Université de Montréal

Utilisation de la visualisation interactive pour l'analyse des dépendances dans les logiciels

par
Simon Bouvier

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des arts et des sciences
en vue de l'obtention du grade de Maître ès sciences (M.Sc.)
en informatique

août, 2011

Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Utilisation de la visualisation interactive pour l'analyse des dépendances dans les logiciels

présenté par:

Simon Bouvier

a été évalué par un jury composé des personnes suivantes:

Pierre McKenzie,	président-rapporteur
Houari Sahraoui,	directeur de recherche
Pierre Poulin,	codirecteur
Philippe Langlais,	membre du jury

Mémoire accepté le: . . . . . .

RÉSUMÉ

La compréhension de la structure d'un logiciel est une première étape importante dans la résolution de tâches d'analyse et de maintenance sur celui-ci. En plus des liens définis par la hiérarchie, il existe un autre type de liens entre les éléments du logiciel que nous appelons liens d'adjacence. Une compréhension complète d'un logiciel doit donc tenir compte de tous ces types de liens. Les outils de visualisation sont en général efficaces pour aider un développeur dans sa compréhension d'un logiciel en lui présentant l'information sous forme claire et concise. Cependant, la visualisation simultanée des liens hiérarchiques et d'adjacence peut donner lieu à beaucoup d'encombrement visuel, rendant ainsi ces visualisations peu efficaces pour fournir de l'information utile sur ces liens. Nous proposons dans ce mémoire un outil de visualisation 3D qui permet de représenter à la fois la structure hiérarchique d'un logiciel et les liens d'adjacence existant entre ses éléments. Notre outil utilise trois types de placements différents pour représenter la hiérarchie. Chacun peut supporter l'affichage des liens d'adjacence de manière efficace. Pour représenter les liens d'adjacence, nous proposons une version 3D de la méthode des *Hierarchical Edge Bundles*. Nous utilisons également un algorithme métaheuristique pour améliorer le placement afin de réduire davantage l'encombrement visuel dans les liens d'adjacence. D'autre part, notre outil offre un ensemble de possibilités d'interaction permettant à un usager de naviguer à travers l'information offerte par notre visualisation. Nos contributions ont été évaluées avec succès sur des systèmes logiciels de grande taille.

Mots clés: visualisation du logiciel, visualisation de structure hiérarchique, visualisation de liens d'adjacence, algorithme de placement, possibilités d'interaction

ABSTRACT

Understanding the structure of software is an important first step in solving tasks of analysis and maintenance on it. However, in addition to the links defined by the hierarchy, there exists another type of links between elements of software that are called adjacency links. A complete understanding of software must take account of all these types of links. Visualization tools are generally effective in helping a developer in his understanding of software by presenting the information in a clear and concise manner. However, viewing these two types of links generate in general a lot of visual clutter, making these visualizations inefficient to provide useful information on these links. We propose in this M.Sc. thesis a 3D visualization tool that can represent both the hierarchical structure of an application and the adjacency relationships between its elements. Our tool uses three different types of layout to represent the hierarchy. Each layout can support efficiently the display of adjacency links. To represent adjacency links, we propose a 3D version of the Hierarchical Edge Bundles algorithm. We also use a metaheuristic algorithm to improve our layouts to further reduce visual clutter in the adjacency links. Moreover, our tool provides a set of interaction possibilities that allows a user to navigate through the information provided by our visualization. Our contributions have been evaluated successfully on large software systems.

Keywords: software visualisation, visualisation of hierarchical structure, visualisation of adjacency links, layout algorithm, interaction possibilities

TABLE DES MATIÈRES

LISTE DES FIGURES

à ma famille

REMERCIEMENTS

Je tiens d'abord à remercier chaleureusement mon directeur Houari Sahraoui ainsi que mon co-directeur Pierre Poulin pour l'aide extrêmement précieuse qu'ils m'ont apportée durant ma maîtrise. Le soutien de tous les instants qu'ils m'ont témoigné durant l'élaboration de mon projet ainsi que la grande patience dont ils font fait preuve à mon égard ont permis de rendre ce mémoire possible.

Je voudrais aussi remercier mon ami Guillaume Langelier pour tout le temps et l'aide qu'il m'a consacrés durant le développement de mon projet. Ses réponses et conseils éclairés m'ont permis de progresser plus rapidement dans mon travail, et l'amitié qu'il m'a témoignée durant ces dernières années a rendu mon séjour à ce laboratoire encore plus agréable.

Je remercie aussi mes deux frères, Daniel et Nicolas, pour le support moral qu'ils m'ont apporté au cours de ces dernières années, ainsi que mon amie Reika Arima pour ses nombreux encouragements.

Finalement, je tiens à remercier du fond du cœur mes parents, dont le support sans faille, autant sur le plan moral que financier, m'ont permis d'aller jusqu'au bout. Je voudrais les remercier d'avoir cru en moi, de m'avoir supporté à travers mes difficultés et, en fin de compte, d'avoir rendu tout cela possible.

CHAPITRE 1

INTRODUCTION

1.1 Contexte

La programmation par les objets est, de nos jours, le paradigme le plus utilisé pour le développement de nouveaux systèmes. Ce paradigme a commencé à prendre de l'ampleur il y a déjà près de trois décennies, et donc à l'exception de vieux logiciels hérités, comme par exemple ceux utilisés par certaines banques et compagnies d'assurances, la plupart des logiciels utilisés en entreprise suivent les règles de ce paradigme. Ceci implique que, que ce soit pour faire de la maintenance sur un logiciel utilisé par son entreprise ou pour développer une nouvelle application, un développeur risque fort d'avoir à comprendre la structure générale d'un logiciel qui a été développé en orienté objets.

La compréhension du logiciel est une première étape importante à réaliser avant de pouvoir effectuer des tâches de maintenance sur celui-ci. Ceci est une évidence en soi car, avant de pouvoir apporter un changement ou une correction, il faut tout d'abord déterminer les parties à modifier. De plus, dans un cas comme dans l'autre, il faut aussi pouvoir déterminer quels seront les impacts des modifications.

Une façon efficace de comprendre le logiciel est d'explorer visuellement les informations extraites de celui-ci. La visualisation est une façon efficace de représenter toutes sortes d'informations à celui qui l'utilise. Une de ses grandes forces est de pouvoir représenter sous des formes familières, souvent en utilisant des métaphores du monde réel, des données intangibles qui ne possèdent pas forcément une représentation naturelle. Les images produites ainsi permettent de tirer profit de la puissance du système visuel humain afin d'accélérer la compréhension.

Dans ce contexte, de nombreux outils de visualisation du logiciel ont été proposés ces dernières années. Ces outils ont grandement facilité la compréhension de logiciels de grande taille [16, 17]. Cependant, certains types d'information demeurent difficiles à représenter de manière efficace. C'est notamment le cas de la représentation conjointe

des liens hiérarchiques et des autres types de liens ; cette sorte de représentation fait l'objet de ce mémoire.

1.2 Problématique

Les logiciels à objets organisent leurs entités de façon hiérarchique, par exemple, la structure en paquetage en *Java*. Leur compréhension globale commence donc avant tout par la compréhension de la structure hiérarchique présente dans ceux-ci. Afin d'être efficaces, les outils d'aide à la compréhension et à l'évaluation de logiciels qui utilisent la visualisation doivent donc être en mesure d'afficher clairement et sans ambiguïté l'organisation hiérarchique des entités d'un logiciel. Ceci permet ainsi à un utilisateur non seulement d'extraire rapidement une vue d'ensemble des relations hiérarchiques entre les entités mais aussi de mettre en contexte le rôle d'une entité en particulier par rapport au reste du logiciel.

En plus des liens hiérarchiques, il existe toute une gamme de liens entre les entités logicielles. Ces liens, que nous appelons d'adjacence, sont indépendants de la hiérarchie et représentent par exemple les invocations entre méthodes. Bien que la compréhension de la structure d'un logiciel soit un bon début, elle n'est pas suffisante pour effectuer bon nombre de tâches de maintenance. Comprendre comment les parties du logiciel sont inter-reliées et comment elles communiquent entre elles est tout aussi important.

Jusqu'à présent, peu de méthodes permettent de représenter ces deux types de liens simultanément, et encore moins de façon claire lorsqu'utilisées pour visualiser de grands logiciels. Récemment, une technique a été développée pour visualiser des liens d'adjacence dans une structure hiérarchique. Celle-ci, nommée *Hierarchical Edge Bundles* [10], s'est avérée efficace et a été utilisée depuis dans plusieurs outils. Par contre, son adaptation au logiciel demeure approximative en particulier quand les nombreuses propriétés des entités doivent être également représentées.

La problématique que nous cherchons à résoudre dans ce mémoire est donc de pouvoir visualiser, de façon claire et efficace, à la fois la structure hiérarchique d'un logiciel, les propriétés de ses entités et les liens d'adjacence qui existent entre ces dernières. En

partant de la technique des *Hierarchical Edge Bundles* (*HEB*), la visualisation que nous allons proposer doit, d'une part, définir des représentations efficaces pour la structure et les liens, et, d'autre part, faire en sorte que les deux représentations soient compatibles entre elles. Il est aussi important que notre visualisation fonctionne avec des logiciels de grande taille. Finalement, notre visualisation doit offrir diverses méthodes d'interaction permettant à l'utilisateur de naviguer à travers les données qui lui sont présentées.

1.3 Contributions

Nous avons développé un système offrant une visualisation 3D permettant d'afficher la structure hiérarchique d'un logiciel selon divers placements, ainsi que d'afficher les liens d'adjacence entre les éléments de celui-ci.

Au lieu de créer notre système de toutes pièces, nous avons plutôt utilisé l'environnement VERSO [16, 17] comme point de départ. Cet environnement offre déjà un environnement 3D interactif bien construit. Un des avantages principaux est que notre visualisation profite de toutes les fonctionnalités qu'offrait déjà VERSO, non seulement au niveau de la navigation 3D mais surtout au niveau de la représentation des propriétés des entités.

Afin d'offrir une visualisation qui résout la problématique posée à la section précédente, nous avons ajouté diverses fonctionnalités à VERSO ainsi que modifié plusieurs aspects de cet environnement qui étaient déjà en place. Plus spécifiquement, nous avons apporté les contributions décrites dans les prochains paragraphes.

Premièrement, nous avons développé trois types de placements servant à visualiser une hiérarchie. Ceux-ci non seulement passent à l'échelle mais sont construits de façon à être bien adaptés à l'ajout de liens d'adjacence dans la visualisation, et à diminuer l'encombrement visuel causé par l'affichage simultané de ces deux types de liens. Nous avons aussi ajouté un algorithme métaheuristique qui améliore chacun de ces types de placements.

Nous avons intégré à VERSO une représentation 3D des liens d'adjacence en utilisant la méthode *HEB*. Nous avons donc intégré cette méthode dans un environnement

3D en modifiant l'algorithme afin de mieux distinguer les paquets de liens entrant et sortant d'un groupe d'éléments. Nous avons utilisé la couleur des liens afin d'en indiquer la direction, en s'assurant de faire ressortir clairement le point de départ et d'arrivée de chacun, et utilisé la taille des liens afin de les quantifier. Nous avons ajouté diverses méthodes permettant à l'utilisateur d'interagir avec la visualisation afin de naviguer facilement dans la masse d'informations qui lui est présentée, et ainsi d'en faire ressortir les informations qui l'intéressent.

Finalement, nous avons fait une évaluation de notre outil en l'utilisant sur quelques logiciels réels de grande taille. Cette évaluation montre la façon dont nous pouvons utiliser notre outil pour analyser l'information qui nous est présentée, et donne des exemples de découvertes que nous avons faites grâce à celui-ci.

1.4 Structure du mémoire

Le présent mémoire est structuré comme suit. Le chapitre 2 passe en revue plusieurs travaux traitant de la visualisation de liens hiérarchiques, de liens d'adjacence ou de l'affichage des deux en simultané. Le chapitre 3 décrit les trois types de placements que nous avons développés pour la visualisation de structures hiérarchiques. Le chapitre 4 décrit le procédé avec lequel nous créons et affichons les liens d'adjacence dans la visualisation ainsi que les méthodes interactives qu'offre celle-ci pour filtrer l'information affichée à l'écran. Le chapitre 5 fait une évaluation de notre système en le testant sur des logiciels de grande taille. Finalement, une conclusion et des perspectives d'amélioration sont données dans le chapitre 6.

CHAPITRE 2

TRAVAUX CONNEXES

Comme nous venons de le mentionner dans le chapitre précédent, notre objectif est d'aider un utilisateur à effectuer des tâches d'analyse et de maintenance sur un logiciel en lui offrant une visualisation interactive de celui-ci. Afin que cette visualisation puisse fournir des informations utiles à l'accomplissement d'une bonne partie des tâches qu'un utilisateur voudrait effectuer sur un logiciel, celle-ci doit afficher de façon concurrente la structure hiérarchique organisant les éléments du logiciel visualisé ainsi que les liens d'adjacence existant entre ceux-ci. La première étape dans l'élaboration de cette visualisation est de déterminer de quelle façon nous voulons représenter la structure hiérarchique d'un logiciel ainsi que les liens d'adjacence existant entre ses éléments. Les représentations utilisées pour afficher ces deux types de liens doivent non seulement être efficaces lorsqu'utilisées séparément, mais elles doivent aussi être compatibles entre elles afin qu'on puisse les utiliser simultanément dans une seule et même visualisation qui soit claire et compréhensible. Plusieurs méthodes ont été développées jusqu'à maintenant pour afficher chacun de ces types de liens, ainsi que pour les afficher de façon simultanée. Nous allons donc passer en revue dans ce chapitre quelques travaux plus ou moins récents portant sur la représentation de ces différents types de liens, en comparant leurs forces et leurs faiblesses. La section 2.1 donne un résumé des trois façons générales les plus répandues pour afficher des liens entre des éléments. La section 2.2 passe en revue quelques travaux portant sur la façon de représenter une structure hiérarchique. La section 2.3 décrit certains travaux portant sur la façon de représenter des liens d'adjacence entre des éléments, sans que ceux-ci soient nécessairement organisés de façon hiérarchique. La section 2.4 décrit quelques travaux portant sur la représentation simultanée d'une structure hiérarchique et des liens d'adjacence qui existent entre les éléments de cette structure. Finalement, la section 2.5 explique rapidement en quoi consiste le logiciel VERSO, sur lequel s'appuie notre travail, ainsi que ses principales fonctionnalités.

2.1 Principales catégories de représentation de liens

La plupart des méthodes développées jusqu'à aujourd'hui pour afficher un groupe d'éléments et les liens existant entre eux, que ce soit des liens hiérarchiques ou d'adjacence, peuvent être réparties dans deux catégories de représentation des liens : les représentations nœuds-liens et les représentations par inclusion. Une autre catégorie de représentation des liens, moins répandue, est la représentation matricielle. Les trois sections suivantes vont chacune décrire une de ces catégories.

2.1.1 Représentation nœuds-liens

La façon la plus simple et naturelle que l'on peut imaginer pour visualiser les liens existant entre des éléments est de représenter les éléments par un symbole quelconque (un point, un cercle, un carré, etc.) et de représenter le lien qui existe entre deux éléments de façon explicite en traçant un trait (une ligne, une ligne polygonale, une courbe, etc.) entre les symboles les représentant. Les traits peuvent utiliser des symboles, par exemple une pointe de flèche, ou un de leurs attributs visuels, par exemple leur couleur ou leur taille, pour afficher le sens du lien, si celui-ci est dirigé. La figure 2.1 montre un exemple simple d'un graphe dirigé, visualisé en utilisant une représentation nœuds-liens.

