le programme analysé, il n'y a aucune représentation du lien entre le domaine métier et le code.

Figure 2 : Exemple de représentation du logiciel TraceGraph



[5] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

3.1.2 Relo

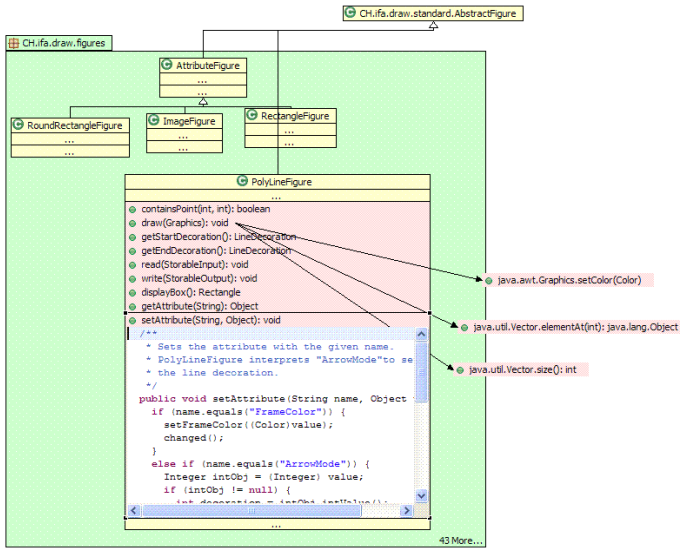
Relo[6] est un Plug-in Eclipse (plus entretenu depuis 2007) qui permet d'effectuer du reverse modeling.

On retrouve les différents niveaux de hiérarchie d'héritage, les méthodes ainsi que leurs codes.

Ce Plug-in Eclipse représente seulement les classes et leurs méthodes sans tenir compte de la trace d'exécution, il fournit une représentation statique de l'application. On ne peut pas observer l'enchainement de des différentes méthodes et les concepts qu'elles représentent dans le code. Le problème est identique aux autres applications. En effet, nous pouvons voir une hiérarchie des différentes classes, mais cette dernière ne permet pas d'effectuer un lien entre le domaine métier et le code. C'est un outil principalement développé pour comprendre la structure du code, mais il ne permet pas de comprendre à quoi répond le code dans le domaine métier.

RSA[7] permet d'obtenir le même résultat que Relo.

Figure 3 : Exemple de représentation avec le logiciel Relo



[6] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

[7] RSA : Rational Software Architect est une application de modélisation fournit par IBM.

3.1.3 FEAT

FEAT[8] est un outil développé sous la forme d'un Plug-in Eclipse qui permet de localiser, décrire et analyser les éléments du code source, cet outil est aussi de base dans Eclipse.

Il permet au développeur de créer des vues d'éléments structurellement liés en les ajoutant à des graphiques, pouvant être associés avec du code source par un mécanisme de requête.

Il ajoute une perspective à Eclipse qui permet d'avoir un rendu visuel.

Mais ça reste un outil manuel, avec lequel il faut selectionner soi-même les méthodes, attributs ou code que l'on veut analyser.

Cet outil permet de faire une représentation statique des méthodes ainsi que des classes dans lesquelles elles sont déclarées et référencées. On ne voit aucunement les ontologies impliquées dans la trace d'exécution, donc le lien entre le domaine métier et le code est inexistant.

3.1.4 SHriMP

SHriMP[9] est une suite applicative qui contient les plug-ins Creole et Jambalaya.

Creole permet de parcourir le code source et la documentation via des liens hypertexte intégrant code et navigation structurelle. Ils permettent également d'effectuer une recherche générale, une recherche d'artefacts et une recherche de relations.

Jambalaya permet de visualiser les ontologies, pas de mise à jour depuis 2006, impossible de faire fonctionner la version disponible.

Pour que la suite SHriMP fonctionne, on part du principe qu'une documentation existe, ce qui n'est pas toujours le cas. De plus, il faut que la documentation se situe au niveau métier dans le cas contraire, on ne pourra pas faire le lien entre les besoins métiers et le code. Cette documentation est souvent faite par les informaticiens, donc elle se situe au mauvais niveau, elle est principalement technique et sert à décrire le code.

---

[8] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

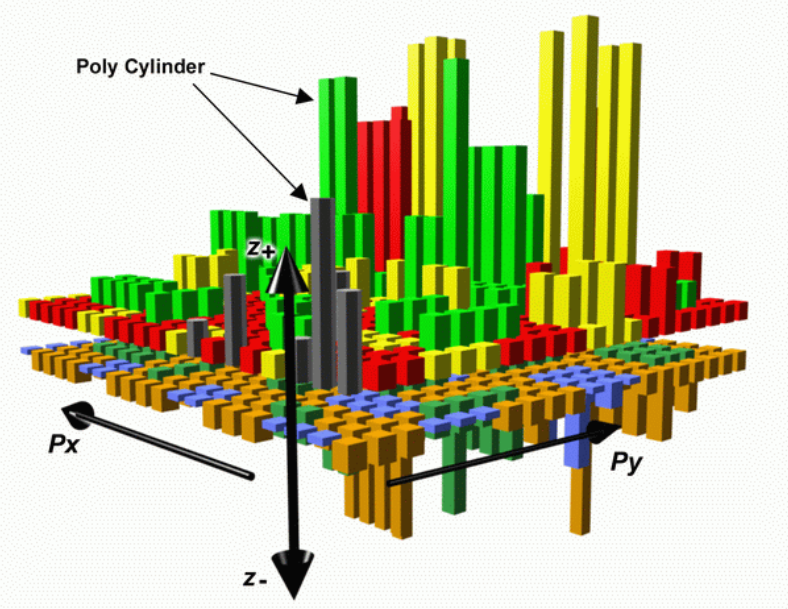
[9] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

3.1.5 Source 3D Viewer

Source 3D Viewer[10] (sv3D) est un framework de visualisation qui utilise une représentation en 3D. Il utilise des cylindres afin de représenter une entité logicielle (par exemple : caractères, lignes de code, méthodes, fonctions, classes, etc.) et il utilise des containers pour représenter les agrégats de ces éléments. La hauteur et la couleur des cylindres sont utilisées pour représenter les différentes mesures du logiciel ou des autres données.

sv3D a été conçu pour faciliter les interactions avec différents outils d'analyse de code source.

Figure 4 : Exemple de représentation avec le logiciel Source 3D Viewer



Cet outil permet d'avoir une représentation visuelle très colorée avec différentes formes et taille de cylindre. En revanche, on peut difficilement comprendre le résultat de la représentation fournit par le programme. En effet, cet outil ne permet de faire uniquement de l'analyse statique de code.

[10] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

3.1.6 IRISS

La localisation de concept est une des activités les plus courantes lorsqu'il faut changer un élément existant du logiciel. La méthode utilisée par IRISS [11]est « Information retrieval (IR) » qui est un add-on pour Microsoft Visual Studio .NET.

Le processus de localisation de concept d'IRISS est conçu de la manière suivante :

- Prétraitement du code source et de la documentation.
- Construction d'un vecteur représentant le système logiciel en utilisant le Latent Semantic Indexing (LSI).
- Exécuter des requêtes définies par l'utilisateur.
- Récupérer et analyser les résultats, qui sont retournés comme un classement des éléments du code source en fonction de la granularité.

La technique du LSI est basée sur les commentaires des programmeurs lors de la conception du logiciel. Elle nécessite donc obligatoirement des commentaires et une certaine structure de ces derniers, car elle se base sur le langage naturel. En effet, sans commentaires venant du niveau du métier, cette technique sera inutilisable pour la représentation des différents concepts impliqués. Pour représenter les concepts métier, il serait nécessaire que les commentaires soient orientés sur le domaine métier.

[11] XIE, X., POSHYVANYK D., MARCUS, A. – 3D Visualization for Concept Location in Source Code. ICSE '06 Proceedings of the 28th international conference on Software engineering.

4. Techniques d’analyse

4.1 Analyse statique

L’analyse statique d’un logiciel regroupe différentes méthodes dans le but d’obtenir des informations sur le fonctionnement d’un logiciel sans l’exécuter, pour corriger une erreur de programmation ou de conception. La méthode principalement utilisée est l’analyse du code source du logiciel.

4.2 Analyse dynamique

L’analyse dynamique d’un logiciel a pour but d’étudier le fonctionnement d’un logiciel à son exécution. À l’aide de cette technique, nous allons pouvoir obtenir des informations comme les méthodes utilisées et leurs fréquences d’appels sur le temps et l’instanciation des classes. Elle repose sur l’utilisation d’une trace d’exécution.

4.2.1 Trace d’exécution

Une trace d’exécution représente l’enregistrement des événements liés à la réalisation d’un use case avec ses appels de méthodes lors de son exécution dans un logiciel.

Dans le cas de la compréhension de logiciel, il est important de faire l’enregistrement de l’utilisation la plus fréquente d’un use case et non des alternatives possibles, ni de l’entièreté du logiciel. En effet, il serait difficile d’étudier précisément le fonctionnement du logiciel dû à l’abondance d’informations, pour un grand logiciel d’entreprise on estime le nombre d’événements générés à plusieurs millions.

Les méthodes représentent des concepts métiers.

De plus, la trace d’exécution (analyse dynamique) contenant les appels de méthodes nous permettra d’observer la proximité de plusieurs concepts. Cette observation serait est rendu possible car la trace est une représentation dynamique des différents appels de méthodes. Ainsi, nous obtiendrons une topologie des concepts et les liens entre ces derniers.

Contrairement à une analyse statique dans laquelle les concepts peuvent être éloignés dans leurs implémentations dans la structure du code, par conséquent il serait impossible d’observer leurs proximités.

4.2.1.1 Contenu d’une trace

La trace d’exécution va contenir toutes les méthodes ainsi que leurs classes qui ont été utilisées lors de la réalisation du use case sur le logiciel.

4.2.1.2 Segmentation d'une trace

Comme énoncé au chapitre 4.2.1 une trace d'exécution est susceptible de contenir un nombre importants d'enregistrement, il est possible pour faciliter et simplifier l'analyse de la trace d'effectuer une segmentation[12] en segments contigus. En effet, nous pourrions analyser distinctement chaque segment pour obtenir des informations ciblées ainsi que le nombre d'occurrence de chaque méthode dans le segment.

Figure 5 : Exemple de trace

	Trace											
Appel n°	1	2	3	4	5	6	7	8	9	10	11	12
Méthodes	M1	M3	M0	M5	M2	M4	M1	M3	M0	M6	M7	M5

La figure 5 présente l'exemple d'une trace qui contient douze appels de sept méthodes différentes (M1, M2, M3, M4, M5, M6, M7).

Figure 6 : Exemple de segmentation d'une trace

	Trace											
Segments	**1**			**2**			**3**			**4**		
Appel n°	1	2	3	4	5	6	7	8	9	10	11	12
Méthodes	M1	M3	M0	M5	M2	M4	M1	M3	M0	M6	M7	M5

En appliquant la technique de segmentation, nous pourrions diviser la trace en 4 segments contigus de taille égale à raison de 3 méthodes par segment.

[12] DUGERDIL PH. - Using trace sampling techniques to identify dynamic clusters of classes. IBM CAS SOFTWARE AND SYSTEMS ENGINEERING SYMPOSIUM (2007,Dublin). : proceedings of the 2007 conference of the center for advanced studies on Collaborative research. Hamilton auditorium of Dublin, 24 octobre 2007. 9 p.

Pour finir, dans le cadre de ce travail, nous allons utiliser une technique développée par la HEG[13] pour faire le lien entre les méthodes et les concepts métiers. Cette technique permet de retrouver les concepts métier référencés dans les méthodes. Il est ainsi possible de remplacer dans une trace d'exécution les méthodes par les concepts référencés. Admettons que nous avons la relation méthode-concept suivante : M0->C1, M1 -> C1, M2 -> C3, M3 -> C3, M4 -> C4, M5 -> C4, M6 -> C4, M7 -> C5. La « trace de concepts » deviendrait alors :

	Trace											
Segments	**1**			**2**			**3**			**4**		
Appel n°	1	2	3	4	5	6	7	8	9	10	11	12
Concepts	C1	C3	C1	C4	C3	C4	C1	C3	C1	C4	C5	C4

En calculant la fréquence des concepts dans chaque segment de la trace, nous obtenons une nouvelle représentation de l'information :

	Trace							
Segments	**1**		**2**		**3**		**4**	
Concepts	C1	C3	C4	C3	C1	C3	C4	C5
Fréquence	2	1	2	1	2	1	2	1

[13] Haute Ecole de Gestion de Genève

5. Identification des liens entre le code et les ontologies au moyen d'une visualisation

L'idée est d'exploiter une représentation hiérarchique des différents concepts impliqués dans un segment de la trace avec des niveaux d'abstraction et différents mécanismes d'abstraction.

La segmentation permet de fragmenter la quantité d'information, mais celle-ci reste tout de même très importante. Cependant, grâce aux mécanismes d'abstraction nous allons pouvoir réduire la quantité d'information toute en veillant à ne pas perdre d'informations importantes.