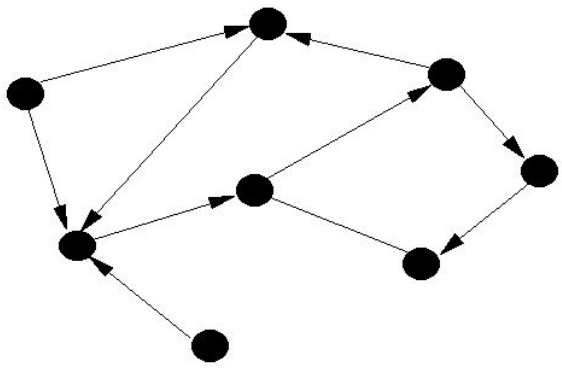


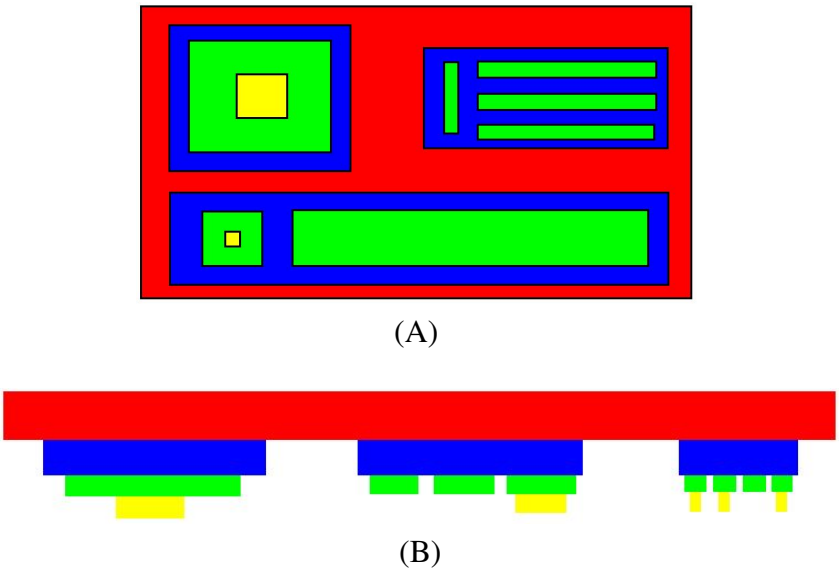
Figure 2.1 – Exemple d'une représentation nœuds-liens. Cette figure présente un graphe dirigé en utilisant une représentation nœuds-liens. Le sens des liens est indiqué en utilisant une pointe de flèche.

2.1.2 Représentation par inclusion

L'idée de base de la représentation par inclusion est de représenter de façon implicite la relation parent-enfant entre deux entités par l'inclusion de la représentation graphique de l'enfant dans la représentation graphique du parent. Le terme *inclusion* utilisé ici fait référence, de façon générale, aux contraintes visuelles, principalement la taille et la position, que la représentation graphique du parent impose à la représentation graphique de ses enfants. La plupart des représentations par inclusion indiquent les relations parent-enfant par le positionnement relatif des entités entre elles, ainsi que par les contraintes de taille que le parent impose à ses enfants. La figure 2.3 présente deux exemples de représentation par inclusion, chacune utilisant une manière différente d'inclure visuellement un élément dans un autre. Il est important de noter que les représentations par inclusion ne sont applicables en pratique qu'à des éléments organisés de façon hiérarchique, et ne peuvent donc pas être utilisées pour afficher des liens d'adjacence. En effet, ceux-ci ne sont pas aussi restreints que les liens hiérarchiques, ce qui fait que n'importe quel élément peut être lié avec n'importe quel autre, nous empêchant d'utiliser leurs positions relatives pour représenter les liens qui existent entre eux. Par exemple, il serait en pratique impossible de placer côte à côte, dans la visualisation, les éléments liés par des liens d'adjacence, l'espace devenant rapidement saturé autour de ceux ayant des liens avec beaucoup d'éléments différents. Un exemple plus flagrant est lorsque nous avons un cycle dans les liens et que nous utilisons une représentation par imbrication, comme le montre la figure 2.3.

2.1.3 Représentation matricielle

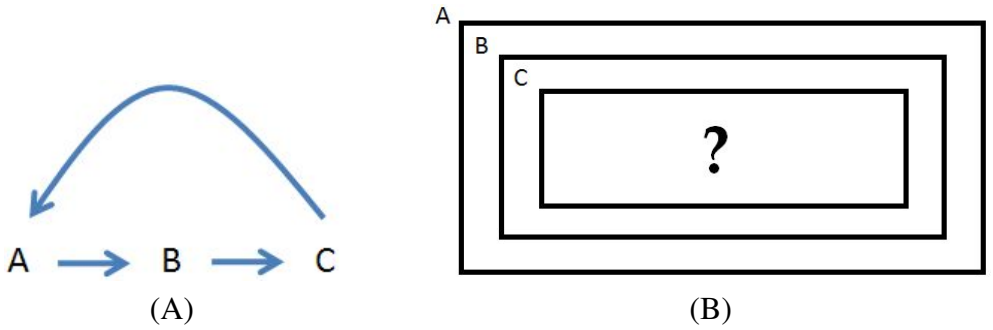
La représentation matricielle consiste essentiellement à représenter les liens existant entre les éléments à l'aide d'une matrice, c'est-à-dire un tableau en deux dimensions. Nous associons un élément à chaque rangée et à chaque colonne, puis nous indiquons un lien entre deux éléments en remplissant la cellule située à l'intersection de la rangée du premier et de la colonne du second, et vice-versa si le lien est bidirectionnel ou s'il n'est pas dirigé. Pour voir les liens existant entre les éléments d'un même groupe, il suffit



(A)

(B)

Figure 2.2 – Exemples de représentations par inclusion. L'image A) montre une représentation hiérarchique où les éléments sont représentés par des rectangles, et les relations parent-enfant sont indiquées par l'imbrication de ces rectangles. Afin de mieux faire ressortir ces imbrications, nous utilisons une couleur différente pour chaque niveau de la hiérarchie. Cette forme d'inclusion est, pour la plupart des gens, la plus naturelle qu'il puisse y avoir. L'image B) montre une autre hiérarchie, en utilisant une forme différente d'inclusion pour indiquer les relations parent-enfant. Les éléments de la hiérarchie sont représentés par des barres horizontales de longueur décroissante à mesure que nous descendons plus profondément dans celle-ci. La relation parent-enfant est indiquée par le placement d'une barre en-dessous de celle qui représente son parent : la relation est très facile à voir, même si l'enfant n'est pas visuellement à l'intérieur de son parent, comme c'est le cas dans l'image A). Ceci montre que la notion d'inclusion ne se limite pas à un élément qui est visuellement englobé par un autre.



(A) (B)

Figure 2.3 – Exemple d'imbrication impossible. L'image A) montre trois éléments liés de façon dirigée, avec un cycle dans les liens. L'image B) montre l'imbrication partielle que nous pouvons faire pour représenter les liens existant entre ces éléments. Il nous est impossible de la compléter, car pour y arriver le rectangle représentant l'élément *C* doit englober celui représentant l'élément *A*, ce qui est évidemment impossible, car ce dernier l'englobe déjà.

d'associer, dans le même ordre, les éléments de ce groupe aux rangées et aux colonnes. La figure 2.4 donne un exemple d'une représentation matricielle.

2.2 Visualisation d'une structure hiérarchique

Dans cette section, nous allons présenter divers travaux proposant des méthodes servant à visualiser une structure hiérarchique. La section 2.2.1 décrit quelques méthodes de type nœuds-liens, tandis que la section 2.2.2 décrit des méthodes de représentation hiérarchique par inclusion. En ce qui concerne la représentation matricielle, à notre connaissance, il n'existe pas de méthode, ou du moins de méthode efficace, pour représenter une hiérarchie à l'aide d'une matrice.

2.2.1 Visualisation de la hiérarchie en utilisant une représentation nœuds-liens

La hiérarchie d'un logiciel correspond à un graphe hiérarchique, qui est un cas particulier de graphe orienté. La relation parent-enfant entre les éléments oriente les liens et apporte des contraintes supplémentaires qui permettent de réorganiser de façon systématique l'emplacement des éléments afin de clarifier la visualisation des liens existant entre eux. Trois des méthodes les plus connues pour faire cela sont : l'arborescence,

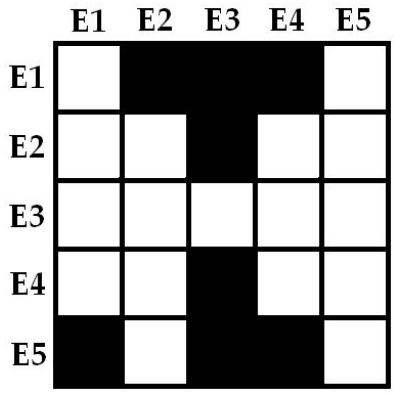


Figure 2.4 – Exemple d'une représentation matricielle. La matrice dans cette image montre les liens dirigés existant entre les éléments *E1* à *E5*. Les éléments ne pouvant être liés avec eux-mêmes, la diagonale principale est nécessairement vide. Nous pouvons remarquer que la rangée de l'élément *E3* est vide, tandis que sa colonne est pleine, sauf bien sûr la cellule située sur la diagonale principale. Ceci indique que cet élément n'appelle personne, mais est appelé par tout le monde.

l'arbre radial et l'arbre de ballons [8]. La figure 2.5 présente un graphe hiérarchique où les éléments ont été positionnés de façon arbitraire, sans suivre de règle de placement précise : la hiérarchie est difficile à saisir dans cette visualisation. Les figures 2.6 à 2.8 montrent le même graphe, mais en utilisant une des trois méthodes susmentionnées pour repositionner ses éléments avant de l'afficher.

Une méthode de représentation nœuds-liens, un peu plus sophistiquée pour afficher une hiérarchie, est la méthode des *Cone Trees* [20], qui offre une visualisation 3D interactive de la hiérarchie. Dans cette visualisation, chaque élément de la hiérarchie qui a des enfants est placé au sommet d'un cône semi-transparent, alors que ses enfants sont placés à la base de ce cône, en étant espacés de façon égale. Les cônes servent aussi à espacer, dans la visualisation, les différents niveaux de la hiérarchie, et ils ont tous la même hauteur afin que ces niveaux soient équidistants. La visualisation permet d'afficher la hiérarchie à l'horizontale ou à la verticale, et celle-ci procure à l'utilisateur les fonctionnalités nécessaires pour naviguer librement dans son espace 3D. Ces fonctionnalités de navigation sont indispensables pour permettre à l'utilisateur de contourner les problèmes liés à l'occultation des éléments, problèmes qui sont inhérents aux visualisations en trois dimensions. Finalement, l'utilisateur peut interagir avec la hiérarchie de

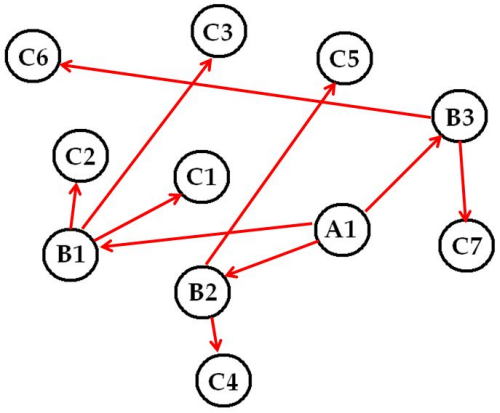


Figure 2.5 – Exemple d'un graphe hiérarchique. Les éléments de ce graphe ont été positionnés de façon arbitraire. La hiérarchie globale est difficile à saisir au premier coup d'œil.

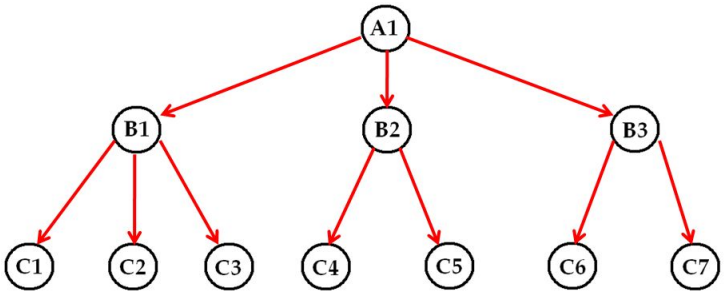


Figure 2.6 – Exemple d'une arborescence. On commence par placer la racine dans le haut de la visualisation, puis on place ses enfants directement en-dessous d'elle, et ainsi de suite de façon récursive pour chacun de ses enfants. Ceci donne des couches successives d'éléments, espacées également, où chaque couche représente un niveau de la hiérarchie. On peut faire varier le placement en plaçant les éléments de bas en haut, de gauche à droite ou de droite à gauche.

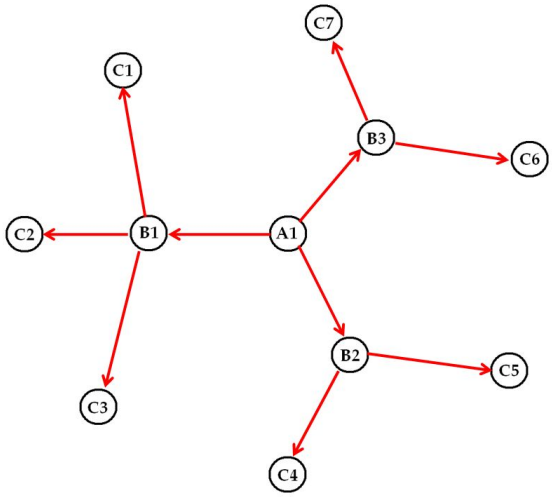


Figure 2.7 – Exemple d'un arbre radial. La racine est située au centre du placement. Ses enfants sont ensuite placés sur un cercle entourant la racine, espacés de façon équitable selon leur nombre de descendants, et ainsi de suite de façon récursive pour leurs enfants. Les éléments sont donc répartis sur une série de cercles concentriques, où chacun de ces cercles représente un niveau de la hiérarchie.

diverses façons, notamment en ouvrant ou fermant des branches de celle-ci, en sélectionnant un élément afin de le mettre en surbrillance ou en déplaçant un élément, ainsi que la sous-structure qui en découle, afin de les repositionner ailleurs dans la visualisation. La figure 2.9 montre deux exemples de cette visualisation.

Le plus grand défaut des représentations nœuds-liens que nous venons de montrer, ainsi que des représentations nœuds-liens en général, est leur faible extensibilité, défaut qui est principalement dû à l'usage inefficace qu'elles font de leur espace d'affichage. En effet, celles-ci ne fonctionnent très bien que pour visualiser de petites hiérarchies, et ne sont donc pas adaptées pour des hiérarchies de moyenne ou grande taille, par exemple de 1 000 éléments ou plus.

2.2.2 Visualisation de la hiérarchie avec une représentation par inclusion

2.2.2.1 Visualisations en 2D

Un des exemples les plus classiques, et aussi des plus utilisés, de représentation d'une structure hiérarchique par inclusion est la visualisation *Treemap* de Schneider-

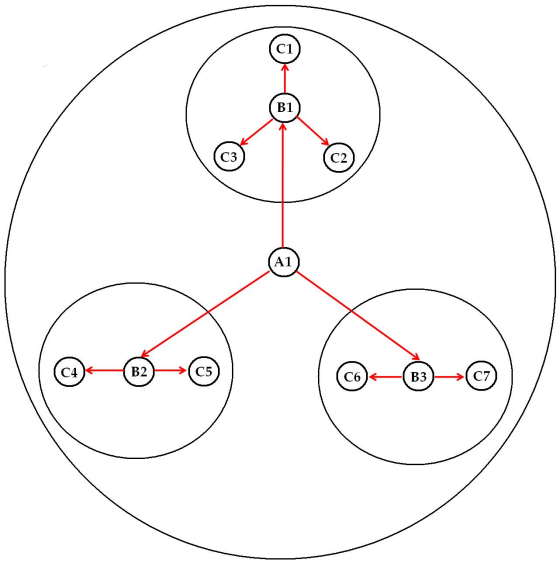
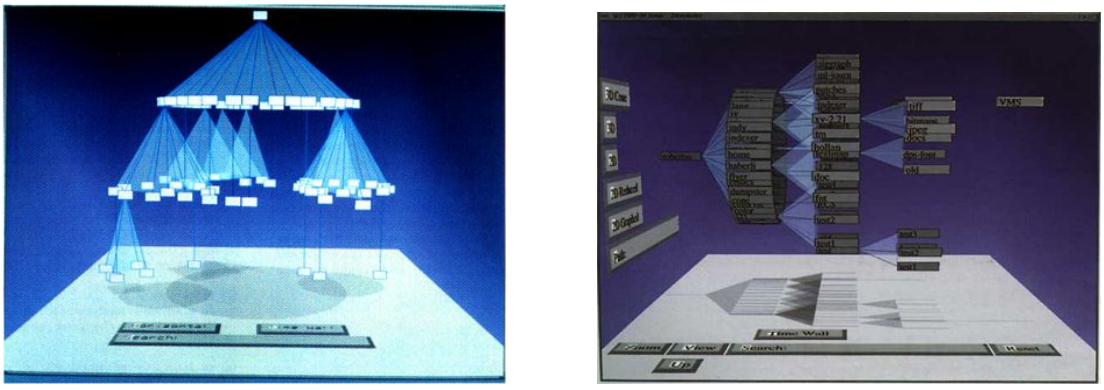


Figure 2.8 – Exemple d'un arbre de ballons (*balloon tree* en anglais). La racine est située au centre du placement avec ses enfants placés autour d'elle, un peu à la manière de l'arbre radial, mais à la différence que cette méthode trace un cercle qui englobe la racine ainsi que tous ses descendants. Ce processus est répété de façon récursive pour les enfants de la racine. La taille d'un cercle associé à un élément quelconque est donc dépendante du nombre de descendants qu'il possède. S'il n'a aucun descendant, alors on ne trace pas de cercle autour de lui. Chaque cercle représente un niveau de la hiérarchie, et leur imbrication correspond au concept de la représentation par inclusion : nous pouvons donc considérer cette visualisation comme une sorte d'hybride entre une représentation nœuds-liens et une représentation par inclusion.



(A) (B)

Figure 2.9 – Exemples de Cone Trees. Cette figure est tirée de [20].

man et al. [13, 21]. Cette visualisation consiste en un ensemble de rectangles, où chacun représente un élément de la hiérarchie et est dessiné dans les limites de son rectangle parent. Les relations parent-enfant de la hiérarchie sont donc représentées par l'imbrication d'un rectangle dans un autre, ce qui permet d'utiliser efficacement l'espace d'affichage alloué. D'ailleurs, l'utilisation efficace de l'espace d'affichage était un des critères les plus importants aux yeux des auteurs de cette méthode lorsqu'ils l'ont développée. Pour atteindre cet objectif, l'affichage d'un ensemble de rectangles représentant des éléments frères de la hiérarchie est réalisé en partitionnant leur rectangle parent, permettant ainsi de se servir de façon utile de la totalité de l'espace alloué. La visualisation *Treemap* a joui d'une grande popularité depuis sa création, ce qui a donné lieu à certaines améliorations de la visualisation *Treemap* standard [2, 27]. Aussi, certaines visualisations, par exemple la méthode des *Beam Trees* [26], ont pour fondement l'algorithme original du *Treemap*. La méthode des *Beam Trees* utilise les rectangles, qui représentent les éléments de la hiérarchie, obtenus avec l'algorithme *Treemap*, puis modifie leurs dimensions et les affiche en commençant par les éléments les plus profonds, en remontant jusqu'à la racine. La perception de profondeur causée par l'occultation des éléments, occultation créée de façon volontaire par cette méthode, permet de mieux faire ressortir la structure de la hiérarchie visualisée. Cette méthode permet aussi à l'utilisateur de visualiser les résultats en 2D ou en 3D. Une des représentations hiérarchiques utilisées par notre système est aussi dérivée du *Treemap* par imbrication : plus de détails sur la visualisation *Treemap* seront donc donnés dans le chapitre 3.

Une autre représentation hiérarchique par inclusion bien connue est la visualisation *Sunburst* de Stasko et al. [22, 23]. Cette visualisation représente les niveaux de la hiérarchie par des anneaux concentriques de même épaisseur. La racine, représentée par un cercle, est placée au centre de la visualisation, puis les niveaux subséquents sont placés autour d'elle, de l'intérieur vers l'extérieur. Pour représenter les éléments de la hiérarchie, l'algorithme commence par diviser le premier anneau suivant la racine en autant de segments qu'il y a d'éléments à ce niveau. L'angle attribué à chaque segment est proportionnel au rapport existant entre la taille de l'élément représenté par le segment et la taille totale des éléments situés à ce niveau. Il est à noter que, dans ce cas-ci, la taille d'un

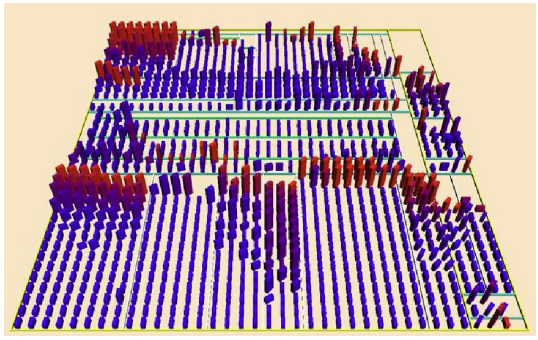
élément est calculée en prenant en considération tous ses descendants. Ensuite, l'algorithme répète le même procédé pour représenter les enfants de ces éléments, mais cette fois-ci en divisant la portion de l'anneau subséquent délimité par l'angle qu'occupe le parent sur son anneau. Cela permet de placer un groupe d'enfants dans les limites du segment représentant leur parent, ce qui correspond à la forme d'inclusion utilisée dans cette visualisation. Ce procédé est ensuite répété de façon récursive sur tous les éléments de la hiérarchie qui ont des enfants.

Pour trouver une utilisation des visualisations *Treemap* et *Sunburst* dans le domaine du génie logiciel, il ne faut pas chercher plus loin que le logiciel VERSO [16, 17], logiciel sur lequel s'appuie notre système. Ce logiciel utilise ces deux visualisations afin d'afficher la structure hiérarchique de logiciels à objets, principalement de logiciels écrits en Java, où les éléments internes de la hiérarchie correspondent à des paquetages et les feuilles à des classes. L'élément de base de la hiérarchie étant la classe, l'auteur de ces articles a dû modifier les visualisations *Treemap* et *Sunburst* standards afin qu'elles travaillent avec des valeurs discrètes pour calculer la taille des éléments. Ceci a principalement comme conséquence de créer des trous dans la visualisation et, dans le cas du *Sunburst*, que les segments d'anneaux n'aient pas tous la même épaisseur. La figure 2.10 montre un exemple des visualisations *Treemap* et *Sunburst* standards, comparées avec celles implémentées dans VERSO.

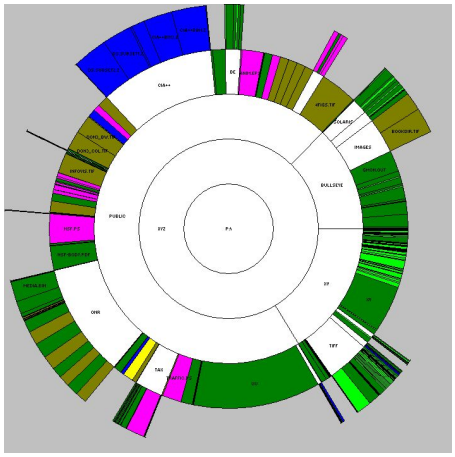
Ces deux visualisations sont qualifiées de « visualisation par remplissage d'espace » (*space-filling* visualisation en anglais). Ceci veut dire essentiellement que ces visualisations cherchent à utiliser au maximum l'espace alloué, c'est-à-dire d'en gaspiller le moins possible par de l'espace vide. Le plus grand avantage de ces visualisations est donc leur très grande extensibilité, surtout comparée aux visualisations nœuds-liens montrées à la section 2.2.1. En contrepartie, une faiblesse de ces visualisations est que, comme les liens hiérarchiques ne sont pas représentés explicitement, la structure de la hiérarchie est plus difficile à saisir que dans une représentation nœuds-liens. Par exemple, il est plus difficile de déterminer à quel niveau se situe un élément donné dans un *Treemap* que dans une arborescence.

La visualisation par imbrication de cercles, que nous appellerons *Circle Packing* [29],

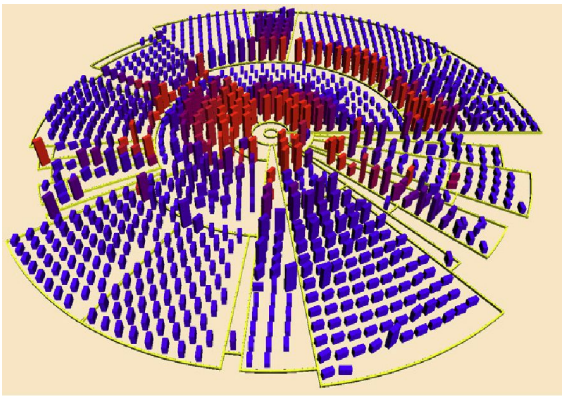
(A)



(B)



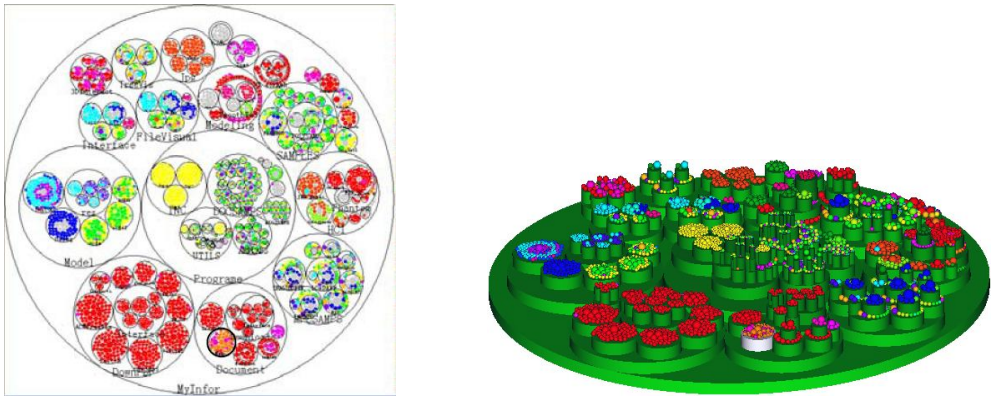
(C)



(D)

Figure 2.10 – Exemples de visualisations *Treemap* et *Sunburst*. L'image A), tirée de [21], montre la visualisation *Treemap* standard développée par l'auteur de cet article. L'image B), tirée de [17], montre l'implémentation de la visualisation *Treemap* dans VERSO. L'image C), tirée de [23], montre la visualisation *Sunburst* standard développée par l'auteur de cet article. L'image D), tirée de [17], montre l'implémentation de la visualisation *Sunburst* dans VERSO.

s'est inspirée de la visualisation *Treemap*, mais en cherchant à rendre plus facilement visible la structure de la hiérarchie comparée à celle-ci. Pour y arriver, la visualisation *Circle Packing* utilise des cercles, plutôt que des rectangles, pour représenter les éléments de la hiérarchie, où la taille du cercle représente la taille de l'élément. La structure de la hiérarchie est représentée en plaçant les cercles représentant des éléments frères dans celui représentant leur parent. Les cercles représentant des éléments frères sont placés de façon à former un amas aussi compact que possible. Pour ce faire, ceux-ci sont placés, un à la fois, à la périphérie de l'amas construit jusque-là, de façon à être tangents aux cercles voisins. Les cercles sont aussi placés de façon à ce que la forme globale de l'amas soit circulaire autant que possible. Comme les éléments sont représentés par des cercles, cela permet de les coller les uns sur les autres tout en laissant un certain espace entre eux afin de mieux les discerner, contrairement aux rectangles utilisés dans le *Treemap*. Cette visualisation permet aussi d'afficher les données en 3D, où la visualisation des relations parent-enfant est renforcée en empilant les éléments enfants sur leur parent, ce dont nous nous sommes inspirés pour la visualisation *Treemap* utilisée dans notre système, qui est présentée à la section 3.2. La figure 2.11 montre un exemple de cette visualisation, autant en 2D qu'en 3D.



(A) (B)

Figure 2.11 – Exemple de la visualisation *Circle Packing*. L'image A) montre la visualisation *Circle Packing* d'un logiciel en utilisant la vue 2D. L'image B) montre la visualisation *Circle Packing* du même logiciel, mais en utilisant la vue 3D. Ces figures sont tirées de [29].

2.2.2.2 Visualisations en 3D

Les visualisations que nous avons montrées à la section précédente ont été conçues a priori pour être affichées dans un espace en deux dimensions. De plus, même lorsque nous les intégrons à un monde en trois dimensions, par exemple les visualisations *Treemap* et *Sunburst* implémentées dans VERSO, celles-ci restent planaires, c'est-à-dire que les éléments sont tous placés sur le même plan, ce qui en fin de compte équivaut à leur version 2D. Il est vrai que les visualisations *Beam Trees* et *Circle Packing* offrent une vue 3D de la hiérarchie, mais leur utilisation de la troisième dimension reste quand même assez limitée. Par contre, il existe d'autres visualisations qui ont été conçues dès le départ pour être affichées dans un espace en trois dimensions, et donc qui utilisent au maximum la troisième dimension pour positionner les éléments.

Un premier exemple de visualisation d'une hiérarchie en 3D est le « Cube d'information » de Rekimoto et al. [19]. Cette visualisation représente les éléments de la hiérarchie avec des cubes semi-transparents, en écrivant sur eux le nom de l'élément représenté par chacun d'eux, et indique les relations parent-enfant en emboîtant ces cubes les uns dans les autres. La figure 2.12 montre un exemple de cette visualisation.

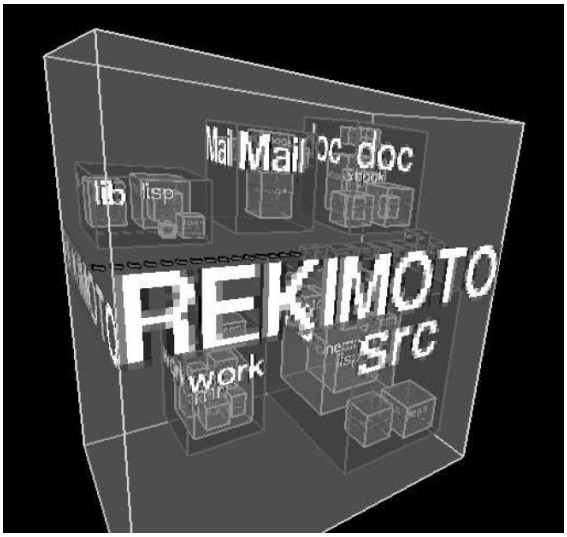


Figure 2.12 – Exemple de la visualisation « Cube d'information ». Cette figure est tirée de [19].

Un autre exemple est la visualisation « Botanique » [15], qui s'inspire du modèle d'un

arbre naturel pour représenter la hiérarchie. Dans cette visualisation, chaque branche de l'arbre représente un élément interne de la hiérarchie, et chaque branche, divergente de celle-ci, représente un de ses enfants. Le tronc de l'arbre correspond à la racine de la hiérarchie, et l'algorithme ajoute les branches divergentes récursivement à partir de celui-ci. Chaque branche a aussi à son extrémité un « fruit », qui est en fait une sphère sur laquelle sont placés des icônes représentant ses enfants qui sont des feuilles. La figure 2.13 montre un exemple de cette visualisation.

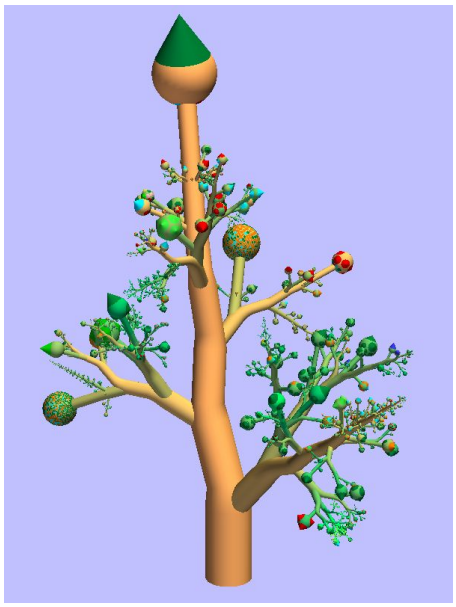


Figure 2.13 – Exemple d'une visualisation « Botanique ». Cette figure est tirée de [15].

Les visualisations 3D que nous venons de montrer ont été conçues au départ pour afficher les éléments d'un système de fichiers. Par contre, la structure d'un tel système est identique, entre autres, à celle d'un logiciel écrit en *Java* : ces visualisations pourraient donc être utilisées, telles quelles, pour afficher la structure hiérarchique d'un logiciel.