Nous allons proposer un outil d'aide à la compréhension selon les définitions des points 2.2 et 2.3. Le but, rappelons-le, est de simplifier l'information. Pour cela, il faudra soustraire l'information déductible à l'aide de deux mécanismes d'abstraction. Nous présenterons ces mécanismes par la suite.

Les mécanismes d'abstraction vont permettre de limiter la quantité d'information, ce qui nous permettra de faciliter la compréhension du code. En effet, sans cette simplification, il est très difficile de faire le lien entre le code et le domaine métier dû à la quantité importante d'informations disponibles. Ces mécanismes vont rendre les liens explicites entre les différents concepts.

Voici les différents objectifs et idées intéressantes que je vais explorer durant la réalisation de mon travail :

- Etude d'un modèle simplifiant la représentation d'une trace
    - Deux mécanismes d'abstractions
        - Spécialisation – Généralisation
        - Composition
    - Recherche de pattern de concepts abstraits
- Extraire les concepts impliqués dans chaque segment de la trace
- Compression de la trace
- Choisir le niveau d'abstraction

Pour commencer, nous allons segmenter la trace et en extraire les méthodes appelées. Par la suite, il faudra faire le lien entre la méthode et le concept qu'elle représente. Pour finir, nous pourrons créer des abstractions à l'aide des deux mécanismes afin de simplifier le contenu de la trace.

5.1 Etude d'un graphique de concepts

Actuellement aucun outil que nous avons observé ne propose une visualisation à l'aide de graphique compréhensible ou apportant un réel plus à la compréhension.

Pour la conception d'un graphique de concepts deux mécanismes d'abstraction seront mis en place :

- Spécialisation - Généralisation
- Composition

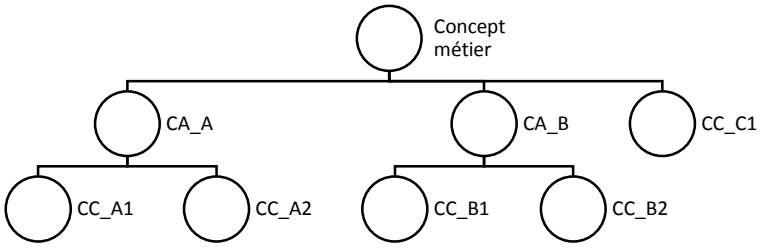
5.1.1 Mécanisme de spécialisation & Généralisation

On peut remarquer qu'il existe des spécialisations de concepts. La relation de spécialisation est le support d'un mécanisme d'abstraction : si on observe un concept spécialisé alors on observe naturellement le concept abstrait auquel il appartient. De plus cette relation est transitive.[14]

5.1.1.1 Exemple théorique

Dans cet exemple théorique, nous avons les Concepts Concrets A1, A2, B1, B2 et C1 ainsi que les Concepts Abstraits A et B qui possèdent comme fils les différents Concepts Concrets.

Figure 7 : Exemple théorique de graphique représentant les concepts et leurs abstractions



CA : Concept Abstrait

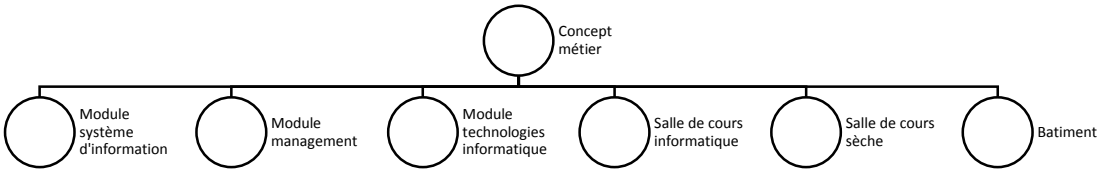
CC : Concept Concret

[14] Définition de Wikipédia « *En mathématiques, une relation transitive est une relation binaire pour laquelle une suite d'objets reliés consécutivement aboutit à une relation entre le premier et le dernier.* »

5.1.1.2 Exemple concret

Je vais utiliser pour cet exemple, le domaine de l'étude. Dans lequel nous avons comme différents concepts comme les « modules » généraux ainsi que les différents types de « salles » et les « bâtiments »

Figure 8 : Exemple théorique de représentation graphique sans généralisation et spécialisation

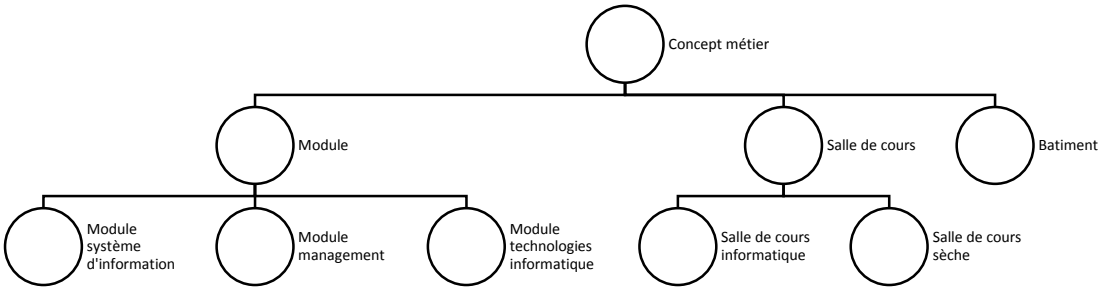


Nous retrouvons dans cet exemple les concepts « Module système d'information », « Module management » et « Module technologies informatique » qui sont tous une spécialisation de « Module ».

De même pour « Salle de cours informatique » et « Salle de cours sèche » qui sont les deux une spécialisation de « Salle de cours ».

Voici ci-dessous une représentation avec l'introduction de concepts abstraits lié par une relation de généralisation et spécialisation :

Figure 9 : Exemple concret de représentation graphique avec généralisation et spécialisation



Dans cet exemple nous retrouvons « Module » et « Salle de cours » comme généralisation (abstraction) des différents concepts métiers.

5.1.2 Mécanisme de composition

Nous avons aussi la notion de relation entre concepts qu'il faudra gérer, car il se peut qu'un concept soit une « partie-de » d'un autre concept. En effet, lorsqu'un concept dépend en partie d'un autre, c'est-à-dire une composition de concepts, cette dépendance peut nous permettre de créer une couche d'abstraction supplémentaire.

Dans le cadre de ce mécanisme, nous allons étudier la composition[15] de concepts.

Il est important de savoir qu'un composant est intimement dépendant de la composition. C'est-à-dire que le composant ne peut pas avoir d'existence propre sans la composition.

Contrairement à une agrégation[15] dans laquelle le composant peut avoir une existence propre.

Nous pouvons donc affirmer que la composition, par définition, est caractérisée par le fait que le cycle de vie du composant est lié à celui du composé ; caractéristique que l'agrégation ne possède pas. Ainsi, un composant ne peut exister sans le composé auquel il appartient. Par conséquent, ceci nous suggère un mécanisme d'abstraction.

Si nous savons qu'un concept est intimement dépendant d'un autre concept, alors nous pouvons créer une abstraction.

La composition est une relation « partie de ». C'est-à-dire que si un concept B fait partie d'un concept A, alors B ne peux pas exister sans A. Ainsi, si A disparaît alors B également.

Dans notre exemple, nous avons donc un concept qui est composé d'autres concepts, c'est-à-dire une composition de concepts.

[15] Terme venant de l'Unified Modeling Language (UML)

5.1.2.1 Agrégation et composition

L'utilisation du mécanisme de composition pour effectuer une simplification de l'information dépend fortement du type de relation, comme nous le verrons ci-après.

5.1.2.1.1 Agrégation

L'agrégation est une association non symétrique, qui représente une relation de type composé et composant.

Contrairement à une composition, ici, un composant peut avoir une existence indépendante du composé. Ce qui diminue la force du couplage.

Figure 10 : Exemple d'agrégation

Dans cet exemple, nous avons la voiture et les roues de la voiture. La voiture est composée de roue. Mais la roue est un élément qui peut avoir une existence propre sans la voiture.

5.1.2.1.2 Composition

La composition est une association qui exprime un couplage fort et une relation « intime ». Nous retrouvons comme l'agrégation une relation de type composé et composant, mais dans le cas d'une composition, le composant ne peut pas avoir une existence propre sans le composé. En effet, les cycles de vies des composants et du composé sont liés. Si le composé est détruit, ses composants le sont aussi.

Figure 11 : Exemple de composition

Dans cet exemple, nous avons le bâtiment et les salles de cours. Nous savons que les salles de cours sont physiquement dépendantes du bâtiment. Donc, si nous supprimons le bâtiment, nous supprimerons aussi les salles de cours, car elles ne peuvent pas avoir d'existence sans le bâtiment. Nous pouvons donc affirmer que les salles de cours font « partie de » bâtiment.

5.1.2.1.3 Abstraction et composition

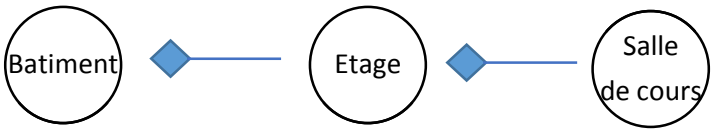
Nous suggérons de pouvoir remplacer l'occurrence d'un composant par le composé qui le contient. Ce mécanisme ne possède pas la même validité que pour la spécialisation. Toutefois, il devrait nous permettre de simplifier l'information de la trace.

De plus, nous pouvons appliquer ce mécanisme d'abstraction par transitivité comme pour le mécanisme de généralisation - spécialisation.

Figure 12 : Exemple d'abstraction non valide

Dans cet exemple, la transitivité n'est pas applicable car les relations ne sont pas de même nature. En effet l'observation de matériel n'implique pas d'observer obligatoirement une salle de cours.

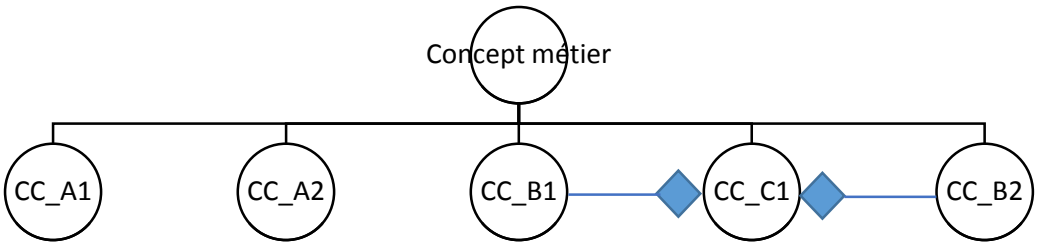
Figure 13 : Exemple d'abstraction valide



Ici, la transitivité est applicable car la relation est de même type. La salle de cours nous suggère la présence de l'étage, qui lui-même nous suggère la présence du bâtiment. Donc on peut appliquer une simplification de l'information par abstraction : si on observe une salle de cours dans la trace de concepts, on pourra remplacer ce concept par le bâtiment qui est forcément présent (sinon la salle de cours ne le serait pas).

5.1.2.2 Exemple théorique

Figure 14 : Exemple théorique de simplification de la trace par le mécanisme de composition

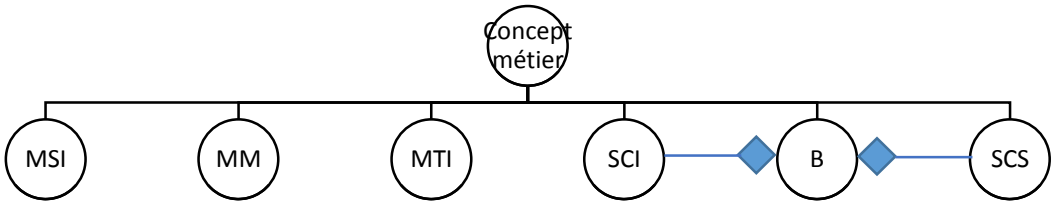


	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	CC_B1	CC_A2	CC_B2	CC_A1	CC_A2	CC_C1	CC_A1
Mécanisme d'abstraction	↓		↓			↓	
	CC_C1	CC_A2	CC_C2	CC_A1	CC_A2	CC_C1	CC_A1

On peut remplacer CC_B1 et CC_B2 par CC_C1. Cette simplification est possible en exploitant le mécanisme d'abstraction par la composition dans lequel nous avons énoncé le fait de remplacer l'occurrence du composant (CC_B1 et CC_B2) par le composé (CC_C1).

5.1.2.3 Exemple Concret

Figure 15 : Exemple concret de graphique représentant la dépendance entre concepts



Voici ci-dessous les abréviations des noms de concepts utilisés pour l'exemple.

MSI : Module système d'information

MM : Module management

MTI : Module technologies informatiques

SCI : Salle de cours informatique

SCS : Salle de cours sèche

B : Bâtiment

Figure 16 : Exemple concret de détection de composition de concept dans une trace

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	SCI	MSI	SCS	MM	SCI	MTI	SCS
Mécanisme d'abstraction	⇓		⇓		⇓		⇓
	B	MSI	B	MM	B	MTI	B

On peut déduire la présence B dans l'observation de SCI et SCS.