2.3 Visualisation de liens d'adjacence

Comme mentionné dans le chapitre précédent, en plus des liens hiérarchiques, il peut exister des liens d'adjacence entre les éléments. Ces liens ne sont pas aussi contraints que les liens hiérarchiques, et peuvent ainsi unir n'importe quelle paire d'éléments, indépen-

damment des liens hiérarchiques pouvant exister entre eux. Nous allons donc présenter dans cette section quelques travaux permettant d'afficher des liens d'adjacence entre des éléments, et ce peu importe que ceux-ci soient organisés de façon hiérarchique ou non. La section 2.3.1 décrit quelques visualisations utilisant une représentation nœuds-liens pour afficher les liens d'adjacence, tandis que la section 2.3.2 décrit une visualisation qui utilise une représentation matricielle. Pour ce qui est des représentations par inclusion, comme mentionné à la section 2.1.2, celles-ci ne peuvent être utilisées pour représenter des liens d'adjacence, donc évidemment il n'y a pas de visualisation dans cette section qui les utilise.

2.3.1 Visualisation de liens d'adjacence en utilisant une représentation nœuds-liens

La façon la plus simple d'afficher des liens d'adjacence est de tracer une ligne droite entre chaque paire d'éléments liés. Cette visualisation fonctionne bien pour de petits groupes d'éléments, mais dès que le nombre d'éléments et de liens augmente, par exemple à quelques centaines ou plus, il devient extrêmement difficile de discerner quoi que ce soit dans celle-ci. La figure 2.14 montre un exemple de cette difficulté. Les techniques d'agrégation d'arcs forment une catégorie de visualisation de liens qui s'est développée au cours des dernières années et qui a gagné rapidement en popularité. L'idée générale de ces visualisations est de fusionner les liens en paquets selon certains critères tels leur direction, les éléments qu'ils unissent ou leur proximité dans l'espace d'affichage, afin de réduire la complexité de la visualisation et de faire ressortir les tendances générales dans l'ensemble des liens. Ces visualisations sont très extensibles, pouvant afficher des centaines d'éléments avec des milliers de liens tout en offrant de l'information pertinente à la personne qui les utilise.

Un premier exemple de cette catégorie de visualisation est le *Flow Map Layout* [18]. Cette visualisation s'est inspirée des cartes de flux qu'utilisent depuis longtemps les cartographes pour indiquer le transit d'objets ou de personnes entre une source et différents lieux. L'idée principale de ces cartes, et du même coup de cette visualisation, est de fusionner les liens qui ont une destination commune en les représentant avec un seul lien

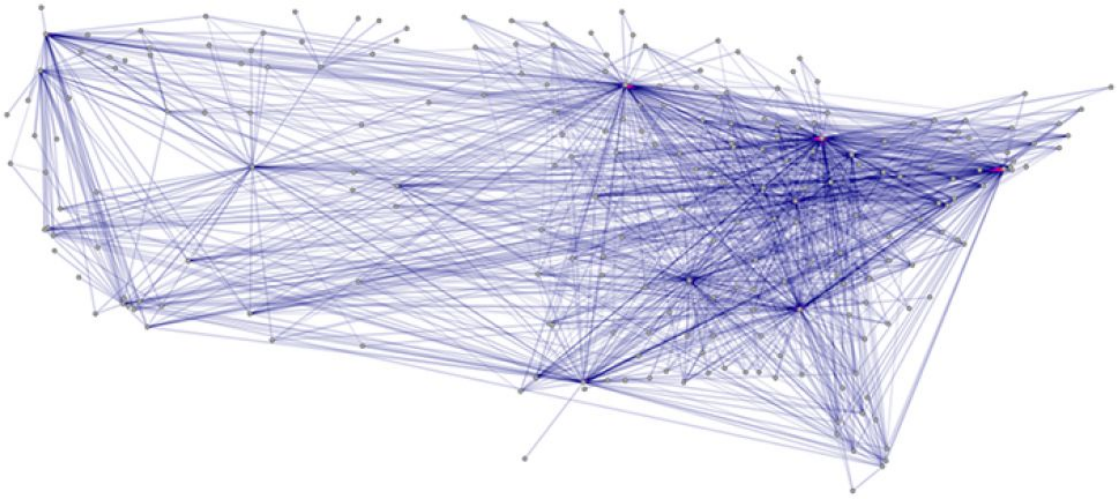


Figure 2.14 – Exemple de visualisation avec des lignes droites. Cette figure est tirée de [11].

plus épais, puis de les séparer en deux liens distincts plus minces quand leur chemin diverge. Afin de produire des cartes de flux, cette visualisation travaille à partir d'un ensemble d'éléments ayant une position précise dans le graphe et de l'ensemble de liens existant entre ceux-ci. La figure 2.15 montre un exemple de cette visualisation.

Cette visualisation permet de réduire beaucoup l'encombrement visuel dans les liens d'adjacence, produisant ainsi des résultats clairs et qui permettent d'obtenir de l'information utile. Par contre, celle-ci est très limitée, étant donné qu'elle ne permet de voir que les liens émanant d'une source précise. Étant donné que notre objectif est de pouvoir visualiser tous les liens d'adjacence à la fois, afin de voir les tendances dans ceux-ci, cette visualisation n'est pas adaptée pour répondre à nos besoins.

Un autre exemple de visualisation par agrégation d'arcs est la méthode *Force-Directed Edge Bundling* [11], ou *FDEB*. Cette méthode utilise un graphe où les éléments sont liés à l'aide de lignes droites, comme dans la figure 2.14, puis applique un algorithme qui regroupe les liens en paquets selon la force qu'ils exercent les uns sur les autres. En résumé, l'algorithme place sur chaque lien un certain nombre de points de séparation pi, qui est le même pour chaque lien, ce qui permet de séparer les liens en plusieurs segments afin de pouvoir les déformer. L'attraction qui existe entre deux liens donnés est calculée séparément pour chaque paire de points de séparation pi correspondant sur

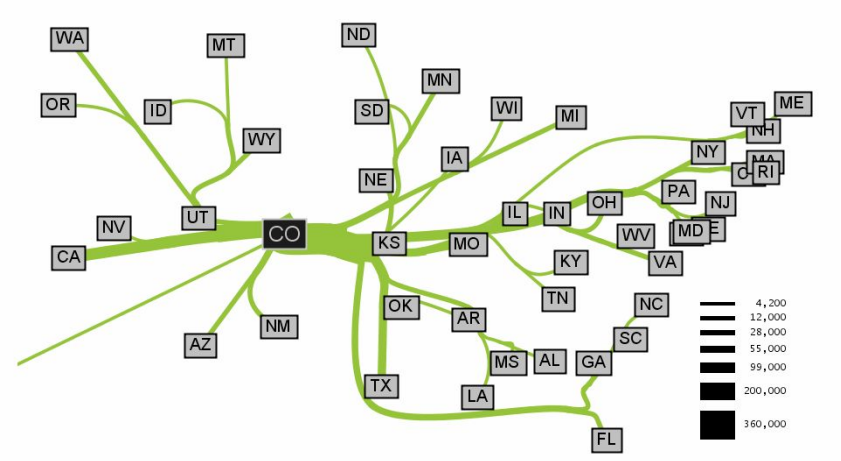


Figure 2.15 – Exemple de la visualisation Flow Map Layout. Cet exemple montre l'émigration en provenance du Colorado vers les autres états des États-Unis, de 1995 à 2000. Cette figure est tirée de [18].

ces deux liens. Chaque point de séparation est déplacé selon la force qu'exercent sur lui les points qui sont ses voisins directs sur son lien, ainsi que le point correspondant situé sur l'autre lien, chacun cherchant à l'attirer vers lui-même. La figure 2.16 montre un exemple de cette visualisation. Les avantages de cette méthode sont sa simplicité d'implémentation, et le fait que les liens produits n'ont pas une courbure trop prononcée, et donc que les paquets de liens sont faciles à suivre. Son plus grand défaut est au niveau des performances. En effet, le temps nécessaire pour produire la visualisation, lorsque celle-ci contient une grande quantité de liens, est relativement élevé comparé à d'autres méthodes d'agrégation d'arcs.

En plus de la visualisation *FDEB*, il existe d'autres visualisations des liens d'adjacence par agrégation d'arcs qui donnent des résultats similaires à celle-ci. Par exemple, la méthode *Geometry-Based Edge Clustering* [5], ou *GBEC*, commence par créer une maille de contrôle qui est ensuite utilisée pour regrouper les liens en paquets, en forçant les liens à passer par certains points de contrôle précis de cette maille. Un autre exemple est la méthode *Hierarchical Edge Bundles for General Graphs* [12]. Celle-ci utilise les données d'un graphe non-hiérarchique afin de créer une structure hiérarchique virtuelle qui organise les éléments de celui-ci, où les éléments sont regroupés selon leurs

Figure 2.16 – Exemple de la visualisation *FDEB*. Le graphe montré dans cet exemple est exactement le même que celui montré à la figure 2.14. La couleur rouge indique les endroits où passent une grande quantité de liens. Cette figure est tirée de [11].

affinités. Ensuite, elle utilise la méthode des *Hierarchical Edge Bundles* [10], dont nous reparlerons sous peu, afin d'afficher les liens d'adjacence existant entre les éléments en se basant sur la hiérarchie qu'elle a créée précédemment.

Les visualisations que nous venons de présenter, même si elles n'ont pas été conçues à cet effet, fonctionneraient bien avec des structures logicielles non organisées de façon hiérarchique. Par contre, comme ces visualisations sont conçues pour fonctionner avec des graphes généraux, elles ne profitent pas de la structure hiérarchique du graphe, lorsque celui-ci en a une, pour mieux guider les liens.

2.3.2 Visualisation de liens d'adjacence en utilisant une représentation matricielle

À notre connaissance, une des rares visualisations relativement efficaces qui utilisent une représentation matricielle pour afficher des liens d'adjacence est la méthode des *Package Reference Fingerprint* [1]. Cette méthode affiche les liens entrants ou sortants d'un paquetage à l'aide d'une matrice en deux dimensions. Pour une matrice représentant les liens entrants, la cellule dans le coin supérieur gauche correspond au paquetage sous analyse, et les cellules de la première colonne et de la première rangée correspondent aux paquetages qui utilisent au moins une classe de ce dernier. Les cellules internes affichent,

à l'aide de petits carrés, les classes qui sont utilisées à la fois par le paquetage associé à sa rangée et par le paquetage associé à sa colonne. La matrice utilisée pour afficher les liens sortants d'un paquetage est organisée de la même façon, mais à l'inverse : la dernière colonne est associée aux paquetages, au lieu de la première, et le paquetage sous analyse est dans le coin supérieur droit. La couleur des cellules sert à indiquer la quantité de classes utilisées par les paquetages associés à une cellule donnée. En utilisant les couleurs des cellules de la matrice, les auteurs de cette visualisation ont réussi à définir quelques patterns de couleurs qui permettent d'obtenir rapidement différentes informations sur la structure du paquetage. La figure 2.17 montre un exemple simple de cette visualisation.

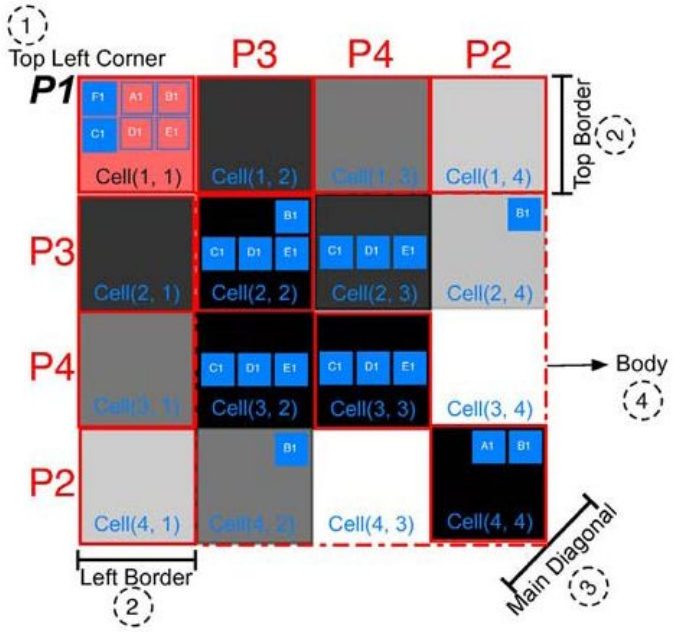


Figure 2.17 – Exemple de visualisation *Package Reference Fingerprint*. Cette figure est tirée de [1].

Le plus grand avantage offert par cette méthode est qu'elle permet de voir si certaines classes sont toujours utilisées conjointement, ce qui indiquerait que celles-ci travaillent de concert pour offrir un même service. En fin de compte, cela nous permet de voir si le paquetage est cohésif d'un point de vue fonctionnel, c'est-à-dire par rapport à l'utilisation que fait le reste du système des classes qu'il contient. Par contre, cette visualisation

n'est pas du tout adaptée à nos besoins car elle affiche seulement les liens entrants ou sortants d'un seul paquetage à la fois, ce qui évidemment ne permet pas de voir l'ensemble des liens d'un seul coup d'œil.

2.4 Visualisation simultanée de liens hiérarchiques et de liens d'adjacence

Nous avons présenté dans les deux sections précédentes des visualisations qui étaient conçues soit pour afficher des liens hiérarchiques, soit pour afficher des liens d'adjacence entre des éléments, et donc qui étaient incapables, ou du moins mal adaptées, pour afficher les deux simultanément. Dans cette section, nous allons passer en revue quelques travaux qui doivent afficher simultanément ces deux types de liens afin d'être en mesure de créer une visualisation offrant l'information voulue. Nous allons décrire rapidement, pour chaque article, les données que les auteurs veulent visualiser, la façon dont ils les visualisent, et, finalement, si leur visualisation est adaptée pour atteindre l'objectif que nous nous sommes fixés dans le chapitre 1.

Un premier exemple de visualisation simultanée de liens hiérarchiques et de liens d'adjacence est le logiciel VERSO [16, 17] sur lequel s'appuie notre système. Celui-ci affiche la hiérarchie logique d'un logiciel, c'est-à-dire la hiérarchie de modules (paquetages, espaces de noms, etc.) dans lesquels sont réparties les classes, avec une visualisation *Treemap* ou *Sunburst*, comme décrit à la section 2.2.2.1. Les liens d'adjacence, quant à eux, sont représentés à la demande, c'est-à-dire que l'utilisateur doit d'abord sélectionner une classe dans la visualisation, puis le système affiche les liens connectés à celle-ci : lorsqu'aucune classe n'est sélectionnée, le système n'affiche aucun lien d'adjacence. Pour afficher ces liens, le système diminue la saturation des classes qui ne sont pas liées à celle sélectionnée, et laisse aux classes qui le sont leur couleur habituelle. La figure 2.18 montre un exemple de l'affichage des liens d'adjacence d'une classe dans VERSO. Le plus grand inconvénient que présente cette façon d'afficher les liens d'adjacence, c'est-à-dire en utilisant la couleur des éléments, est qu'elle est incapable d'afficher l'ensemble des liens simultanément, ce qui est une des fonctionnalités les plus importantes que doit offrir notre visualisation. Il est donc évident que cette visualisation est

largement insuffisante pour répondre à nos besoins : d'ailleurs, c'est justement pour cela que nous l'avons modifiée pour y ajouter, entre autres, une façon différente d'afficher les liens d'adjacence.

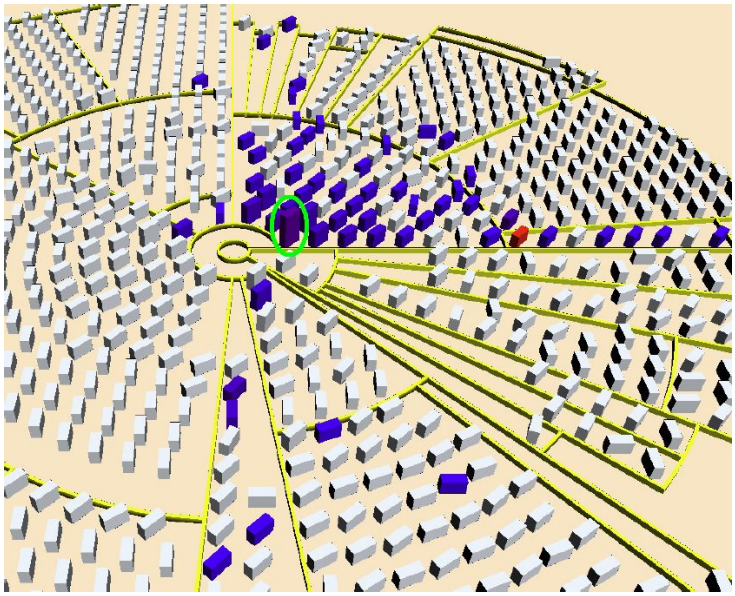


Figure 2.18 – Exemple d'affichage des liens d'adjacence dans VERSO. Les classes liées avec celle qui est encerclée ont conservé leur couleur initiale : toutes les autres sont devenues grises. Cette figure est tirée de [17].

Un autre exemple de visualisation simultanée de liens hiérarchiques et de liens d'adjacence est celle de Greevy et al. [6]. Leur visualisation est utilisée pour afficher les traces d'exécution des fonctions (*features* en anglais) d'un logiciel afin d'en comprendre le comportement dynamique. Une trace d'exécution est constituée de l'ensemble des instanciations d'objets qui se sont produites pendant l'exécution de la fonction, ainsi que des messages que se sont envoyés ces objets. Afin de pouvoir mettre en contexte les données des traces d'exécution, ils affichent aussi la structure hiérarchique définie par les liens d'héritage qui existent entre les classes du logiciel.

Pour afficher cette structure hiérarchique, ils représentent chaque classe par une boîte, puis les positionnent afin qu'elles forment une arborescence dans laquelle tous les éléments d'un même niveau sont placés côte à côte. Les liens hiérarchiques existant entre les classes sont représentés par une ligne droite de couleur noire.

Pour afficher les traces d'exécution, ils représentent chaque instanciation d'objet par une boîte placée au-dessus de la classe modèle de l'objet, en empilant ces boîtes dans le même ordre que les objets qu'elles représentent sont créés. Les liens d'adjacence existant entre ces objets, c'est-à-dire les messages qu'ils s'envoient au cours de l'exécution de la fonction, sont représentés par une ligne droite de couleur rouge. La figure 2.19 montre un exemple de leur vue d'ensemble, qui affiche tous les objets et messages créés au cours de la trace d'exécution.

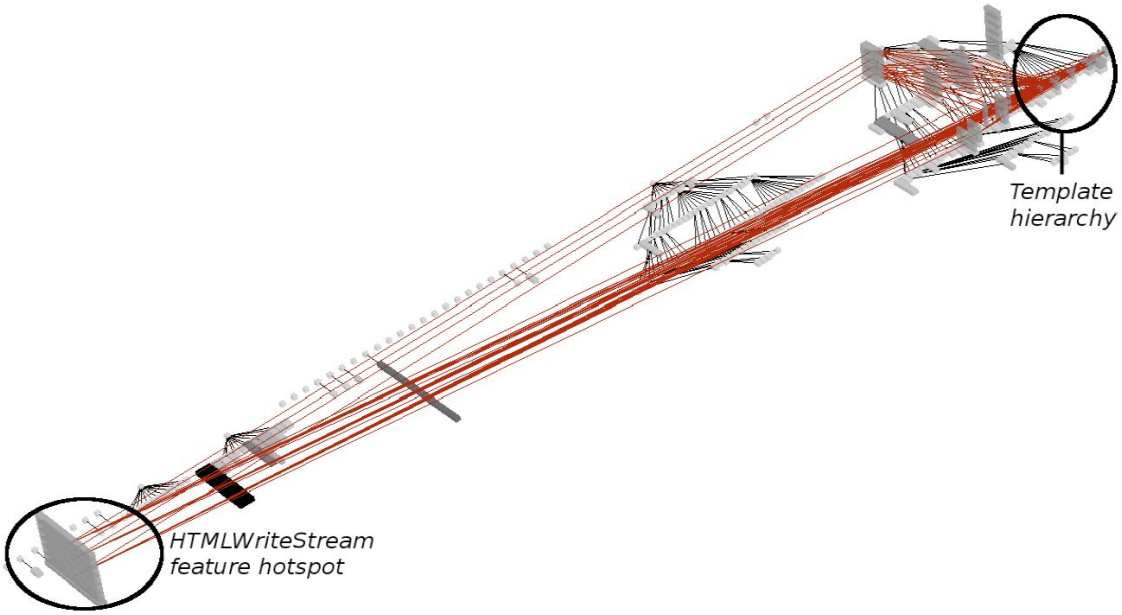


Figure 2.19 – Exemple de la vue d'ensemble offerte par la visualisation de Greevy et al. [6]. Cette figure est tirée de [6].

Il est évident pour nous que la visualisation présentée par les auteurs de [6] n'est pas du tout adaptée à l'affichage simultané de la hiérarchie d'un logiciel et de l'ensemble des liens statiques existant entre ses classes. Premièrement, l'arborescence est une représentation hiérarchique très peu extensible. Pour ce qui est de la visualisation des liens d'adjacence, la ligne droite est une représentation très inefficace qui ne permet pas de voir les tendances dans les liens, et qui en plus dans cette visualisation crée de l'interférence avec les liens de la hiérarchie. De plus, ils n'indiquent pas le sens des liens, nous empêchant ainsi de voir, pour chaque message, quel objet l'envoie et lequel en est le

récipiendaire. Au final, même si cette visualisation était efficace pour afficher les informations qui les intéressaient, elle ne l'est pas pour les informations que nous voulons visualiser.

La méthode des *Hierarchical Edge Bundles* [10], ou *HEB*, est une méthode d'agrégation d'arcs à l'instar de celles mentionnées à la section 2.3.1 mais qui, contrairement à elles, nécessite absolument une structure hiérarchique pour fonctionner. Cette méthode ne peut donc qu'afficher des liens d'adjacence existant entre des éléments organisés de façon hiérarchique. Depuis sa création, la méthode des *HEB* a gagné rapidement en popularité et a été utilisée dans plusieurs visualisations devant afficher simultanément des liens hiérarchiques et des liens d'adjacence, y compris la nôtre ; nous expliquerons donc plus en détail cette méthode à la section 4.3.1. Pour l'instant, voici quelques exemples de visualisation utilisant cette méthode.

Un premier exemple est le système EXTRAVIS [4, 9], développé par le même auteur que la méthode des *HEB*. Ce système offre deux visualisations complémentaires permettant d'afficher une trace d'exécution, c'est-à-dire d'afficher les appels faits entre les éléments du logiciel au cours de son exécution. La première visualisation utilisée, nommée *Massive Sequence View*, permet d'obtenir une vue d'ensemble des appels faits entre les éléments du logiciel en fonction du moment, lors de l'exécution visualisée, où ils ont été réalisés. Cette visualisation a donc été conçue en premier lieu pour voir clairement l'ordre dans lequel les appels ont été faits, ce qui la rend moins efficace pour afficher les tendances générales dans les liens d'adjacence. Ainsi, même si celle-ci permet de visualiser l'ensemble des liens d'adjacence existant entre les éléments, tout en affichant la structure hiérarchique de ceux-ci, étant donné que la notion de temps n'existe pas pour l'affichage de liens statiques, cette visualisation n'est pas pertinente pour ce que nous voulons réaliser. La deuxième visualisation qu'ils utilisent, nommée *Hierarchical Edge Bundle View*, sert à afficher essentiellement la même chose que la première mais en faisant fi du moment où les appels ont été réalisés : celle-ci serait donc tout à fait appropriée pour afficher des liens statiques. Dans cette visualisation, la structure hiérarchique du logiciel est représentée par un placement radial 2D, comme celui que nous allons décrire à la section 3.3, et les liens d'adjacence sont représentés avec la méthode des *HEB*.

La taille de chaque courbe est utilisée pour indiquer la quantité d'appels fait entre les deux éléments qu'elle unit, tandis que leur couleur est utilisée pour indiquer dans quelle direction les appels se font. La figure 2.20 montre un exemple de cette visualisation.

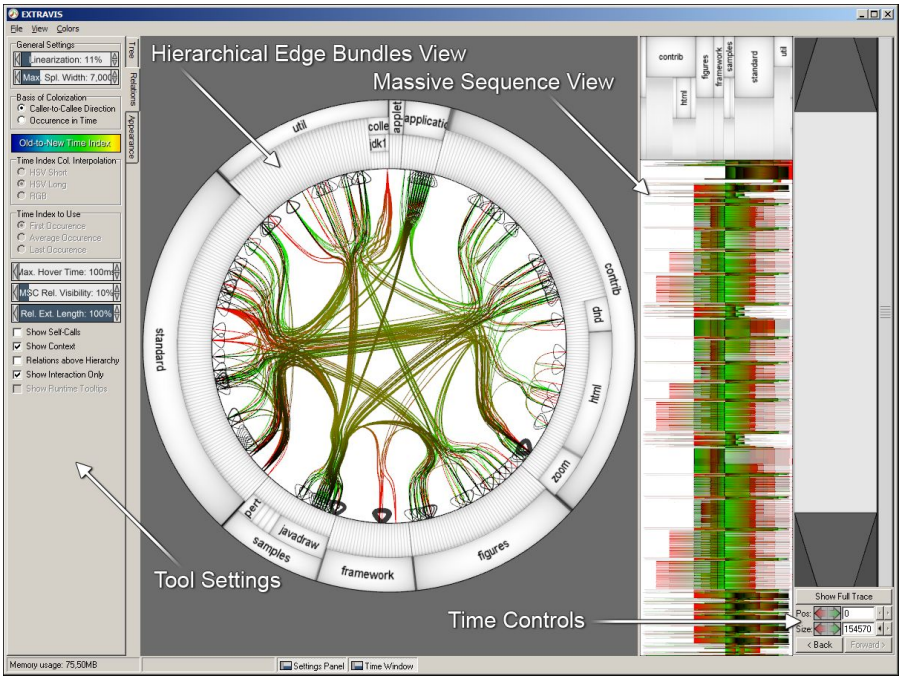


Figure 2.20 – Exemple d'une visualisation obtenue avec le système EXTRAVIS. La visualisation *Hierarchical Edge Bundles View*, au centre, est celle qui nous intéresse car elle est mieux adaptée pour afficher des liens statiques que la *Massive Sequence View* située à droite.

Même si cette visualisation est à la base assez similaire à la nôtre, il existe quand même quelques points de différentiation importants entre les deux. Premièrement, leur visualisation est en deux dimensions tandis que la nôtre offre un environnement 3D dans lequel l'utilisateur peut naviguer librement. Aussi, leur visualisation n'utilise qu'une seule sorte de placement pour afficher la hiérarchie alors que notre système en offre trois à l'utilisateur, en lui permettant de changer de type de placement en temps réel. Finalement, cette visualisation offre un peu moins de méthodes interactives pour filtrer l'information qu'elle affiche, principalement au niveau des liens d'adjacence, comparée à la nôtre.

Une visualisation similaire à celle que nous venons de décrire est le système So-

lidSX, de Telea et al. [24], qui est utilisé dans cet article pour visualiser un graphe d'appels extrait d'un logiciel dont les éléments sont organisés de façon hiérarchique. Par contre, contrairement à EXTRAVIS, SolidSX est conçu pour afficher des liens statiques, signifiant ainsi que la visualisation n'a pas à se préoccuper d'afficher de l'information concernant l'ordre des appels ou le moment précis auquel ils ont été effectués.

Finalement, une visualisation très récente, qui en fait vient tout juste d'être publiée au moment d'écrire ces lignes, est la méthode des *3D Hierarchical Edge Bundles* de Caserta et al. [3]. Cette visualisation affiche, dans un espace en trois dimensions, la hiérarchie logique d'un logiciel ainsi que les liens d'adjacence statiques qui existent entre ses classes. La structure hiérarchique est affichée en utilisant la métaphore de la ville, comme dans VERSO ou CodeCity [30] : les classes sont représentées par des boîtes et la structure de la hiérarchie est affichée à l'aide d'une représentation par inclusion. Dans leur visualisation, deux placements sont offerts pour représenter la hiérarchie d'un logiciel : un placement par imbrication et un placement qu'ils ont nommé *Street Layout*, qui représente la hiérarchie de paquetages comme des rues perpendiculaires. Les liens d'adjacence sont affichés en intégrant la méthode des *HEB* dans chacun de leur placement, et ils utilisent la couleur des liens pour les quantifier ainsi que pour afficher leur direction. La figure 2.21 montre un exemple de cette visualisation.

2.5 Version initiale de VERSO

Comme nous l'avons déjà mentionné, notre système a été créé à partir du système VERSO, développé par Langelier et al. [16, 17]. Afin de clarifier ce qui était déjà présent dans le système avant notre intervention, dans cette section nous allons faire un survol des principales fonctionnalités qui existaient déjà dans la version de VERSO à partir de laquelle nous avons travaillé.

Le logiciel VERSO est un système qui permet de visualiser la hiérarchie logique d'un logiciel ainsi que la valeur de certaines métriques calculées sur ses classes. Comme mentionné à la section 2.2.2.1, VERSO permet d'afficher la hiérarchie d'un logiciel en utilisant la visualisation *Treemap* ou *Sunburst*. Il est à noter que la version de VERSO

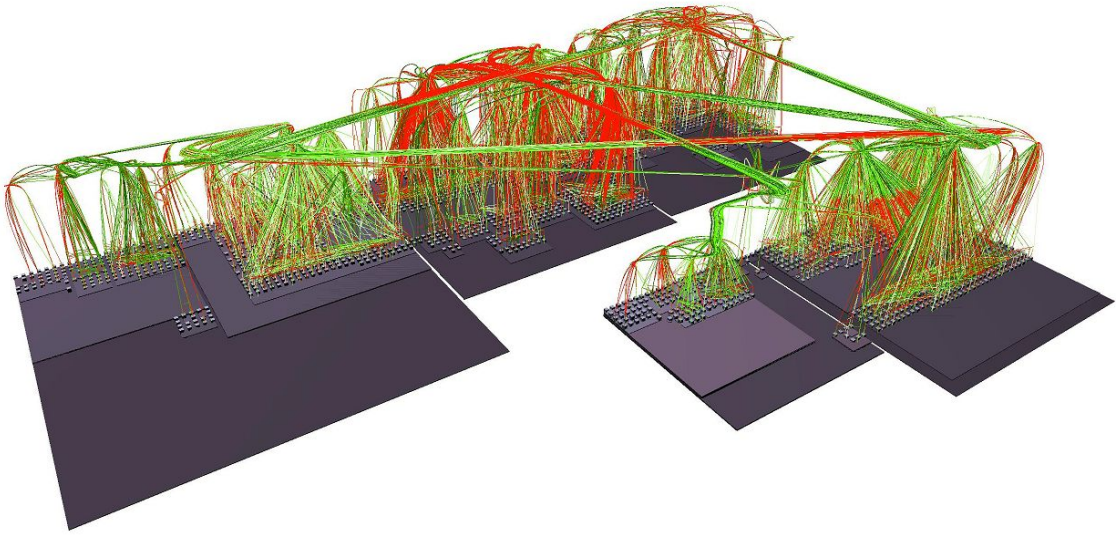


Figure 2.21 – Exemple de la visualisation *3D HEB*.

que nous avons utilisée, VERSO Java, n'est pas exactement la même que celle décrite dans [16, 17], la principale différence étant que celle-ci n'implémente que le placement *Treemap*. Mis à part cela, les principes essentiels de la visualisation restent les mêmes que ceux présentés dans [16, 17].

Les éléments du logiciel, c'est-à-dire les classes et les interfaces, sont représentés respectivement à l'aide d'une boîte et d'un cylindre et certains de leurs attributs graphiques représentent la valeur d'une métrique. La hauteur d'une boîte représente la taille de la classe, sa couleur représente son niveau de couplage avec les autres classes et son orientation par rapport à l'axe *y* représente son niveau de cohésion. La figure 2.22 montre un exemple d'une classe où ces trois valeurs augmentent simultanément.