5.1.3 Combinaison des deux mécanismes d'abstraction

En combinant les deux mécanismes d'abstractions, nous pouvons obtenir une simplification de la représentation de la trace de concepts étant donné la nécessité de simplifier l'information pour faciliter la compréhension.

5.1.3.1 Ordre des mécanismes

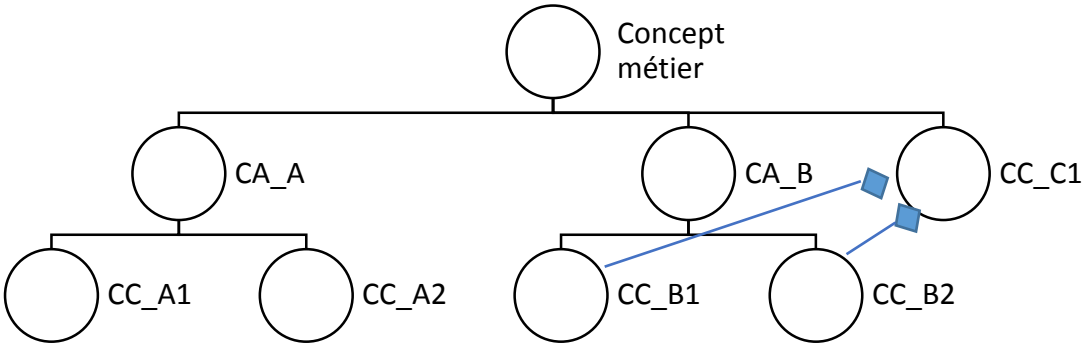
L'ordre d'application des mécanismes d'abstraction est important. En effet car le mécanisme d'abstraction par généralisation – spécialisation sera toujours pertinent. Cependant le mécanisme d'abstraction par composition n'est pas toujours approprié car on a deux objets différents, ce qui n'est pas le cas pour le mécanisme de généralisation – spécialisation.

Par conséquent, tant que le mécanisme d'abstraction par généralisation – spécialisation est applicable, il sera mis en œuvre, puis ensuite le mécanisme d'abstraction par composition.

5.1.3.2 Exemple théorique

Si nous reprenons la spécialisation – généralisation de la Figure 7 et la composition de la Figure 14 nous pouvons construire la représentation suivante :

Figure 17 : Exemple théorique de représentation hiérarchique de l'ontologie



A l'aide de cette représentation nous pouvons observer l'ontologie représenter sous forme hiérarchique.

Sur la figure 20, est représentée la simplification de la trace. Cette simplification est possible car CC_B1 et CC_B2 sont des spécialisations de CA_B donc on peut remplacer les occurrences CC_B1 et CC_B2 par CA_B dans un premier temps en appliquant le mécanisme d'abstraction par spécialisation – généralisation.

Figure 18 : Exemple théorique de simplification de la trace à l'aide du mécanisme d'abstraction par spécialisation - généralisation

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	CC_B2	CC_A1	CC_B1	CC_A2	CC_C1	CC_B2	CC_A2
1er mécanisme	CA_B	CA_A	CA_B	CA_A	CC_C1	CA_B	CA_A

Ensuite nous pouvons appliquer le mécanisme d'abstraction par composition pour apporter un niveau de simplification supplémentaire.

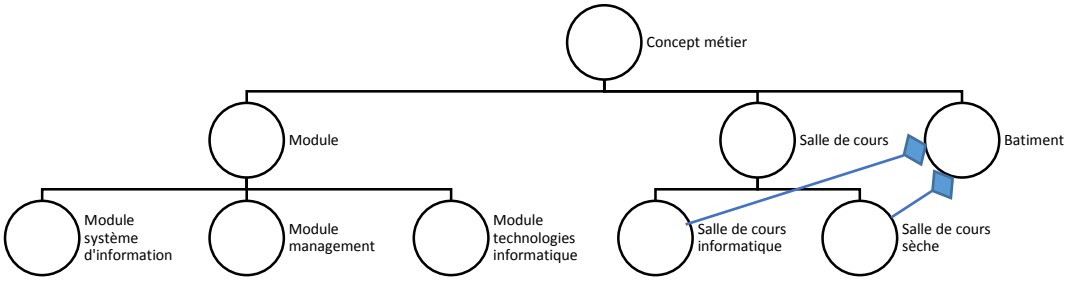
Figure 19 : Exemple théorique de simplification de la trace à l'aide du mécanisme d'abstraction par composition

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	CC_B	CA_A	CC_B	CA_A	CC_C1	CA_B	CA_A
2ème mécanisme	CC_C1	CA_A	CC_C1	CA_A	CC_C1	CC_C1	CA_A

5.1.3.3 Exemple concret

Pour les exemples concrets qui vont suivre le domaine des « Etude » sera utilisé pour schématiser les explications.

Figure 20 : Exemple concret de la représentation hiérarchique d'une ontologie



Voici ci-dessous les abréviations des noms de concepts utilisés pour l'exemple.

MSI : Module système d'information

MM : Module management

MTI : Module technologies informatiques

SCI : Salle de cours informatique

SCS : Salle de cours sèche

Bâtiment : B

SC : Salle de cours

Figure 21 : Exemple concret de simplification de la trace à l'aide du mécanisme d'abstraction par spécialisation - généralisation

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	SCI	MSI	SCS	MM	SCI	MTI	SCS
1er mécanisme	SC	M	SC	M	SC	M	SC

En ajoutant le mécanisme d'abstraction par composition :

Lorsque le composant « Salle de cours informatique » sera évoquée, nous pourrons faire le lien avec « Bâtiment » car nous savons qu'une « Salle de cours » est un composant de « Bâtiment ».

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	SCI	MSI	SCS	MM	SCI	MTI	SCS
1er mécanisme	SC	M	SC	M	SC	M	SC
2ème mécanisme	B	M	B	M	B	M	B

Le 1er niveau de simplification concerne le mécanisme d'abstraction par spécialisation – généralisation. Le 2ème niveau représente la simplification avec la mise en œuvre du mécanisme d'abstraction par composition.

La mise en œuvre de ces deux mécanismes nous permet d'observer que B et M se suivent de manière régulière dans la trace. Le couple {B ; M} apparait 3 fois dans la trace.

5.1.4 Recherche de pattern de concepts par simplification

À l'aide de la combinaison des deux mécanismes d'abstraction nous pouvons obtenir depuis la trace des patterns de concepts qui, sinon, auraient été impossible à observer.

5.1.4.1 Exemple théorique

Figure 22 : Représentation hiérarchique de l'ontologie

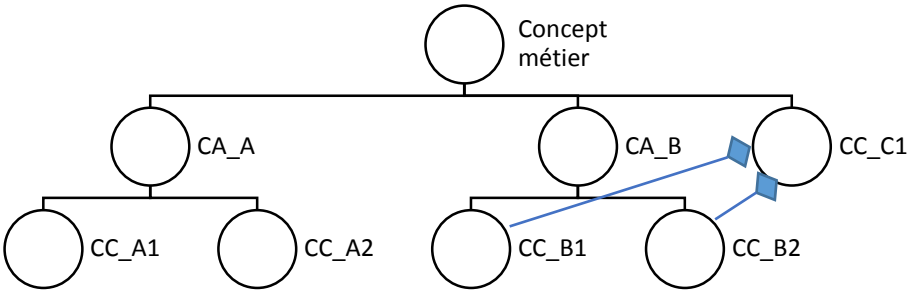


Figure 23 : Simplification de la trace à l'aide des deux mécanismes d'abstraction pour la recherche de pattern de concepts

	Trace						
Appel N°	1	2	3	4	5	6	7
Concepts	CC_B2	CC_A1	CC_B1	CC_A2	CC_C1	CC_B2	CC_A2
1er mécanisme	CA_B	CA_A	CA_B	CA_A	CC_C1	CA_B	CA_A
2ème mécanisme	CC_C1	CA_A	CC_C1	CA_A	CC_C1	CC_C1	CA_A

Le 1er mécanisme de simplification concerne le mécanisme d'abstraction par spécialisation – généralisation. Le 2ème niveau représente la simplification avec la mise en œuvre du mécanisme d'abstraction par composition.

La mise en œuvre de ces deux mécanismes nous permet d'observer que CC_C1 et CA_A se suivent de manière régulière dans la trace. Ce qui nous permet de simplifier de manière significative le nombre de concepts apparaissant. Avec la réduction du nombre de concepts, nous allons plus facilement faire le lien entre le code et le métier. Cependant, cet outil reste un outil d'aide à la compréhension, de ce fait il faudra s'assurer de la validité sémantique des patterns obtenus et observer s'il y a un réel lien entre les différents concepts.

5.1.5 Compression de la trace

La compression de la trace nous permettra de réduire la quantité d'information disponible. Pour cela, nous devrons calculer le nombre d'occurrences continues d'un concept, ce qui nous permettra d'effectuer une compression.

Figure 24 : Exemple de trace non compressé

	Trace							
Appel N°	1	2	3	4	5	6	7	8
Concepts	CC_C1	CA_A	CC_C1	CA_A	CC_C1	CC_C1	CC_C1	CA_A

Dans la Figure 24 nous pouvons observer une répétition continue du concept CC_C1. Ce qui signifie que plusieurs méthodes font référence au même concept métier. En effectuant une compression nous obtiendrons une simplification de la représentation de l'information.

Figure 25 : Exemple de compression de la trace

	Trace					
Appel N°	1	2	3	4	5	6
Occurrence continue	1	1	1	1	3	1
Concepts	CC_C1	CA_A	CC_C1	CA_A	CC_C1	CA_A

Nous pouvons observer dans la Figure 25 une réduction du nombre de représentations des concepts tout en gardant l'ordre d'appel ainsi que le nombre d'occurrence continue des concepts. Ce qui nous permet d'obtenir une simplification de la trace.

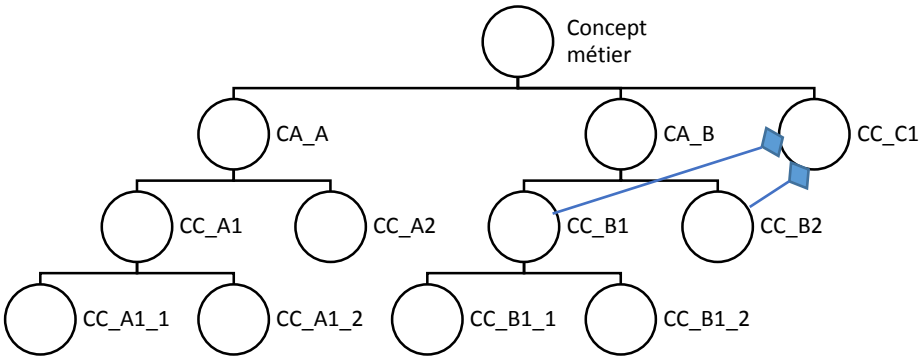
5.1.6 Niveau d'abstraction

Le choix du niveau d'abstraction va déterminer lors du processus d'abstraction jusqu'à quel niveau nous allons chercher le concept général dans le cas du mécanisme de spécialisation – généralisation et le concept de composition pour le mécanisme de composition.

Ce choix va nous permettre de fixer le niveau de granularité que l'on souhaite lors des mécanismes d'abstraction.

Le niveau d'abstraction est le résultat de la profondeur d'héritage de l'ontologie et du nombre de compositions existantes. Il sera découpé en deux parties, jusqu'à un certain niveau le mécanisme de généralisation – spécialisation sera appliqué et par la suite le mécanisme de composition.

Figure 26 : Représentation hiérarchique de l'ontologie



Dans l'ontologie représentée à la Figure 26 nous pouvons observer trois niveaux d'héritage et un niveau de composition. Donc nous aurons en tout cinq niveaux d'abstraction. Le niveau 0 correspond à l'application d'aucun mécanisme d'abstraction, du niveau un à quatre l'application du mécanisme de généralisation - spécialisation et pour finir le niveau 5 correspond à l'application du mécanisme de composition après celui de généralisation – spécialisation.

6. Outil et Analyse

Dans cette partie, il sera question de l'outil que j'ai développé ainsi que les différents résultats obtenus à l'aide de ce dernier. En effet, cet outil implémente les mécanismes d'abstraction décrient aux chapitres 5.1.1 et 5.1.2 ainsi que la compression de la trace décrit au chapitre 5.1.5.

A l'aide de cet outil, je commencerai par effectuer différents tests pour montrer l'effet des mécanismes et de la sensibilité du niveau d'abstraction. Ensuite, je démontrai l'importance de la compression en effectuant une comparaison de la trace avant et après la compression. Enfin, je ferai des tests concernant la sensibilité de la force de correspondance entre les méthodes et les concepts.

6.1 Description de l'outil

Lors du développement de cet outil, j'ai décidé de l'intégrer dans l'application HEG Trace Analyzer pour bénéficier de la technique de segmentation développé par la HEG ainsi que diverses Class et méthodes existantes. Cet outil est paramétrable par l'utilisateur de manière à ce qu'il puisse analyser la trace avec des possibilités différentes.

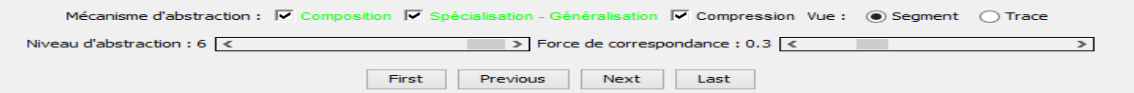
6.1.1 Objectif de l'outil

Le but de cet outil est d'appliquer les différents mécanismes expliqués précédemment. Cet outil permettra d'effectuer une simplification de la trace d'exécution afin d'obtenir une représentation graphique de cette dernière dans le but d'apporter des informations aidants à la compréhension du logiciel.

6.1.2 Présentation générale de l'interface utilisateur

L'interface utilisateur de cet outil est composée d'une partie configuration avec différents paramètres qui devront être configurés selon le résultat que l'on souhaite obtenir.

Figure 27 : Interface utilisateur de l'outil



- Mécanisme d'abstraction
  - Spécialisation – généralisation
    - Case à cocher permettant l'activation du mécanisme d'abstraction par spécialisation – généralisation décrit au chapitre 5.1.1
  - Composition
    - Case à cocher permettant l'activation du mécanisme d'abstraction par composition décrit au chapitre 5.1.2
- Force de correspondance
  - Ascenseur permettant de définir le degré de correspondance du concept pour la méthode. La force de correspondance défini la probabilité relationnelle entre concept et méthode.
- Niveau d'abstraction
  - Ascenseur permettant de définir le niveau d'abstraction pour les mécanismes d'abstraction décrit au chapitre 5.1.6.
- Compression
  - Case à cocher permettant l'activation la simplification de la trace. Compression de la trace en supprimant une suite redondante de concept décrit au chapitre 5.1.5.
- Vue
  - Bouton radio permettant de changer la représentation.
    - Segment : permet d'avoir une représentation par segment en conservant l'ordre d'appel des concepts.
    - Trace : permet d'avoir une représentation de la trace avec le nombre d'occurrence de chaque concept par segment.
- First
  - Permet d'afficher le premier segment
- Next
  - Permet d'afficher le segment suivant
- Previous
  - Permet d'afficher le segment précédent
- Last
  - Permet d'afficher le dernier segment

6.2 Résultats obtenus

Les résultats qui seront présentés dans la suite des chapitres ont été obtenus suite à l'exploitation d'une trace d'une taille de 100'000 appels qui a été segmentée en 100 segments. Il faut environ 15 secondes de traitement pour lier les méthodes aux concepts, puis en faire une représentation graphique.

6.2.1 Analyses

Les analyses générales seront faites avec le paramétrage suivant :

- Force de correspondance : 0.3
- Niveau d'abstraction : maximum, 6
- Compression : Activé

6.2.1.1 Analyse des résultats avec le mécanisme de spécialisation – généralisation

Les résultats que je vais présenter montrent que le mécanisme de spécialisation – généralisation nous permet de remonter dans la hiérarchie d'héritage de l'ontologie pour obtenir les concepts parents. Nous pouvons observer une légère modification dans le nombre d'occurrences par concept. En effet, à l'aide de ce mécanisme nous avons pu regrouper deux concepts et ainsi effectuer une légère simplification.

Figure 28 : Segment 15 sans mécanisme d'abstraction

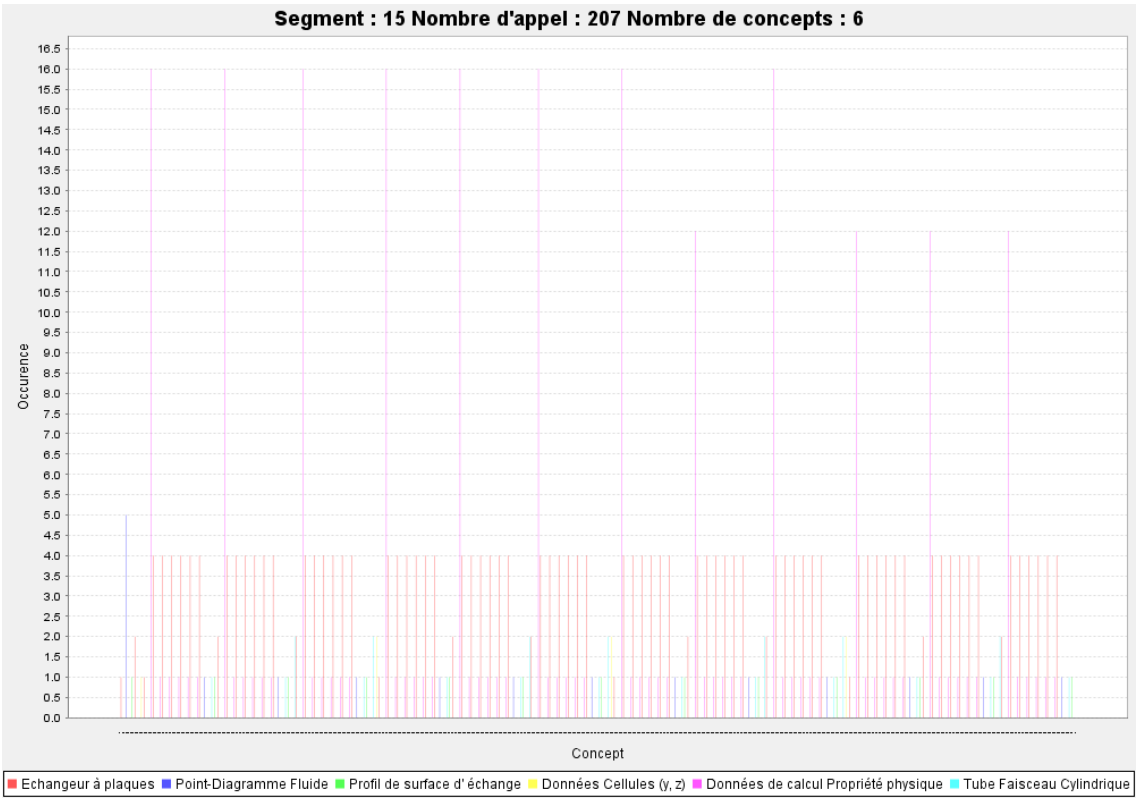


Figure 29 : Segment 15 avec le mécanisme de généralisation - spécialisation

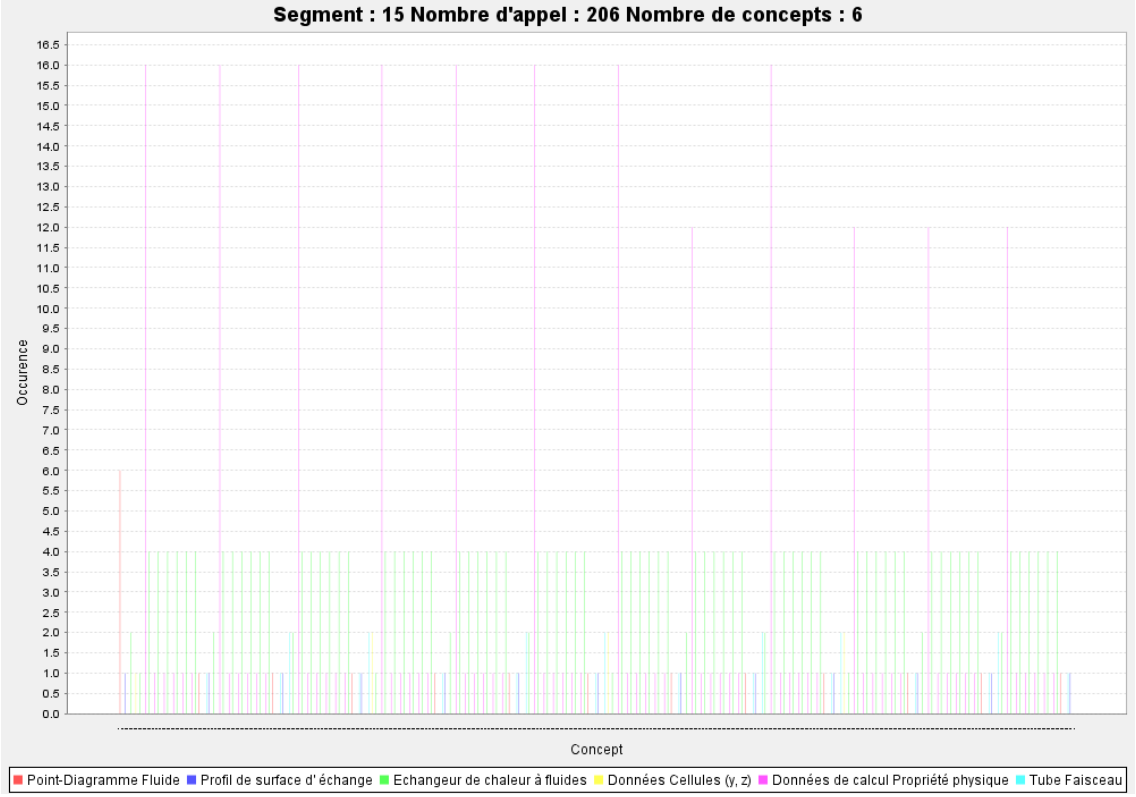


Tableau 1 : Comparatif des concepts avec et sans le mécanisme de généralisation spécialisation

Sans le mécanisme	Avec le mécanisme
Point-Diagramme Fluide	Point-Diagramme Fluide
Profil de surface d'échange	Profil de surface d'échange
Echangeur à plaques	Echangeur de chaleur à fluide
Données Cellules (y,z)	Données Cellules (y,z)
Données de calcule Propriété physique	Données de calcule Propriété physique
Tube faisceau cylindrique	Tube faisceau

A l'aide de ce mécanisme nous avons pu obtenir deux concepts parents, donc des concepts plus généraux que nous pourrons plus facilement exploiter par la suite.

Nous pouvons observer une variation dans le nombre d'occurrence pour le concept Point-Diagramme, ce qui signifie qu'à l'aide du mécanisme de spécialisation – généralisation nous avons pu regrouper deux concepts spécialisés en un concept parent pour obtenir une simplification.

6.2.1.2 Analyse des résultats avec le mécanisme de composition

Les résultats que je vais présenter montrent que le mécanisme de composition nous permet de réduire considérablement le nombre de concepts présents dans le segment.