Les façons d'interagir avec la visualisation sont aussi les mêmes que celles décrites dans [16, 17]. Comme mentionné à la section 2.4, les liens d'adjacence sont affichés à la demande et sont représentés en diminuant la saturation des classes qui ne sont pas liées à la classe sélectionnée. La navigation se fait à l'aide de la souris et du clavier et celle-ci permet de zoomer sur la visualisation, de faire tourner la caméra autour d'elle et de déplacer la caméra horizontalement ou verticalement. Une fonctionnalité qui était présente dans VERSO Java mais qui ne l'était pas dans la version de VERSO présentée

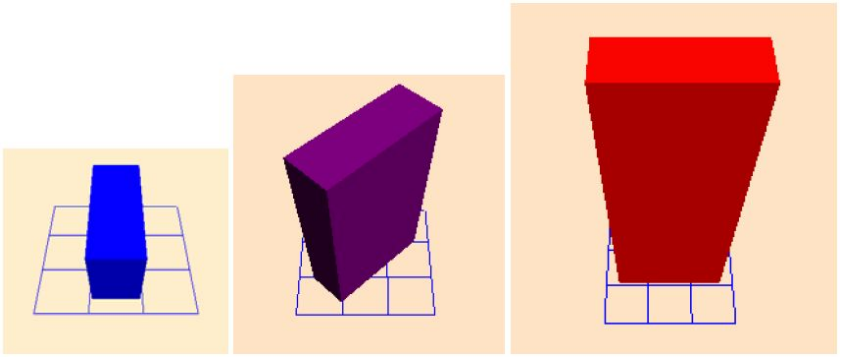


Figure 2.22 – Exemple de la visualisation de métriques dans VERSO. Dans cet exemple, en suivant les images de gauche à droite, la taille de la classe ainsi que son niveau de couplage augmentent tandis que sa cohésion diminue. Ces trois caractéristiques sont représentées respectivement par une augmentation de la taille de la boîte, par sa couleur allant du bleu au rouge et par son orientation devenant parallèle à l'axe z.

dans [16, 17] est le mécanisme de fermeture des paquetages. Ce mécanisme permet de simplifier la visualisation en remplaçant un ensemble de classes et de paquetages par une seule entité, qui elle représente le paquetage parent de ces éléments. L'entité utilisée pour cela est une boîte comme pour les classes, dont les dimensions correspondent au paquetage qu'elle représente et qui est positionnée de façon à coïncider avec les limites de celui-ci dans la représentation hiérarchique. Pour fermer des paquetages, l'utilisateur sélectionne un niveau, puis tous les paquetages situés à ce niveau sont fermés, cachant ainsi les détails des niveaux inférieurs. Cela permet d'obtenir une vue simplifiée de la hiérarchie du logiciel, ce qui est utile lorsque l'on veut seulement voir les parties principales composant celui-ci. Il est à noter que la couleur des boîtes représentant les paquetages fermés est utilisée pour afficher la valeur d'une métrique associée aux paquetages, comme c'est le cas pour les classes.

CHAPITRE 3

VISUALISATION DE LA HIÉRARCHIE

Comme mentionné dans le chapitre 1, la visualisation que nous voulons offrir est composée de deux parties : la visualisation des liens hiérarchiques existant entre les éléments d'un logiciel (la hiérarchie de celui-ci) et la visualisation des liens d'adjacence existant entre ces mêmes éléments. Évidemment, avant de pouvoir afficher des liens d'adjacence entre les éléments d'un logiciel, il faut tout d'abord trouver une représentation adéquate de ces éléments. La première étape dans la construction de notre outil de visualisation consiste donc à trouver une façon de représenter clairement les éléments d'un logiciel ainsi que les relations hiérarchiques existant entre ceux-ci. De plus, la représentation hiérarchique utilisée doit être conçue de façon à bien supporter l'ajout de liens d'adjacence, sans quoi celle-ci nous serait d'aucune utilité. Le présent chapitre décrira donc les représentations hiérarchiques offertes par notre outil, de même que les avantages et inconvénients de chacune. La section 3.1 mentionne quelle catégorie de représentations hiérarchiques nous avons décidé d'utiliser dans notre outil et explique les raisons qui ont motivé ce choix. Chacune des trois sections suivantes décrit une des trois représentations hiérarchiques incluses dans notre outil. La section 3.2 décrit une représentation basée sur le placement *Treemap*, la section 3.3 décrit une représentation de type circulaire basée sur l'arbre radial et, finalement, la section 3.4 décrit une représentation dérivée de celle présentée dans la section précédente. Tous les exemples de notre visualisation qui seront présentés dans ce chapitre utilisent, sauf avis contraire, le logiciel *JHotDraw*, qui est composé de 583 classes réparties dans 67 paquetages, qui eux sont étalés sur 6 niveaux de profondeur (incluant la racine).

3.1 Choix de la catégorie de représentations hiérarchiques

Comme expliqué dans le chapitre 2, la plupart des façons de représenter une hiérarchie sont regroupées en deux grandes familles : les représentations de type nœuds-liens

et les représentations par inclusion. Le premier point à décider consiste donc à savoir laquelle de ces catégories de représentations serait la plus appropriée pour notre outil. Les deux aspects les plus importants que nous recherchons dans une représentation hiérarchique sont une bonne extensibilité et une bonne capacité à intégrer les liens d'adjacence que nous allons y ajouter par la suite, comme il le sera expliqué dans le chapitre 4.

Considérons d'abord les représentations de type nœuds-liens. Même si celles-ci fonctionnent bien pour afficher des hiérarchies relativement petites (quelques dizaines de nœuds), elles sont peu extensibles, devenant facilement encombrées sitôt que nous passons à des hiérarchies ayant une taille moyenne (quelques centaines de nœuds) et sont totalement illisibles pour de grandes hiérarchies (quelques milliers de nœuds). Pour ce qui est de l'intégration des liens d'adjacence, l'algorithme que nous utilisons pour afficher ces liens les représente explicitement à l'aide de courbes, ce qui implique qu'utiliser une représentation de type nœuds-liens pour afficher les liens hiérarchiques créerait de l'interférence avec l'affichage des liens d'adjacence. Le chapitre 4 expliquera pourquoi il est incontournable de représenter de façon explicite les liens d'adjacence, signifiant ainsi que le problème d'interférence ne peut être évité si nous utilisons une représentation de type nœuds-liens pour afficher les liens hiérarchiques. La figure 3.1 montre un exemple d'interférence dans une visualisation de type arborescence si nous y ajoutons des liens d'adjacence.

Les représentations hiérarchiques par inclusion, quant à elles, sont beaucoup plus efficaces que les représentations de type nœuds-liens par rapport aux deux aspects recherchés. En effet, l'absence de représentation explicite des liens hiérarchiques dans celles-ci permet non seulement de réduire l'interférence créée lorsque nous y ajoutons des liens d'adjacence, mais leur permet aussi d'utiliser de manière plus efficace l'espace visuel qui leur est alloué, les rendant ainsi très extensibles.

La grande extensibilité des méthodes par inclusion ainsi que leur excellente capacité à intégrer des liens d'adjacence, surtout en comparaison à la grande faiblesse des représentations de type nœuds-liens par rapport à ces deux aspects, nous ont incités à n'utiliser que cette première catégorie de représentation. Les prochaines sections de ce chapitre présenteront les représentations par inclusion utilisées dans notre outil pour vi-

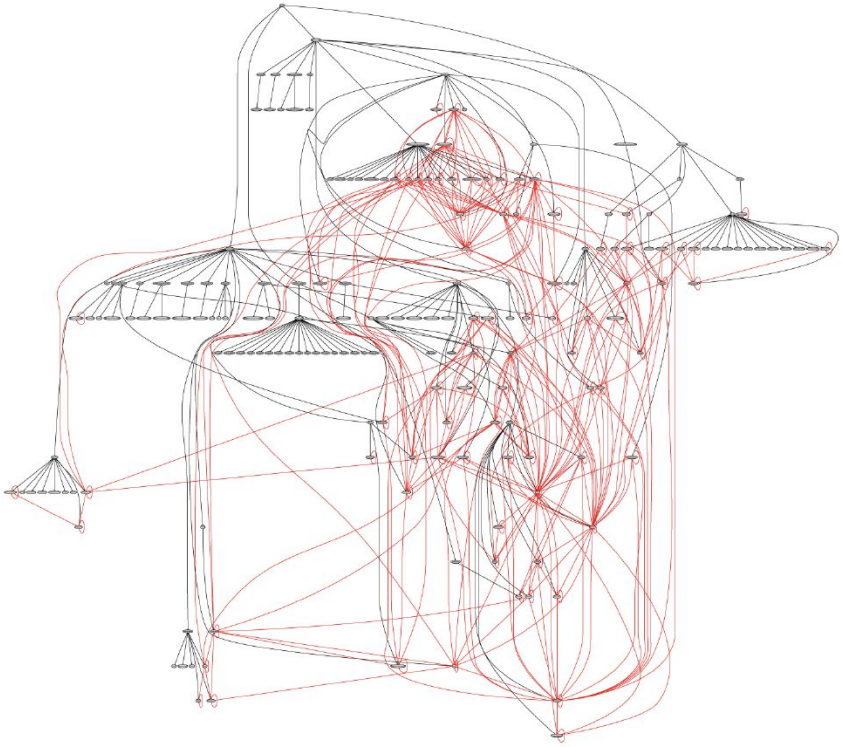


Figure 3.1 – Exemple d'interférence dans une visualisation de type arborescence. Les liens en rouge représentent le graphe d'appel du logiciel. Figure tirée de [10].

sualiser la structure hiérarchique d'un logiciel.

3.2 Placement *Treemap* amélioré

Comme il a été expliqué dans la section 2.5, l'outil de base à partir duquel nous avons travaillé, VERSO, utilise déjà un placement *Treemap* standard pour représenter la hiérarchie de logiciels écrits en Java. Ce placement supporte mal l'ajout de liens d'adjacence, la visualisation devenant facilement encombrée pour tous les logiciels testés excepté ceux de très petite taille. Le placement *Treemap* présentant tout de même des avantages indéniables pour la visualisation de structures hiérarchiques, nous avons appliqué quelques modifications à celui-ci afin de l'adapter à la visualisation concurrente de la structure hiérarchique et des liens d'adjacence du système. La section 3.2.1 explique comment nous avons passé d'un placement *Treemap* par partition à un placement *Treemap* par imbrication. La section 3.2.2 décrit comment nous affichons les paquetages ouverts dans ce placement. La section 3.2.3 explique comment nous améliorons le placement des enfants dans la représentation. Finalement, la section 3.2.4 explique comment utiliser de façon plus efficace l'espace disponible dans le placement.

3.2.1 Visualisation de la hiérarchie par imbrication

Un des critères de base qui a guidé le développement des *Treemap* était l'utilisation la plus efficace que possible de l'espace d'affichage. La partition d'un rectangle parent pour représenter ses enfants est, de ce point de vue, le choix le plus judicieux pour arriver à cette fin. Par contre, dans le cas qui nous intéresse, en plus de la visualisation de la hiérarchie que fournit un placement *Treemap* standard, nous voulons aussi afficher, de façon concurrente, des liens d'adjacence entre les entités. Comme il le sera expliqué au chapitre 4, ces liens sont affichés graphiquement de façon explicite et, dû à ce fait, l'espace requis pour obtenir une visualisation satisfaisante de ce type de liens est en général plus important que l'espace qu'offre un *Treemap* utilisant un algorithme par partition. La façon la plus simple pour pallier ce problème est tout simplement d'espacer l'affichage des nœuds de la hiérarchie. Évidemment, faire cela augmente l'espace qu'utilise

le placement *Treemap* et le rend ainsi moins extensible. Il est donc nécessaire, lorsqu'on détermine la quantité d'espace supplémentaire que l'on veut lui allouer, de faire un compromis entre l'extensibilité offerte par celui-ci et la qualité de la visualisation des liens d'adjacence qu'il permet d'obtenir.

La façon utilisée dans notre outil pour espacer les nœuds est de représenter la structure de la hiérarchie par imbrication des rectangles plutôt que par partition de ceux-ci, comme décrit dans [13]. Pour ce faire, nous ajoutons une bordure vide à l'intérieur de chaque paquetage qui n'est pas une feuille et nous laissons un espace entre chacun de ses sous-paquetages. L'algorithme *Treemap* implémenté travaille seulement avec la hiérarchie de paquetages, sans considérer les classes, pour effectuer son placement. Dans ce cas-ci, les feuilles de la hiérarchie sont les paquetages qui n'ont pas de sous-paquetages, et ce même s'ils contiennent des classes. Il est à noter que nous modifions la hiérarchie afin que chaque groupe de classes frères soit placé dans un (faux) sous-paquetage supplémentaire à leur paquetage parent, à condition que celui-ci ne soit pas une feuille. Ceci fait en sorte qu'un paquetage contient soit des paquetages, soit des classes, ce qui permet de simplifier la façon dont l'algorithme laisse un espace entre les groupes de classes et leurs paquetages frères. L'accumulation des bordures et des espaces permet en général d'écarter suffisamment les éléments pour désengorger de manière satisfaisante la visualisation des liens d'adjacence. La taille des bordures étant un paramètre qu'on peut faire varier, nous pouvons ainsi essayer différentes tailles de bordures afin de trouver un bon compromis, pour chaque logiciel, entre la taille utilisée par la visualisation et la clarté de celle-ci.

La figure 3.2 montre, étape par étape, un exemple simple de transformation d'un placement *Treemap* par partition en un placement *Treemap* par imbrication. La figure 3.3 montre la différence entre le placement *Treemap* standard déjà implémenté dans VERSO et le nouveau placement décrit dans cette section.

3.2.2 Attributs visuels des paquetages ouverts

La figure 3.3 montre que le placement *Treemap* par imbrication permet non seulement d'espacer les éléments, mais aussi de rendre plus explicite la visualisation de la

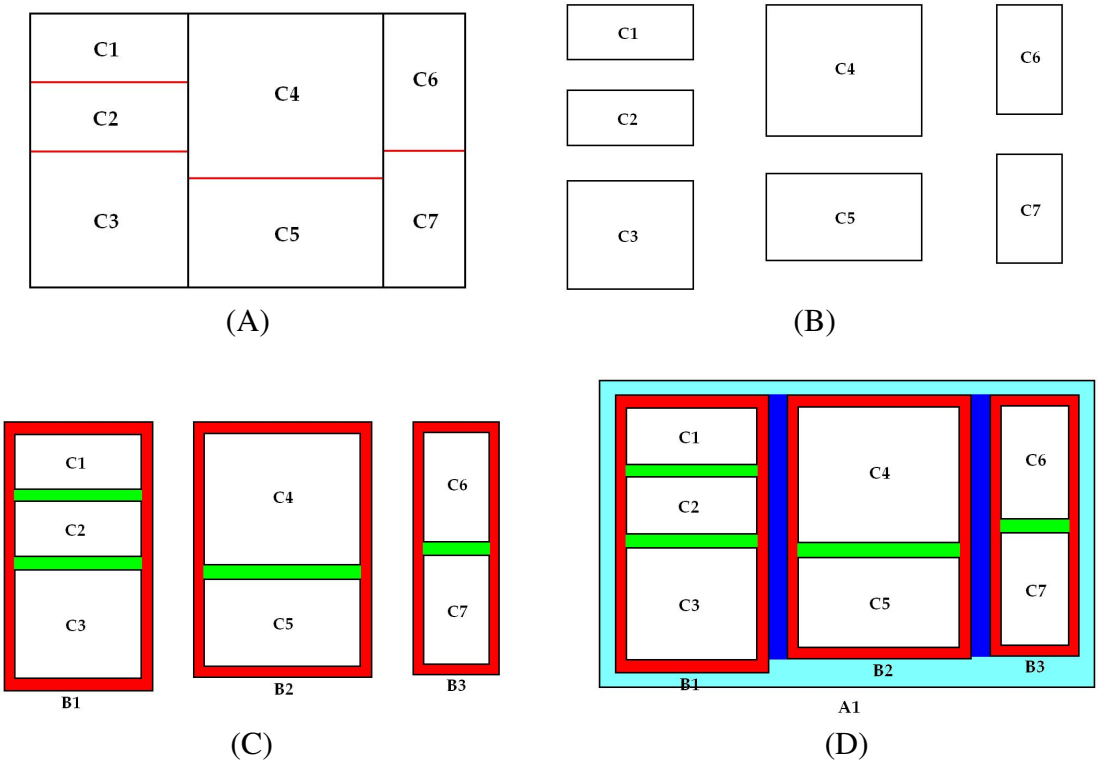
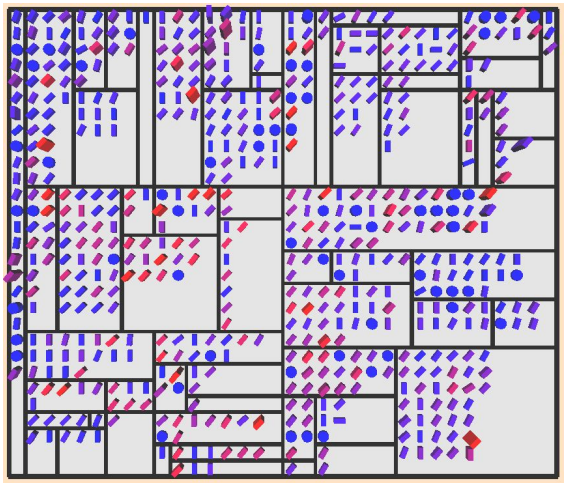
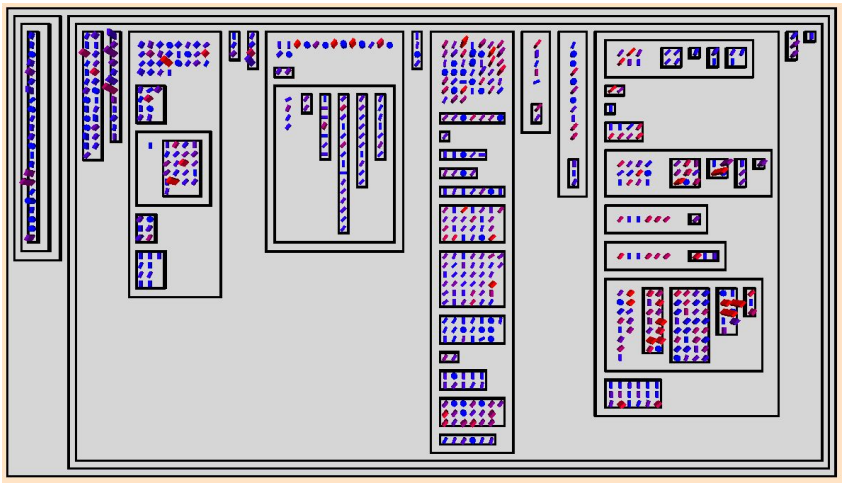