Figure 30 : Segment 15 sans mécanisme d'abstraction

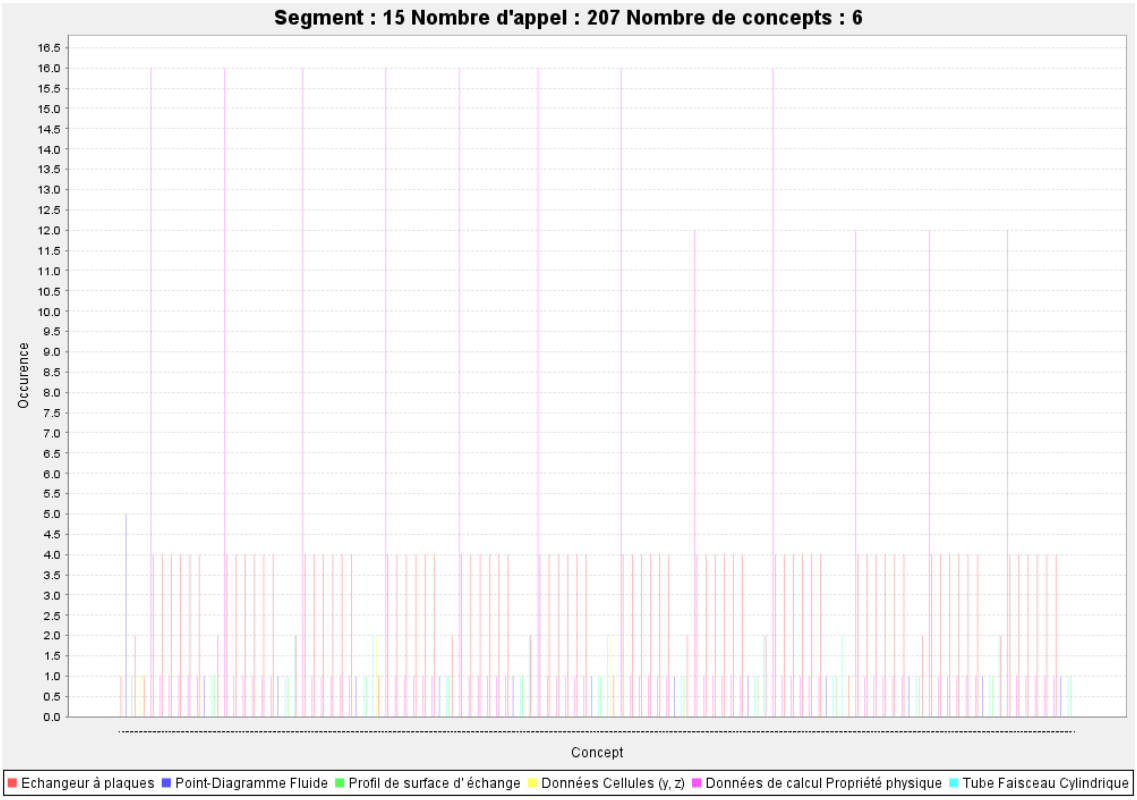
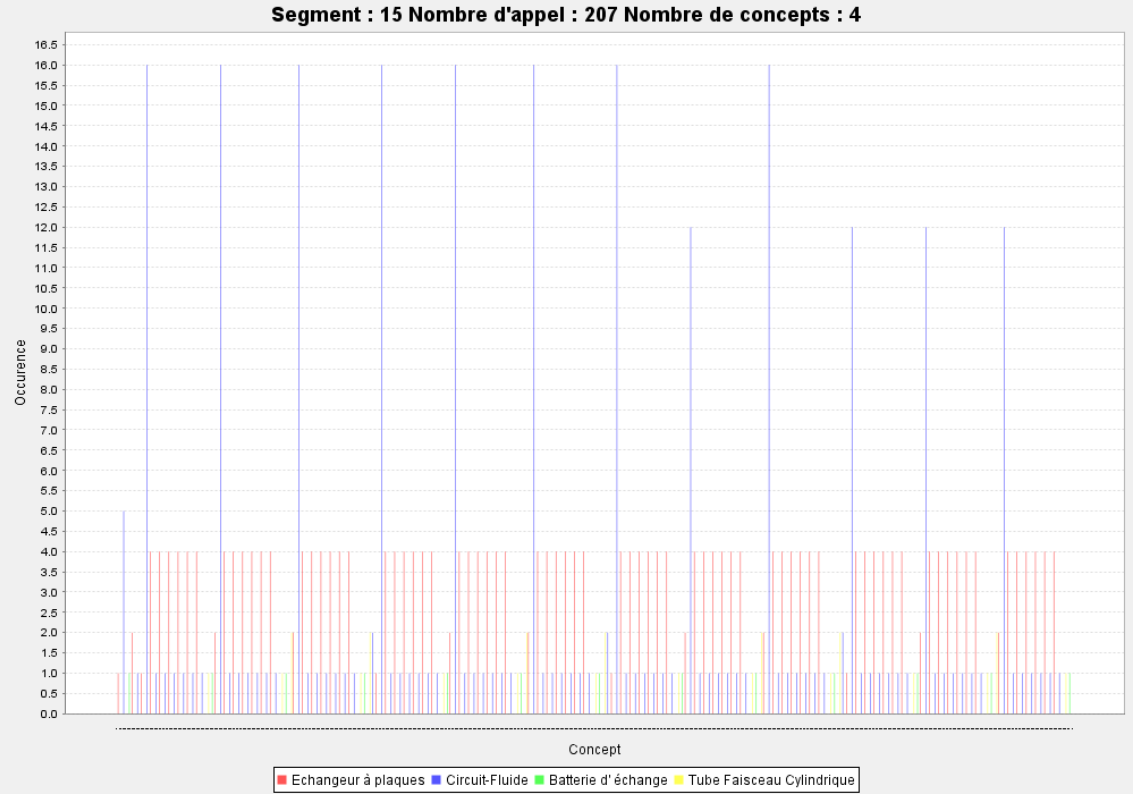


Figure 31 : Segment 15 avec le mécanisme de composition



Nous pouvons observer que nous sommes passés de six concepts à quatre concepts à l'aide de ce mécanisme. Ce qui nous permet de faire une simplification importante du nombre de concepts représentés.

Tableau 2 : Comparatif des concepts avec et sans le mécanisme de composition

Sans le mécanisme	Avec le mécanisme
Point-Diagramme Fluide	Echangeur à plaques
Profil de surface d'échange	Circuit-Fluide
Echangeur à plaques	Batterie d'échange
Données Cellules (y,z)	Tube faisceau cylindrique
Données de calcule Propriété physique	
Tube faisceau cylindrique	

6.2.1.3 Comparaison des résultats avec et sans compression

Dans cette partie nous allons observer les résultats avec et sans compression du segment, ce qui nous permettra de nous rendre compte de l'importance de la compression.

Figure 32 : Segment 2 non compressé

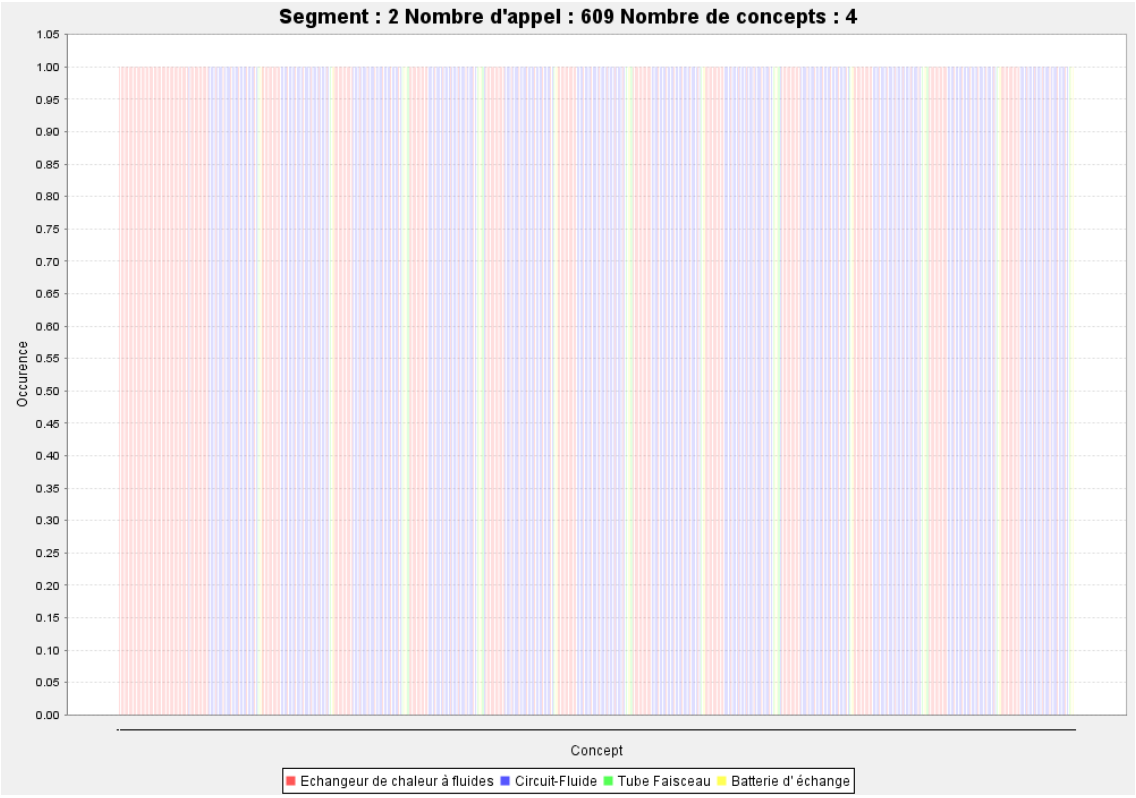


Figure 33 : Segment 2 compressé

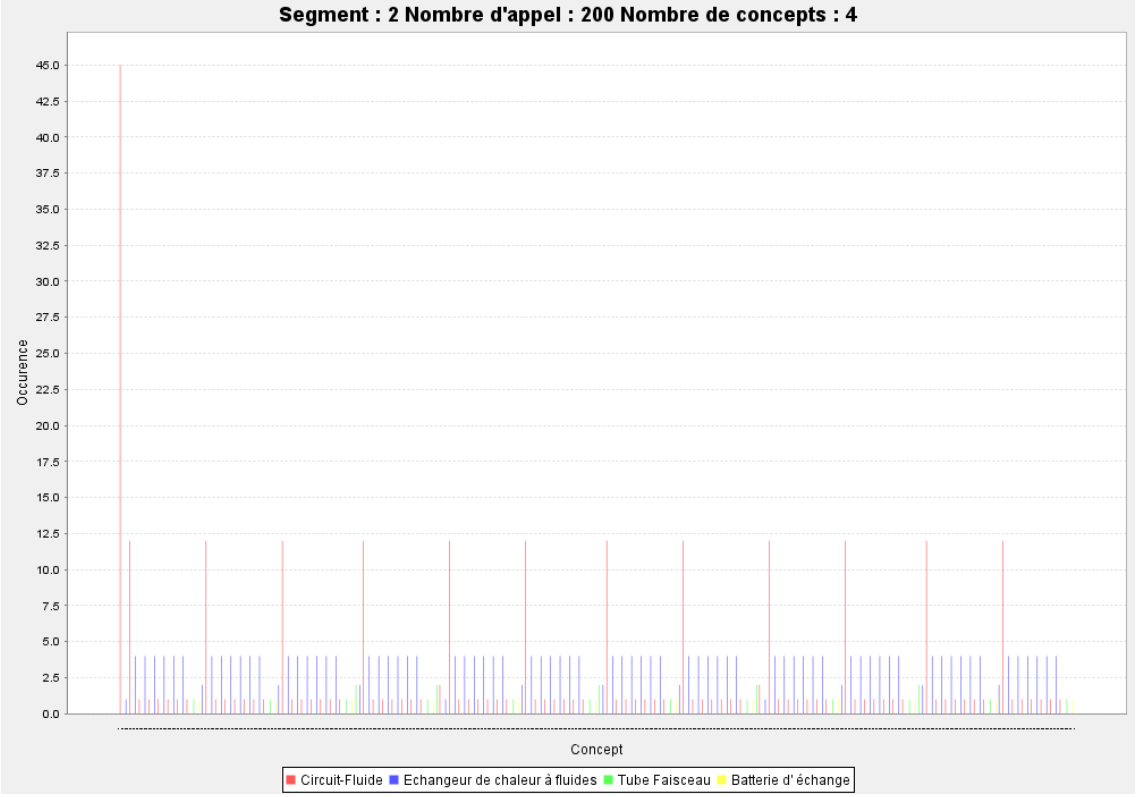


Tableau 3 : Comparatifs du nombre d'appels avec et sans la compression

	Sans compression	Avec compression
Nombre d'appels	609	200

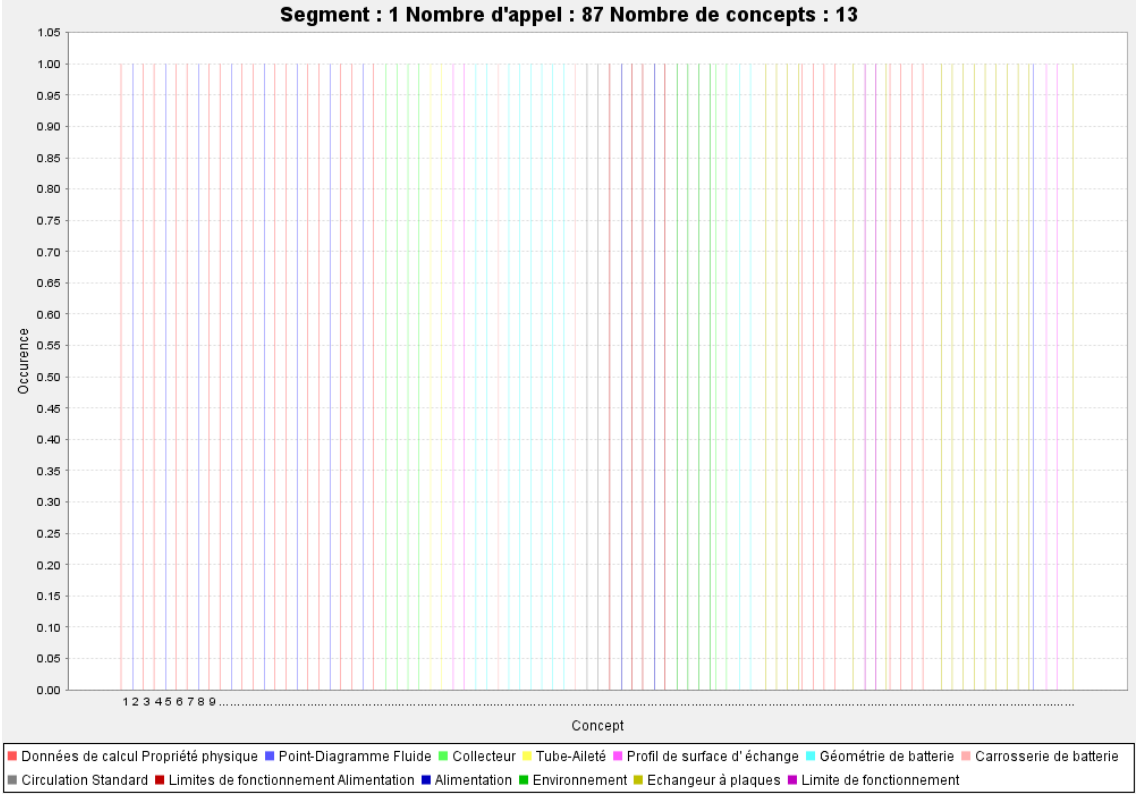
La compression du nombre d'occurrence nous permet de grandement simplifier la représentation du segment avec ses différents concepts. Nous gardons le nombre d'occurrences ainsi que l'ordre d'appel des concepts. Nous obtenons une réduction de 67% du nombre d'appels à l'aide de la compression pour le segment 2. Cette dernière facilite la lecture et la compréhension de la représentation visuelle. En effet nous pouvons observer facilement une certaine récurrence d'appels des différents concepts mais cette partie sera traitée dans le chapitre concernant la recherche de patterns de concepts au point 6.2.2.