Figure 3.2 – Transformation d'un placement *Treemap* par partition en un placement *Treemap* par imbrication. L'image A) montre un exemple de placement *Treemap* par partition. Notre algorithme d'imbrication fonctionne de manière ascendante, c'est-à-dire qu'il commence par modifier les éléments les plus profonds de la hiérarchie en leur ajoutant les bordures et les espaces nécessaires, puis utilise les nouveaux éléments résultants, avec leurs nouvelles dimensions, pour construire le nouvel élément parent. Les rectangles représentant les feuilles de la hiérarchie qu'on obtient avec le placement *Treemap* par partition sont ainsi utilisés comme éléments de base par notre algorithme d'imbrication. L'image B) montre ceux qui seront utilisés dans cet exemple. Comme ces éléments sont des feuilles, nous n'ajoutons pas de bordures à l'intérieur, et donc leurs dimensions restent inchangées. L'image C) montre les parents obtenus avec ces éléments. Ces paquetages séparent leurs sous-éléments de façon horizontale, les plaçant ainsi de haut en bas, alignés à gauche. Les espaces placés entre chaque sous-paquetage sont colorés en vert, tandis que les bordures internes du paquetage parent sont colorées en rouge. L'image D) montre la nouvelle racine obtenue en utilisant les résultats de l'image C). Les sous-éléments de cette racine sont séparés de façon verticale, et sont ainsi placés de gauche à droite, alignés dans le haut de leur parent. Les espacements sont colorés en bleu et les bordures internes en turquoise.



(A)



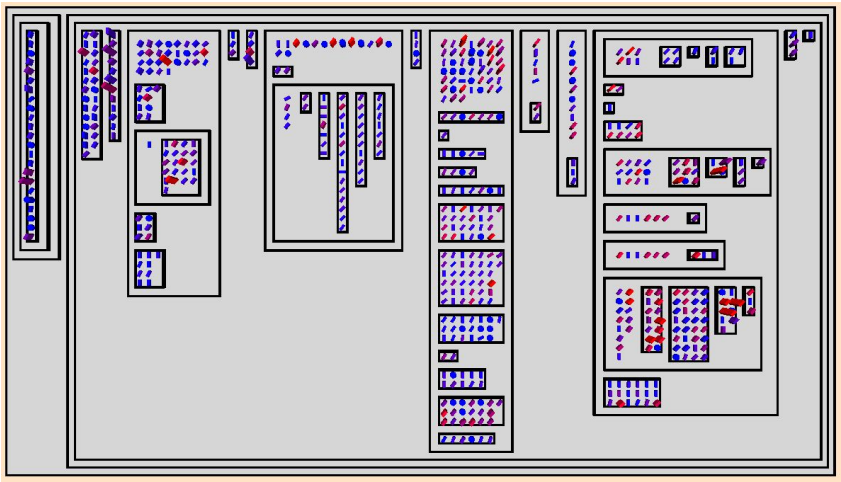
(B)

Figure 3.3 – Exemple d'un placement *Treemap* par imbrication. Cette figure montre la différence entre le placement *Treemap* par partition, déjà présent dans VERSO (image A)), et le placement *Treemap* par imbrication (image B)), obtenu en ajoutant des bordures à l'intérieur des paquetages et un espace entre leurs sous-paquetages.

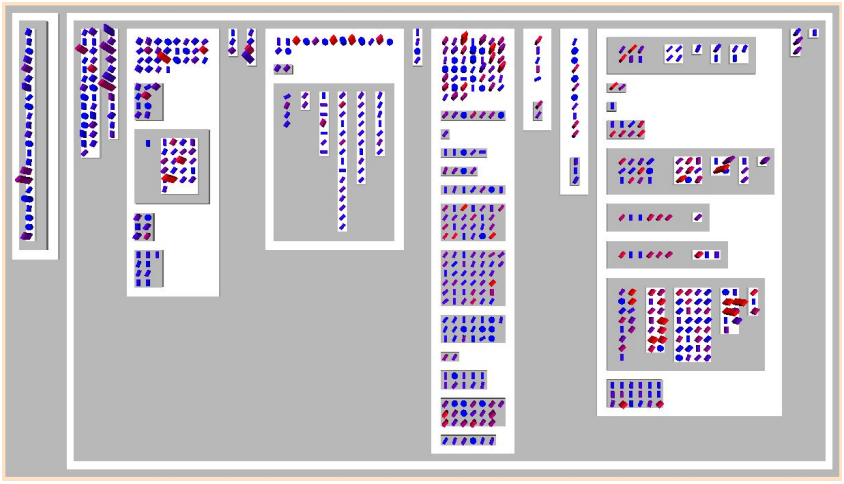
hiérarchie comparée au placement *Treemap* par partition, dans lequel nous devons observer l'alternance du sens des séparations pour la faire ressortir. La raison qui nous a incités à utiliser le placement *Treemap* par imbrication était principalement pour le premier de ces deux avantages, sans vraiment prendre en considération le deuxième. Malgré tout, avoir une visualisation de la hiérarchie plus explicite étant un avantage indéniable, nous avons décidé d'ajouter quelques modifications nous permettant de profiter autant que possible de cet avantage « bonus » que nous offre ce type de placement, et ce sans nécessiter plus d'espace d'affichage ni dégrader la visualisation en général.

Les paquetages ouverts sont ceux qui, à ce moment-là, n'intéressent pas l'utilisateur, celui-ci étant plutôt intéressé par leurs sous-éléments. Comme expliqué dans la section 2, la couleur d'un paquetage fermé est un indice visuel associé à une métrique quelconque servant à donner de l'information sur celui-ci. Dans le cas des paquetages ouverts, cet indice visuel n'est pas nécessaire étant donné qu'il n'intéresse pas l'utilisateur et pourrait même, au contraire, nuire à la visualisation. Afin de régler ce problème tout en continuant d'utiliser de façon utile la couleur des paquetages ouverts, nous avons décidé de redéfinir la façon de leurs associer une couleur. Au lieu de la couleur obtenue par association avec une métrique, tous les paquetages ouverts sont blancs ou gris, en alternant entre ces deux couleurs lorsqu'on change de niveau. Un premier avantage offert par ce changement est qu'il permet de voir plus facilement à quel niveau se trouve un élément donné. Par exemple, si l'on regarde le système avec une vue de dessus, il suffit de compter, à partir d'un l'élément quelconque, le nombre d'alternances de couleur jusqu'à la racine pour savoir à quel niveau de la hiérarchie il se trouve. Par contre, selon nous le plus gros avantage offert par cette alternance de couleur est qu'elle permet de regrouper très facilement les éléments de la hiérarchie qui sont des frères. La figure 3.4 montre le placement obtenu lorsqu'on alterne les couleurs des paquetages au lieu de les délimiter avec une bordure.

Notre outil offrant un monde 3D interactif, nous avons décidé de profiter de cette troisième dimension pour renforcer la visualisation de la hiérarchie. Pour ce faire, lorsqu'un paquetage est ouvert, nous le représentons avec une boîte ayant la même longueur et la même largeur que celle le représentant lorsqu'il est fermé, mais avec une hauteur



(A)



(B)

Figure 3.4 – Placement obtenu en ajoutant l'alternance de couleurs. L'image A) montre le placement obtenu à la section précédente. L'image B) montre le placement obtenu après avoir ajouté l'alternance de couleur, où la racine se voit attribuée arbitrairement la couleur grise.

beaucoup plus petite, qui est la même pour tous les paquetages ouverts. Ensuite, tous ses sous-éléments sont placés visuellement sur la boîte le représentant, donnant ainsi un effet pyramidal à la visualisation du logiciel, ce qui procure un indice visuel supplémentaire pour représenter les relations hiérarchiques. Voir la figure 3.5 pour un exemple de cet effet pyramidal.

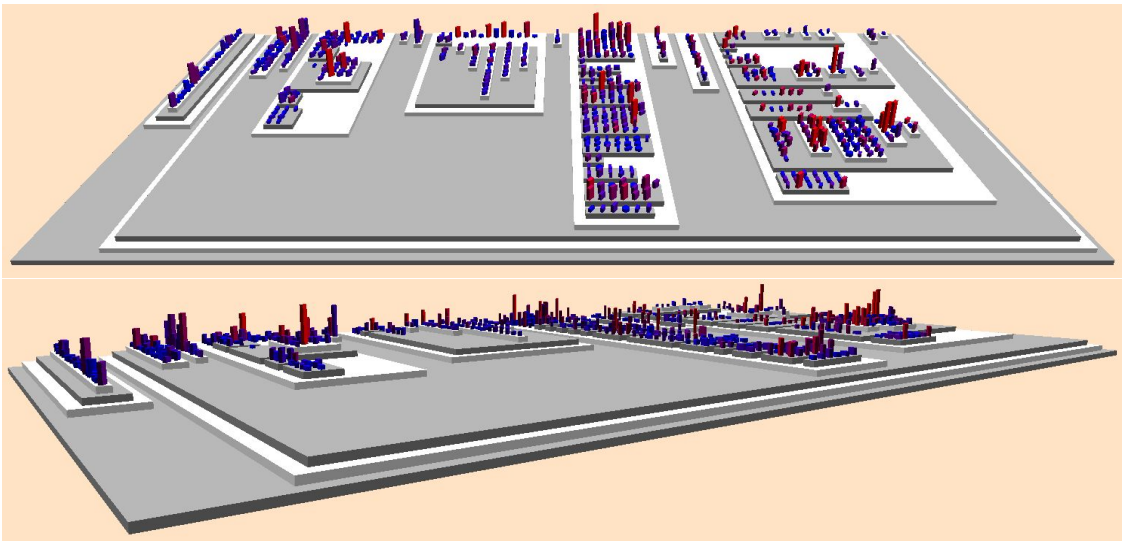


Figure 3.5 – Exemple de l'effet pyramidal qu'on obtient lorsqu'on place les sous-éléments d'un paquetage ouvert sur celui-ci.

3.2.3 Amélioration du placement des enfants

Même avec les modifications apportées jusqu'à maintenant à l'algorithme *Treemap* standard, la façon dont notre algorithme place les éléments dans l'espace de leur parent reste inchangée par rapport à celui-ci. Tout comme l'algorithme initial, notre algorithme, dans sa forme actuelle, place les éléments frères sur une même ligne ou sur une même colonne (selon le sens de la séparation) à l'intérieur de leur parent. Ceci ne pose pas vraiment un problème en soi pour la visualisation de la hiérarchie, mais peut nuire grandement à la visualisation des liens d'adjacence qui seront ajoutés par la suite. En effet, le fait d'aligner un groupe de sous-paquetages frères crée ce qu'on pourrait appeler un « corridor » de liens traversant leur paquetage parent. Ceci fait en sorte que tous les liens devant passer d'un sous-paquetage à un de ses frères empruntent tous le même chemin

étroit, se retrouvant ainsi enchevêtrés dans ce « corridor ». Cette situation se présente dès qu'un paquetage contient quelques sous-paquetages et qu'il y a beaucoup de communication entre eux, et ce indépendamment de leurs positions dans la hiérarchie. Il va de soi que le problème s'accentue à mesure que le nombre de sous-paquetages ou que la quantité de liens entre eux augmente. Évidemment, une telle situation rend la visualisation des liens d'adjacence très encombrée (voir la figure 3.6). De plus, les liens devant passer au-dessus de tous les sous-paquetages se trouvant entre ceux qu'ils relient, cela crée aussi plus d'interférence entre l'affichage de la hiérarchie et l'affichage des liens d'adjacence.

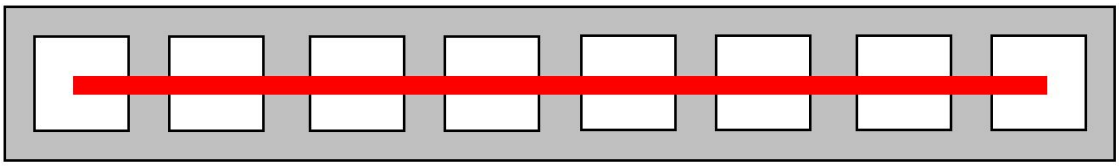


Figure 3.6 – Exemple de l'alignement causé par le placement *Treemap* standard. Le paquetage montré en exemple contient huit sous-paquetages de même taille. En les plaçant de la manière standard, ils se retrouvent tous parfaitement alignés sur une seule et même ligne. La ligne rouge représente le chemin que devront emprunter tous les liens allant d'un sous-paquetage à un autre. Ce « corridor » étroit force les liens à se chevaucher, rendant la visualisation de ceux-ci très encombrée. De plus, il est plus difficile de cette façon de savoir exactement où commence et où se termine un lien. Par exemple, dans l'image ci-haut, il est impossible de déterminer s'il n'y a qu'une seule ligne rouge, allant du premier au dernier rectangle, ou deux lignes, une allant du premier au quatrième rectangle et l'autre du quatrième au dernier, et ainsi de suite.

La solution à ce problème serait de mieux disposer les éléments dans leur parent, de façon à ce qu'ils soient moins alignés. La façon que nous avons envisagée pour atteindre cet objectif est de placer les éléments sur une grille aussi régulière et carrée que possible, au lieu de les placer sur une seule ligne ou une seule colonne. En effet, cela augmenterait le nombre de chemins entre les éléments que pourraient emprunter les liens, évitant ainsi qu'ils passent tous par un unique chemin étroit. La méthode que nous avons développée pour améliorer le placement des éléments utilise toujours la méthode de placement standard, sans rien changer à ses principes fondamentaux. À la place, nous avons décidé d'effectuer un prétraitement qui modifie la structure de la hiérarchie afin de la rendre

binaire, avant que celle-ci ne soit traitée par l'algorithme de placement. Pour rendre la hiérarchie binaire, c'est-à-dire que les nœuds ne possèdent pas plus de deux enfants, à chaque fois qu'un paquetage a plus de deux sous-paquetages, nous les séparons en deux groupes et les plaçons dans deux sous-paquetages virtuels distincts. Ces sous-paquetages virtuels viendront ensuite guider l'algorithme de placement afin qu'il dispose mieux les sous-paquetages dans leur parent.

La nouvelle hiérarchie que nous obtenons doit non seulement être binaire, mais doit en plus être aussi balancée que possible. Un bon balancement est essentiel pour obtenir de bons résultats, car la régularité, et ainsi la qualité de la grille virtuelle sur laquelle seront placés les sous-paquetages est directement liée au balancement de la nouvelle hiérarchie binaire utilisée pour faire le placement. Pour obtenir un bon balancement, lorsque nous séparons les paquetages frères en deux groupes, nous nous assurons que ces groupes soient aussi égaux que possible par rapport au poids total des paquetages qu'ils contiennent, et non pas par rapport au nombre de paquetages inclus dans chacun d'eux. Pour ce faire, nous ajoutons les paquetages un par un, en ordre décroissant de poids, et chacun d'eux est ajouté au groupe avec le plus petit poids total à ce moment-là.