6.2.1.4 Analyse des résultats avec les deux mécanismes et la compression

Les résultats que je vais présenter montrent que les deux mécanismes d'abstraction combinés apportent une simplification de l'information significative en réduisant le nombre de concepts différents par segment et l'application de la compression permet de réduire le nombre d'appel successifs du même concept, de plus elle permet de mettre en évidence certaines suite de concepts (pattern de concepts). Cette simplification facilite la compréhension et la lecture de la représentation visuelle sous forme de graphique, sans cette simplification nous pouvons observer une certaine difficulté de compréhension et de lecture du aux surplus d'informations.

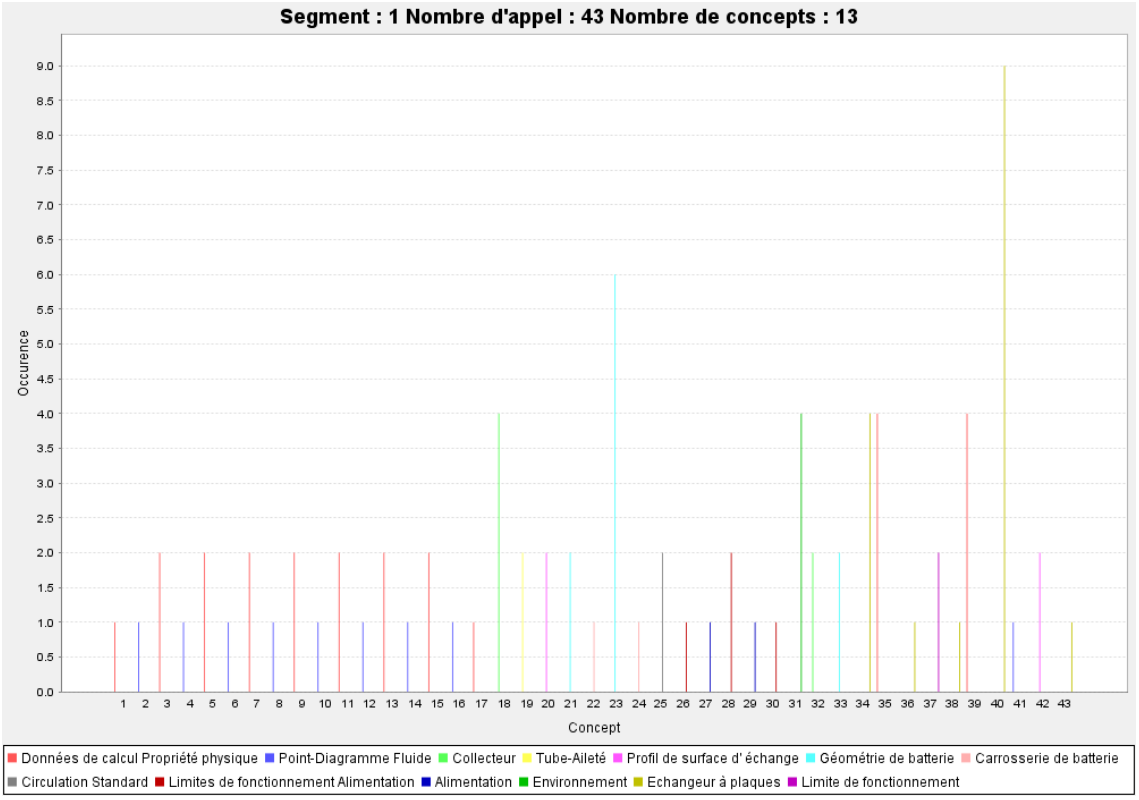
La première partie des résultats concerneront le segment 1 et la seconde partie le segment 3.

Figure 34 : Segment 1 sans mécanisme d'abstraction et sans compression



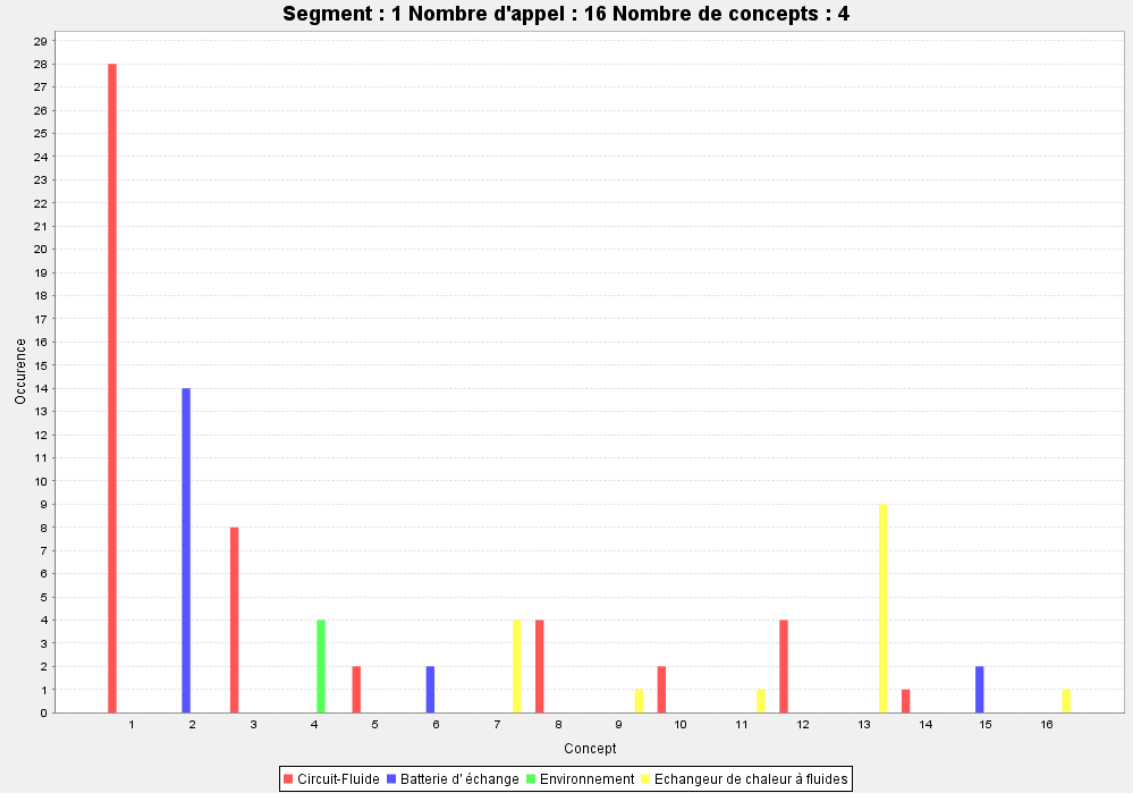
Sur la Figure 34 nous pouvons observer le segment 1 sous une forme brute c'est-à-dire sans mécanisme d'abstraction ni compression. Nous observons l'enchainement des différents concepts qui sont aux nombres de 13 sur 87 appels.

Figure 35 : Segment 1 sans mécanisme d'abstraction et avec compression



L'application de la compression décrit au point 5.1.5 nous permet de réduire considérablement le nombre d'appels nous passons d'un état initial de 87 appels pour 43 appels après compression, donc une réduction de 51% du nombre d'appels.

Figure 36 : Segment 1 avec les deux mécanismes d'abstraction et avec la compression



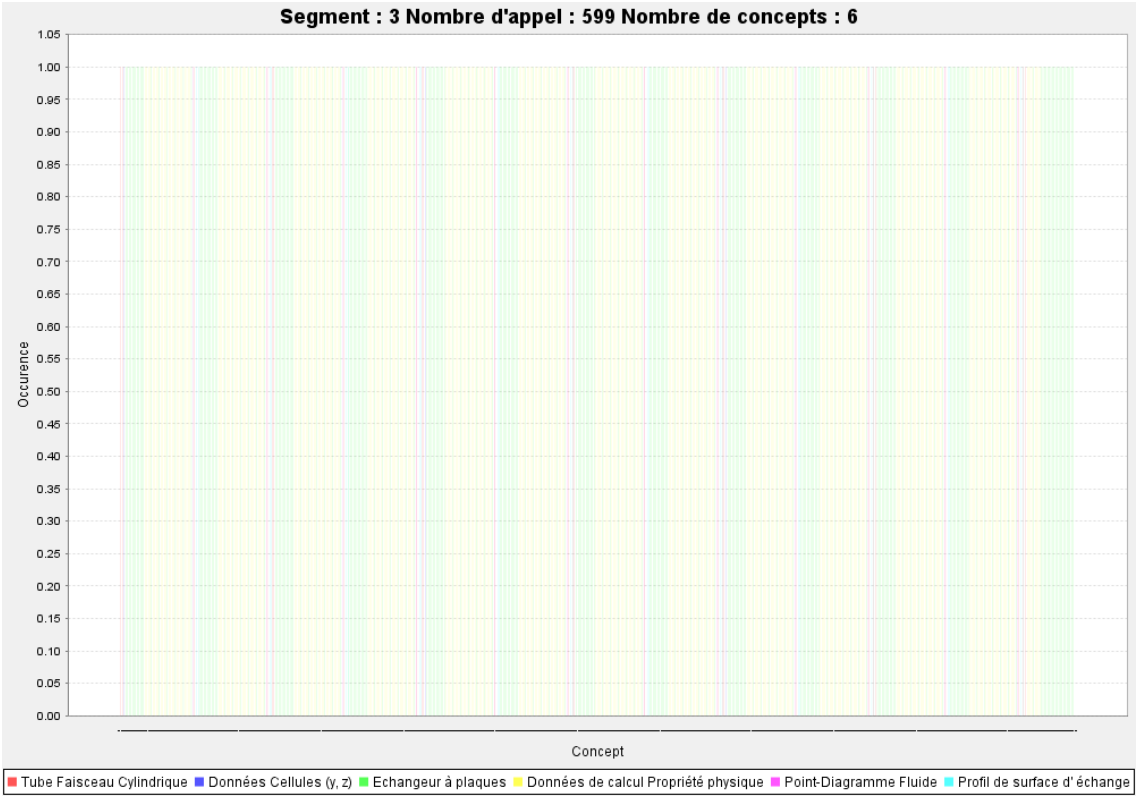
Nous pouvons observer que pour le segment 1 sans aucun mécanisme d'abstraction et de compression nous avons 87 appels pour 13 concepts. Par la suite avec l'aide de la compression nous obtenons 43 appels pour 13 concepts différents. Pour finir, avec les deux mécanismes d'abstraction et la compression nous obtenons 4 concepts différents pour 16 appels. Nous obtenons donc une simplification relativement importante en termes de réduction de la quantité d'information et une facilitation de lecture et de compréhension.

Tableau 4 : Comparatifs du nombre d'appels et du nombre de concepts pour le segment 1

	Sans mécanisme et sans compression	Sans mécanisme et avec compression	Avec mécanisme et avec compression
Nombre d'appels	87	43	16
Nombre de concepts	13	13	4

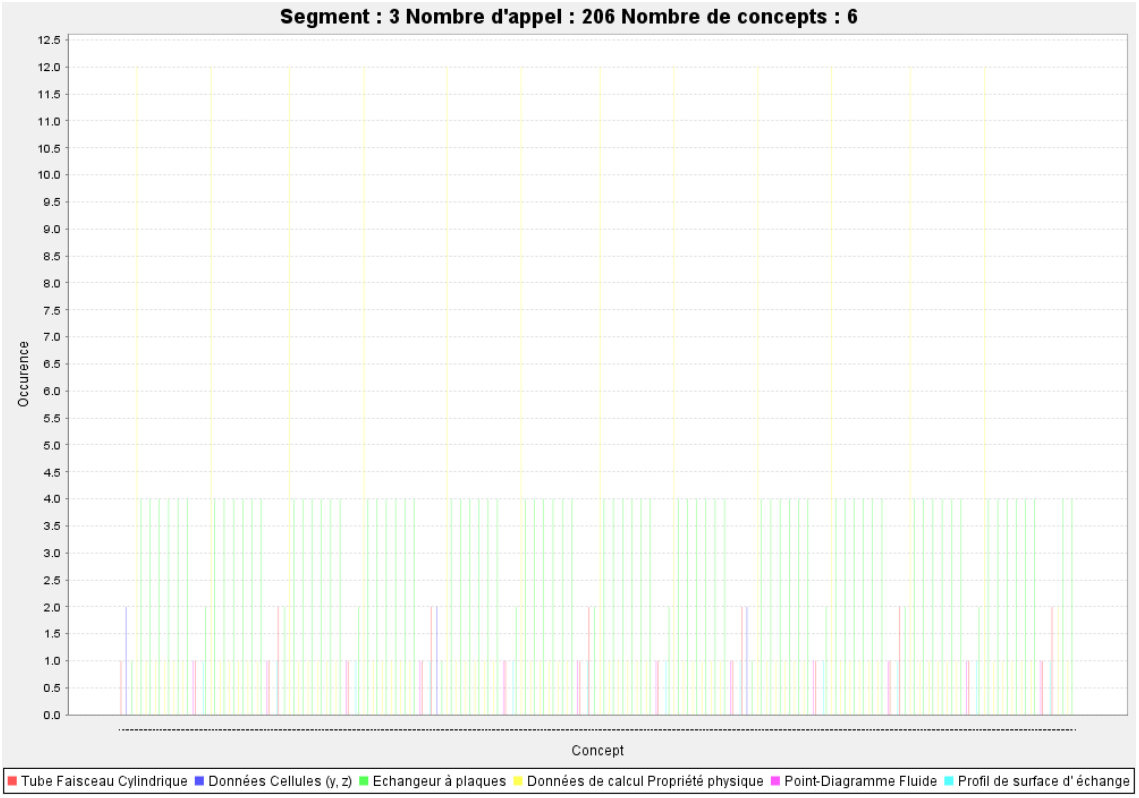
Nous pouvons observer pour le segment 1 une réduction de 81% du nombre d'appels en passant de 87 à 16 appels et une diminution de 70% du nombre de concept en passant de 13 à 4. Les deux mécanismes combinés avec la compression nous permettent de réduire d'une manière important la quantité d'information que nous avons pour la rendre plus compréhensible et plus facilement lisible.

Figure 37 : Segment 3 sans mécanisme d’abstraction et sans compression



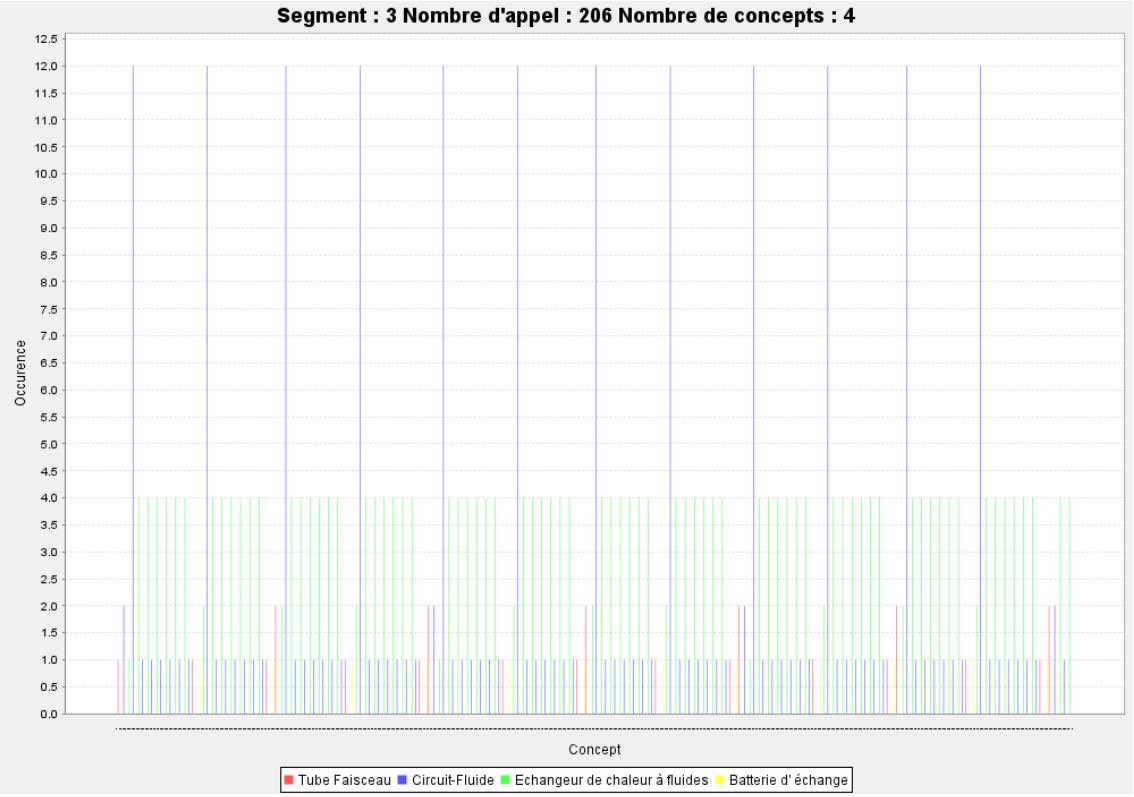
Sur la Figure 37 nous pouvons observer le segment 3 sans mécanisme d’abstraction ni compression. Nous observons l’enchainement des différents concepts qui sont aux nombres de 6 sur 599 appels.

Figure 38 : Segment 3 sans mécanisme d’abstraction et avec compression



L’application de la compression décrit au point 5.1.5 nous permet de réduire considérablement le nombre d’appels nous passons d’un état initial de 599 appels pour 206 appels après compression, donc une réduction de 65% du nombre d’appels.

Figure 39 : Segment 3 avec les mécanismes d'abstraction et avec la compression



Nous pouvons observer que pour le segment 3 sans aucun mécanisme d'abstraction et de compression nous avons 599 appels pour 6 concepts. Par la suite avec l'aide de la compression nous obtenons 206 appels pour 6 concepts différents. Pour finir, avec les deux mécanismes d'abstraction et la compression nous obtenons 4 concepts différents pour 206 appels. Nous obtenons donc une réduction de la quantité d'information qui va nous permettre de faciliter la lecture et la compréhension.

Tableau 5 : Comparatifs du nombre d'appels et du nombre de concepts pour le segment 3

	Sans mécanisme et sans compression	Sans mécanisme et avec compression	Avec mécanisme et avec compression
Nombre d'appels	599	206	206
Nombre de concepts	6	6	4

Nous pouvons observer une diminution du nombre d'appels ainsi que du nombre de concepts. Nous passons de 599 à 206 appels ce qui signifie une diminution de 65% ce qui est rendu possible à l'aide de la compression puis une diminution du nombre de concept de 6 à 4 concepts ce qui correspond à une réduction de 33% du nombre de concepts.

Nous pouvons observer une certaine efficacité de la combinaison des mécanismes d'abstraction de la compression qui nous permet de réduire la quantité d'information en termes d'appels et du nombre de concepts, de plus elle nous permet d'observer des éventuels patterns de concept.

Tableau 6 : Résumé des résultats obtenu sur le segment 1 et le segment 3

	Segment 1	Segment 3
Réduction du nombre d'appels en %	81%	65%
Réduction du nombre de concepts en %	70%	33%

Les résultats du segment 3 sont plus représentatifs des segments suivants car le segment 1 regroupe les méthodes du « Main » en Java qu'on ne peut pas lier à des concepts métiers.

Si nous devions appliquer les pourcentages obtenus du segment 3 à la trace, donc une trace de 100'000 appels, nous aurions plus que 35'000 appels avec une réduction de 65% du nombre d'appels total.

6.2.1.5 Analyse de la sensibilité du niveau d'abstraction

Le niveau d'abstraction selon le niveau défini il va nous permettre de trouver remonter dans la hiérarchie de concepts afin d'obtenir des concepts généraux. Comme expliqué au point 5.1.6 le niveau d'abstraction est composé de de la profondeur d'héritage de l'ontologie et du nombre de composition existante. A l'aide d'un algorithme que j'ai développé j'ai pu estimer la profondeur de l'ontologie à 2 pour le mécanisme de généralisation – spécialisation et de 4 pour la composition, ce qui nous fait 7 niveau possible en partant de 0.

Tableau 7 : Comparatifs du nombre d'appels et du nombre de concepts par niveau d'abstraction pour le segment 3

Niveau d'abstraction	0	1	2	3	4	5	6
Sans compression							
Appels	599	599	599	599	599	599	599
Nombre de concepts	6	6	6	6	4	4	4
Avec compression							
Appels	206	206	206	206	206	206	206
Nombre de concepts	6	6	6	6	4	4	4

Nous pouvons remarque que le niveau d'abstraction fait une légère réduction du nombre de concepts dès le niveau 4, ce qui correspondant à l'activation des deux mécanismes et lorsque l'on commence à récupérer des compositions assez générales.

Tableau 8 : Comparatifs du nombre d'appels et du nombre de concepts par niveau d'abstraction pour le segment 1

Niveau d'abstraction	0	1	2	3	4	5	6
Sans compression							
Appels	87	87	87	87	87	87	87
Nombre de concepts	13	13	13	7	5	4	4
Avec compression							
Appels	43	43	43	35	18	16	16
Nombre de concepts	13	13	13	7	5	4	4

Ce que l'on peut constater pour le segment 1 c'est lorsque nous dépassons le niveau 2 d'abstraction avec compression nous obtenons une diminution progressive du nombre de concept en passant de 7 à 5 puis 4 pour finir.

On peut observer pour les deux segments analysés réagissent à partir du niveau 3. Cependant le segment 3 obtient rapidement le nombre de concepts minimaux possible contrairement au segment 1 qui doit monter jusqu'au 5ème niveau d'abstraction.

6.2.1.6 Analyse de la sensibilité du niveau de la force de correspondance

La force de correspondance permet de définir la probabilité relationnelle entre méthodes et concepts. Il s'agit d'un paramètre important qu'il faut prendre en compte. En effet, une méthode peut avoir plusieurs concepts métiers mais ces derniers possèdent tous une force de correspondance différente.

Lorsque la force de correspondance est élevée, le nombre de concepts liés à une méthode diminue car nous souhaitons obtenir une correspondance plus importante entre ces derniers. Contrairement à une force de correspondance faible qui augmentera le nombre de concepts liés à une méthode.

Il est important de trouver un équilibre pour ne pas avoir un surplus d'informations ou au contraire un manque d'informations.

Tableau 9 : Comparatifs du nombre d'appels et du nombre de concepts selon la force de correspondance pour le segment 3

Force de correspondance	0.1	0.2	0.3	0.4	0.5
Sans mécanisme et sans compression					
Appels	5267	1503	599	44	0
Nombre de concepts	39	19	6	2	0
Sans mécanisme et avec compression					
Appels	4081	972	206	6	0
Nombre de concepts	39	16	6	2	0
Avec mécanisme et avec compression					
Appels	3149	856	206	6	0
Nombre de concepts	10	8	4	2	0

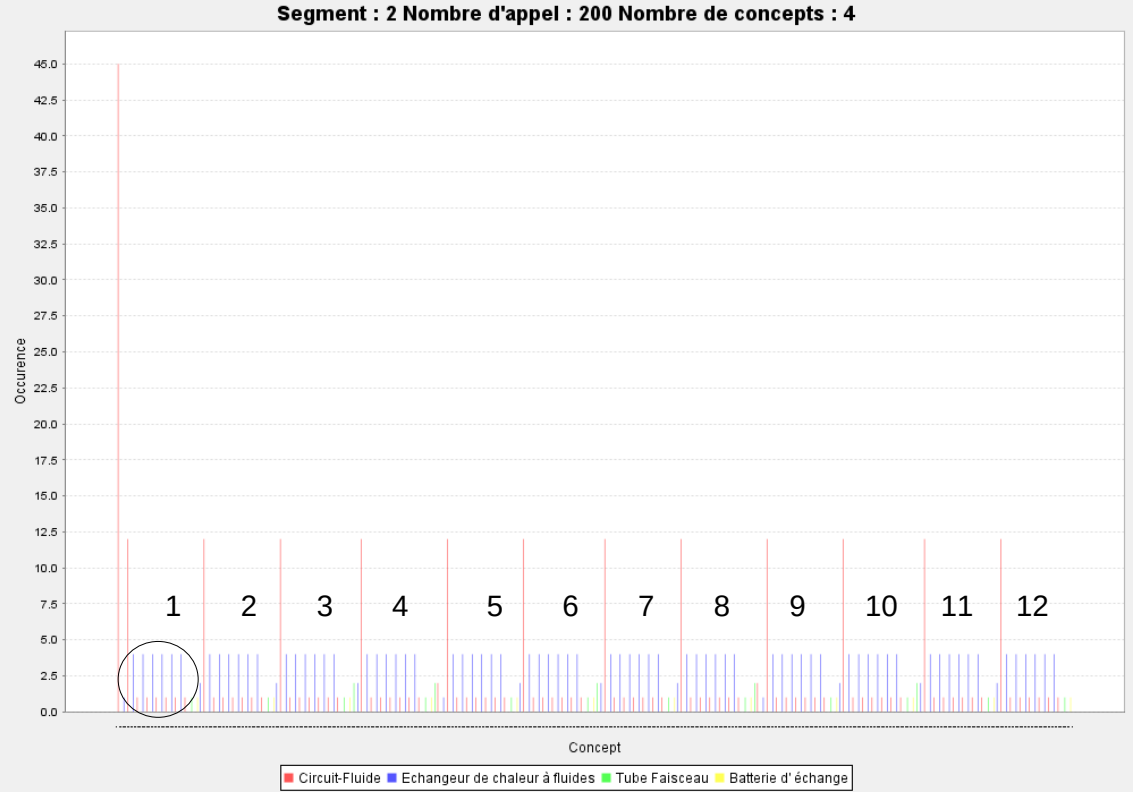
Nous pouvons observer la sensibilité de la force de correspondance en observant les deux extrêmes 0.1 et 0.5 qui nous font passer de 3149 appels et 10 concepts en ayant les mécanismes et la compression pour 0 appel et 0 concept. De plus avec une force de correspondance trop basse nous diminuons la performance des mécanismes et de la compression dû à l'abondance d'informations, notamment à une diminution d'enchainement successive de concepts la compression et moins efficace.

Suites aux résultats obtenus présents sur le Tableau 9, j'ai pu conclure que les forces de correspondance optimales pour cette trace sont 0.2 et 0.3 car nous n'avons ni de surplus ni de manque d'informations. Avec ces dernières nous obtenons de bons résultats dans la simplification et qui reste lisible et compréhensible. Mais Il serait judicieux d'opter pour une force de correspondance à 0.3 pour avoir une certaine cohérence entre les méthodes et les concepts.

6.2.2 Patterns de concepts

Suite à l'application des mécanismes d'abstraction et de la compression, nous pouvons observer d'éventuels patterns de concepts par segment.

Figure 40 : Observation de pattern de concept dans le segment 2



Nous pouvons observer à l'œil une certaine répétions de différents concepts dans le segment 2. Il y a en tout douze. Il serait intéressant pour la suite de faire une simplification supplémentaire. En effet dans le cas où on observer un segment qui serait principalement composé d'une succession pattern de concepts, nous pourrions appliquer une compression supplémentaire sur le segment pour réduire d'avantage le nombre d'appels et limite d'une manière significative la quantité d'information disponible.

Figure 41 : Observation de pattern de concept dans le segment 5

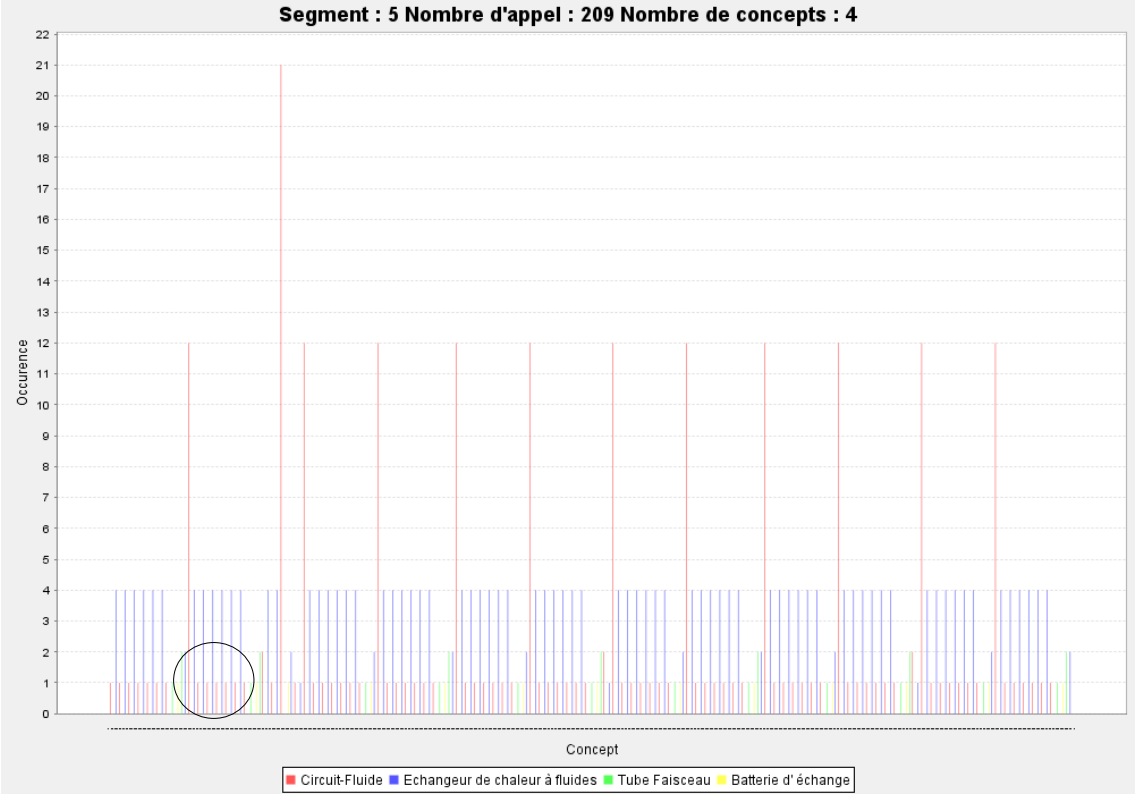
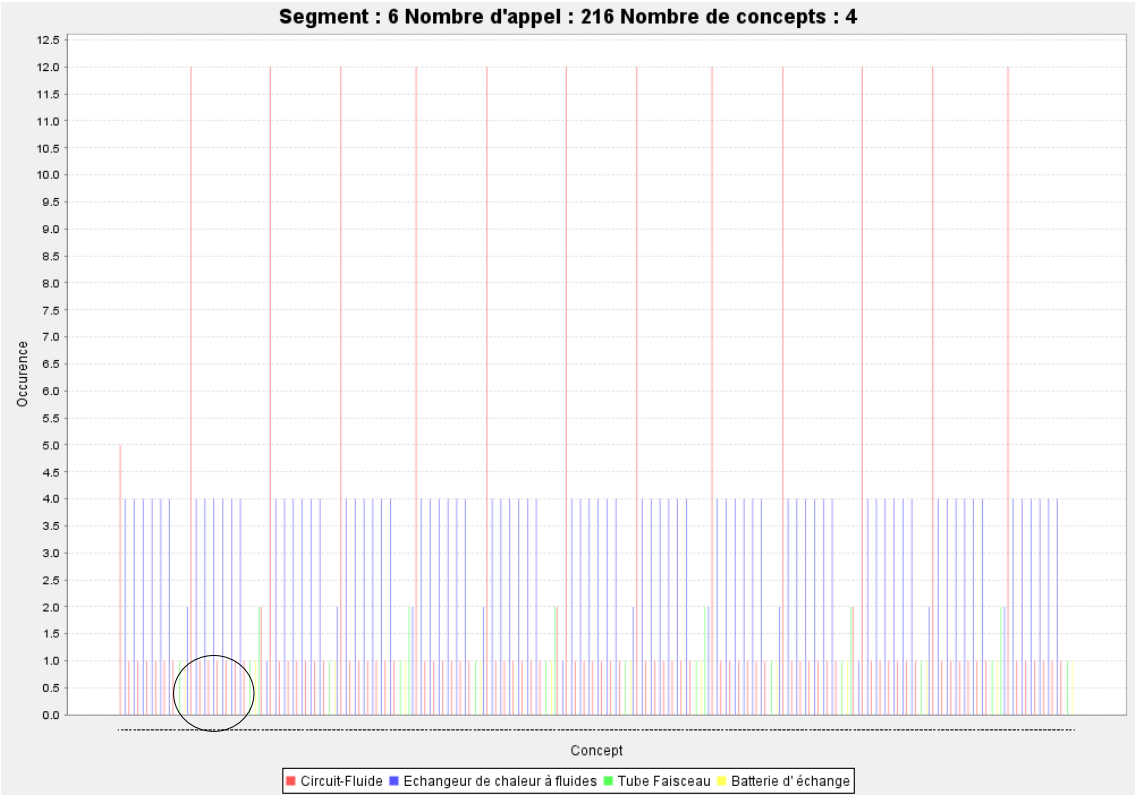


Figure 42 : Observation de pattern de concept dans le segment 6



Nous pouvons observer le même pattern de concepts dans le segment 5 à la Figure 41, le segment 2 à la Figure 40 ainsi que sur la Figure 42 représentant le segment 6. Par la suite il serait intéressant d'effectuer une analyse sur toute trace pour chercher si d'autres segments possèdent le même pattern de concepts. En effet avec une répétions régulière du pattern nous pourrions conclure sur une relation entre les différents concepts se trouvant dans le pattern. Ce qui nous permettrait selon la cohérence du pattern de regrouper cette suite de concepts et de réduire d'avantage la quantité d'information présente. Cependant il faudrait avant tout vérifier la pertinence du pattern et la possibilité d'appliquer une telle simplification sur la trace.

7. Architecture de l'implémentation

Dans cette partie, la manière dont l'outil est composé avec ses différents packages et classes sera détaillée.

- Le package « db » contient la classe « DBAbstractionConcept » qui s'occupe de se connecter à la base de données et contient les différentes requêtes pour cette dernière.
- Le package « object » contient 5 classes :
    - La classe « ConceptPattern » contient les différents patterns de concepts mais elle n'est pas utilisée à ce stade de l'outil car la recherche de pattern ne fonctionne pas de manière optimale
    - La classe « Concept » contient les différents concepts présent dans la trace.
    - La classe « MethodIdentifierWhitConcept » contient la liaison des méthodes et concepts.
    - La classe « Segment » contient toutes les informations relatives à un segment.
    - La classe « SegmentWithConcept » contient un segment et les différents concepts faisant partie du segment.
- Le package « analysisConceptMethod » est le cœur de l'application, elle contient 2 classes :
    - La classe « AbstractionConcept » permet de faire l'affichage et de gérer les actions de l'utilisateur.
    - La classe « AbstractionConceptModel » contient les différents algorithmes de compression, d'abstraction et de création des représentations graphiques.

8. Conclusion

L'application des mécanismes d'abstraction avec la compression génère de très bons résultats sur la réduction de la quantité d'informations tout en révélant des patterns de concepts éventuels. Cependant avec la trace que j'ai utilisé lors de mes tests j'ai noté que le mécanisme de composition permettait d'obtenir un meilleur résultat que le mécanisme de spécialisation – généralisation. A l'aide de l'implémentation des mécanismes d'abstraction j'ai obtenu une réduction du nombre d'appels d'environ 60% ce qui représente une diminution de plus de la moitié du nombre d'appels et une réduction de 30% du nombre de concepts présents par segment.

Il serait intéressant, dans de futures recherches de compléter l'outil en y intégrant la détection de patterns. En effet, il serait possible de simplifier d'avantage les différents segments en détectant une répétition successive de pattern pour en effectuer une compression, notamment comme je le fais actuellement pour les différents concepts. Nous pourrions recenser le nombre d'occurrences du pattern de concepts par segment, si la compression le permet nous pourrions représenter toute la trace avec le nombre d'occurrences des différents patterns. Cependant, il faudra effectuer une étude pour valider la pertinence de cette idée et voir si l'implémentation de celle-ci serait réellement avantageuse. J'ai constaté une certaine répétition d'éventuels patterns de concepts entre les segments observant ces derniers.

Pour conclure, l'outil que j'ai développé avec les différents mécanismes d'abstraction et de compression apporte une approche nouvelle qui j'espère, permettra, de faire avancer les recherches.

Bibliographie

BARON, Jonathan. ***Thinking and Deciding***. Cambridge : Cambridge. ISBN 9780511464874.

MASON, Richard. ***Understanding Understanding***. New-York : Suny. ISBN 0791458717.

SINGER, Jeremy., Kirkham Chris. – ***Dynamic analysis of Java program concepts for visualization and profiling.***

RILLING, J., MENG W.J., CHEN, F., CHARLAND, P. – ***Software Visualization – A Process Perspective.*** VISSOFT 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis.

DEELEN, P., VAN HAM, F., HUIZING, C., VAN DE WTERING, H. – ***Visualization of Dynamic Program Aspects.*** VISSOFT 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis.

KAPEC, Peter. – ***Visualizing software artifacts using hypergraphs***. SCCG 10 Proceedings of the 26th Spring Conference on Computer Graphics.

PEREZ, A., ABREU, R. ***– A Diagnosis-Based Approach to Software Comprehension***. ICPC 2014 Proceedings of the 22nd International Conference on Program.

XIE, X., POSHYVANYK D., MARCUS, A. – ***3D Visualization for Concept Location in Source Code***. ICSE '06 Proceedings of the 28th international conference on Software engineering.

LEHMAN, M. M., BELADY, L. A. – ***A model of large program development.*** IBM Syst. Journal 15(3) Septembre 1976.

DUGERDIL PH. - ***Using trace sampling techniques to identify dynamic clusters of classes***. IBM CAS SOFTWARE AND SYSTEMS ENGINEERING SYMPOSIUM (2007,Dublin). : proceedings of the 2007 conference of the center for advanced studies on Collaborative research. Hamilton auditorium of Dublin, 24 octobre 2007. 9 p.

WIKIMEDIA FOUNDATION. ***Wikipédia*** [en ligne]. https://fr.wikipedia.org/ (consulté 01.09.2015)

OBJECT REFINERY LIMITED. ***JFreeChart*** [en ligne]. http://www.jfree.org/jfreechart/ (consulté 01.09.2015)

RELO. ***Relo*** [en ligne]. http://relo.csail.mit.edu/index.html (consulté 01.09.2015)

FEAT. ***Feat*** [en ligne]. http://cs.mcgill.ca/~swevo/feat/ (consulté 01.09.2015)

THE CHISEL GROUP, UNIVERSITY OF VICTORIA. ***Chisel*** [en ligne] http://thechiselgroup.org/ (consulté 01.09.2015)