Comme mentionné précédemment, nous ne changeons rien aux bases fondamentales de l'algorithme de placement standard, mais il est tout de même nécessaire d'y apporter quelques changements mineurs afin qu'il sache comment traiter les paquetages virtuels que nous avons ajoutés dans la hiérarchie. La seule différence entre le traitement d'un paquetage normal et un paquetage virtuel est que ce dernier ne contient pas de bordure interne. Par contre, afin d'éviter que certains éléments se retrouvent collés dans le placement, l'algorithme ajoute quand même un espace entre un paquetage virtuel et son voisin. Les seuls autres changements apportés servent à modifier l'algorithme d'affichage afin que celui-ci n'affiche pas les paquetages virtuels dans la visualisation.

Une fois tous ces changements apportés, lorsque l'algorithme de placement standard traite la hiérarchie modifiée, cela aura pour effet qu'au lieu de placer un groupe de paquetages frères sur une même ligne, ceux-ci seront placés sur deux colonnes, et vice-versa pour les paquetages qui seraient normalement placés sur une seule colonne. Dû au fait que la hiérarchie est balancée autant que possible, ces paires de lignes ou de colonnes

seront autant que possible de la même longueur. En appliquant récursivement cette division, cela fait en sorte que nous créons une grille aussi régulière et aussi carrée que possible pour placer, dans leur parent, les groupes de paquetages frères de la hiérarchie. Ce placement permet, au final, de créer autant de chemins que possible pour les liens entre les paquetages. Voir la figure 3.7 pour un exemple, étape par étape, de l'application de cet algorithme. Voir la figure 3.8 pour un exemple du résultat final.

Un autre avantage apporté par ce nouvel algorithme de placement est qu'il effectue une certaine amélioration du rapport hauteur/largeur des rectangles représentant les paquetages. Comme expliqué dans [2], des rectangles longs et minces sont à proscrire car leur taille respective, et ainsi la valeur de leur nœud respectif, est plus difficile à comparer. De ce point de vue, il est donc préférable que les rectangles du placement aient un rapport hauteur/largeur se rapprochant autant que possible de 1. Malgré que notre algorithme n'améliore pas autant le rapport hauteur/largeur des rectangles que celui décrit dans [2], ceci ne représente pas un problème pour nous étant donné que cela n'est pas son but premier et que le rapport hauteur/largeur des rectangles du placement ne représentait jamais un problème dans la visualisation des logiciels testés. L'amélioration qu'apporte notre algorithme à ce niveau est donc amplement satisfaisante pour nos besoins, et nous la considérons plutôt comme un avantage bonus procuré par celui-ci.

3.2.4 Utilisation plus efficace de l'espace supplémentaire

Malgré l'ajout de paquetages virtuels dans la hiérarchie et, du même coup, d'espaces supplémentaires dans le placement pour les distancer de leurs voisins, la figure 3.8 de la section précédente montre que notre nouvel algorithme de placement n'utilise pas significativement plus d'espace que la méthode standard. Au contraire, l'espace vide dans le placement semble mieux réparti, lui donnant ainsi une apparence plus compacte. Les zones d'espaces vides dans le placement sont dues au débalancement de la hiérarchie, conjointement au fait que les dimensions des paquetages sont calculées de façon ascendante. Puisqu'il est pratiquement impossible qu'un logiciel réel ait une hiérarchie parfaitement balancée, il est inévitable que le placement en contienne. La figure 3.9 montre un exemple simple afin d'illustrer cet effet. Toutefois, ces zones d'espaces vides ne sont

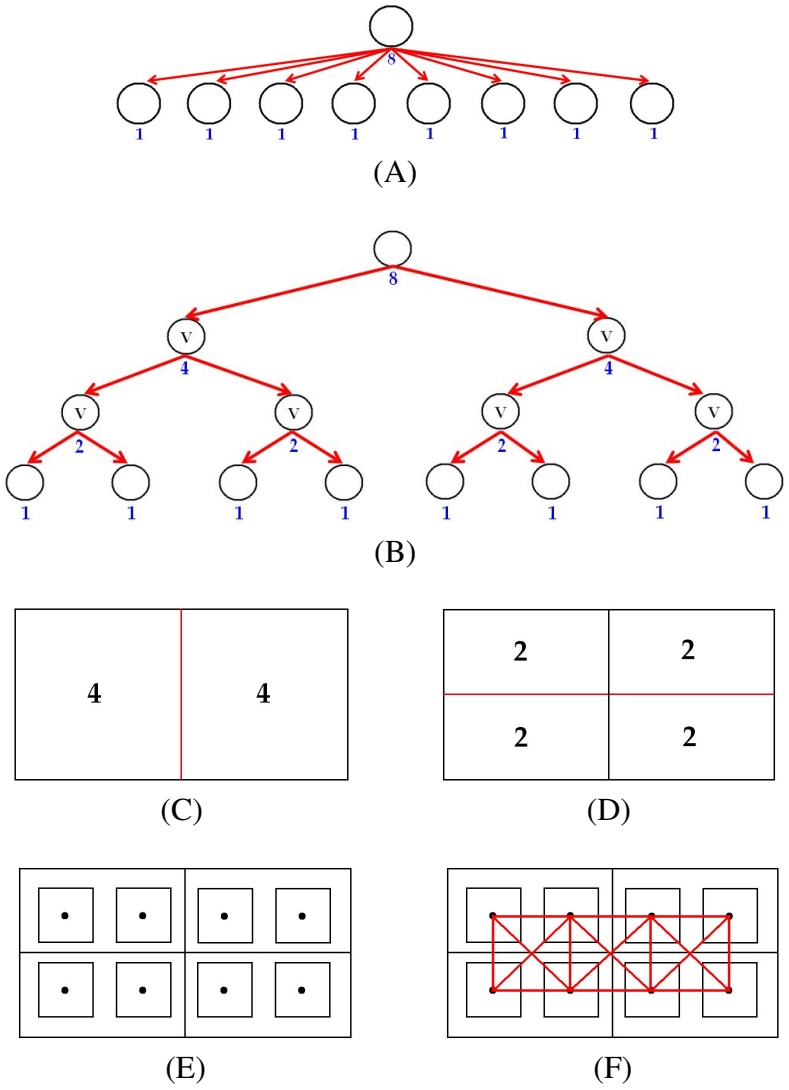
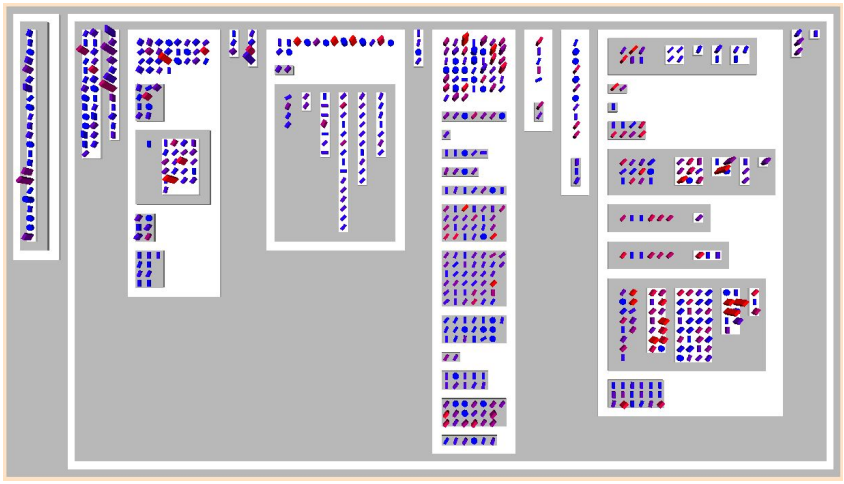
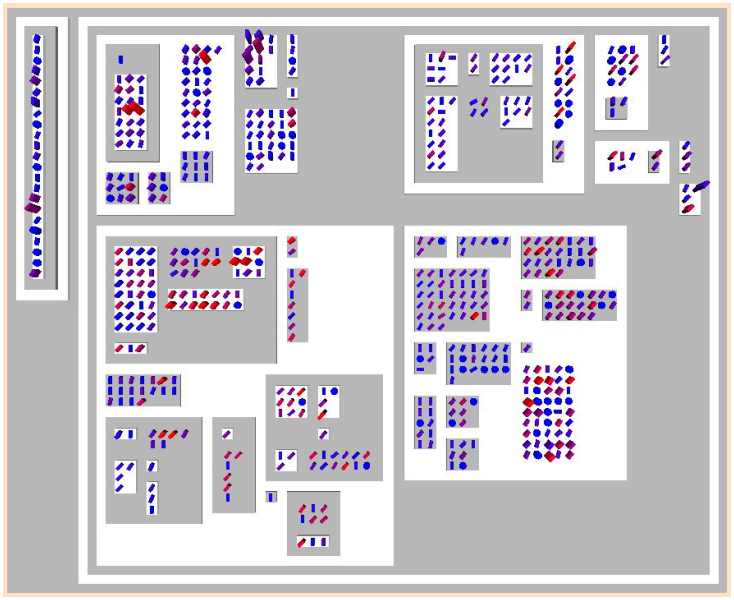


Figure 3.7 – Étapes de l'algorithme d'amélioration du placement des enfants. Reprenons l'exemple donné à la figure 3.6. L'image A) montre la hiérarchie originale : un paquetage avec huit enfants de même poids. L'image B) montre la hiérarchie obtenue après l'application de notre algorithme : un arbre binaire parfaitement balancé contenant six paquetages virtuels (les nœuds avec un « V ») étalés sur deux niveaux. Les images C) et D) montrent les séparations causées par les paquetages virtuels, absentes avec l'algorithme de placement standard. L'image E) montre le placement final obtenu avec notre algorithme. L'image F) montre un sous-ensemble des chemins distinguables existant entre les paquetages, chemins qui sont beaucoup plus nombreux qu'avec l'algorithme de placement standard.



(A)



(B)

Figure 3.8 – Exemple d'amélioration du placement des enfants. L'image A) montre le placement obtenu à la section précédente. L'image B) montre celui qu'on obtient avec notre nouvel algorithme servant à améliorer le placement des enfants.

pas un problème en soi car, comme mentionné dans la section 3.2.1, avoir plus d'espace nous permet de distancer davantage les éléments du placement et ainsi d'afficher plus clairement les liens d'adjacence. Le véritable problème dans ce cas-ci est donc que notre méthode n'utilise pas efficacement cet espace supplémentaire, comme illustré dans la figure 3.10. En se basant sur l'idée que, idéalement, les éléments devraient être aussi espacés que le permet le placement, nous avons modifié notre algorithme afin qu'il puisse mieux utiliser les espaces vides qu'il génère.

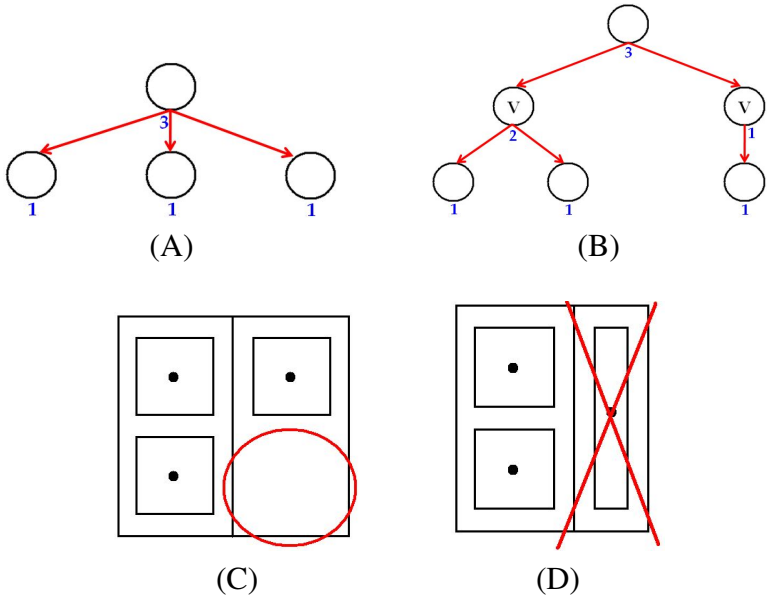
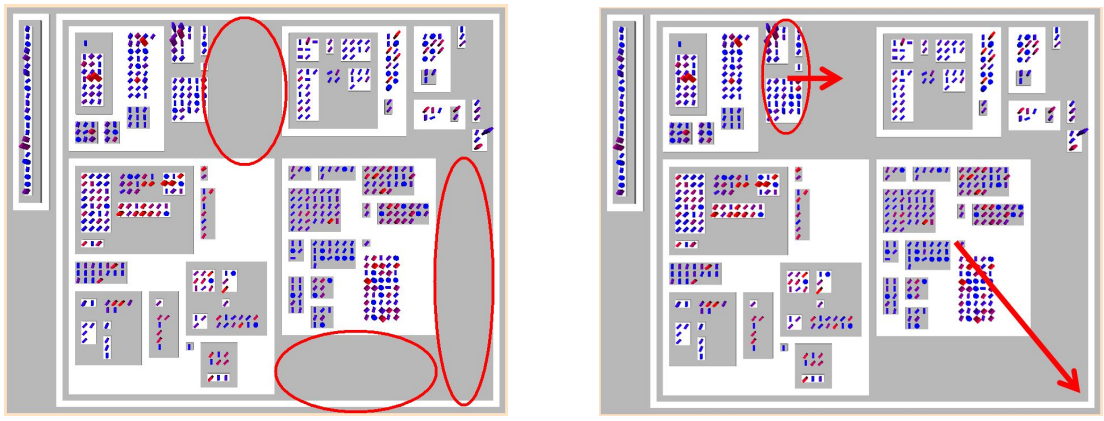


Figure 3.9 – Illustration de la création d'espaces vides. L'image A) montre une hiérarchie dont la racine contient trois enfants de même poids. L'image B) montre la hiérarchie débalancée que l'on obtient après avoir appliqué notre algorithme sur la hiérarchie initiale. La racine possède maintenant deux enfants de poids inégaux. L'image C) montre le placement final qu'on obtient avec notre algorithme. Comme on peut le constater, l'inégalité de poids (et donc de taille) entre les deux paquetages virtuels crée une zone vide dans la partie inférieure droite du placement. Les dimensions des paquetages sont calculées de manière ascendante, signifiant que, lorsque vient le temps de déterminer les dimensions des paquetages du niveau supérieur, on ne peut pas modifier les dimensions des enfants afin d'obtenir un meilleur placement. Comme le montre l'image D), dans cet exemple, cela signifie qu'on ne pourrait pas changer les dimensions du paquetage placé à droite afin qu'il soit plus allongé, ce qui permettrait de combler l'espace vide qu'il y a dans le placement.



(A) (B)

Figure 3.10 – Exemple d'espaces vides dans le placement. L'image A) montre les espaces vides les plus importants dans le placement. L'image B) montre comment on pourrait facilement déplacer le ou les paquetages près de ces zones vides afin de les distancer davantage des autres, et ainsi améliorer la visualisation des liens d'adjacence.

Par défaut notre algorithme, tout comme l'algorithme *Treemap* standard sur lequel il est basé, place systématiquement les paquetages, réels ou virtuels, de gauche à droite en les alignant dans le haut de leur parent. Ceci explique pourquoi les paquetages sont toujours regroupés ensemble dans le coin supérieur gauche de leur parent lorsque celui-ci présente plus d'espace que nécessaire à l'affichage de ses enfants. La solution employée a donc été de modifier notre algorithme afin qu'il aligne les paquetages sur des côtés différents du parent, selon le sens de séparation et selon de nouveaux paramètres introduits dans chaque paquetage.

En résumé, ce que notre algorithme fait est d'éloigner autant que possible chaque paire de paquetages de la séparation qu'il y a entre eux, celle-ci agissant comme une sorte de champ de force qui les repousse dans les deux sens opposés qui lui sont perpendiculaires. La raison expliquant cela est que, généralement, la séparation d'un paquetage est perpendiculaire à son sens le plus long. La distance maximale entre deux paquetages frères peut donc normalement être atteinte lorsqu'on les aligne chacun à un des deux côtés opposés du paquetage parent qui sont parallèles à la séparation (voir la figure 3.11).

La figure 3.12 montre le placement qu'on obtient en utilisant ce nouvel algorithme.

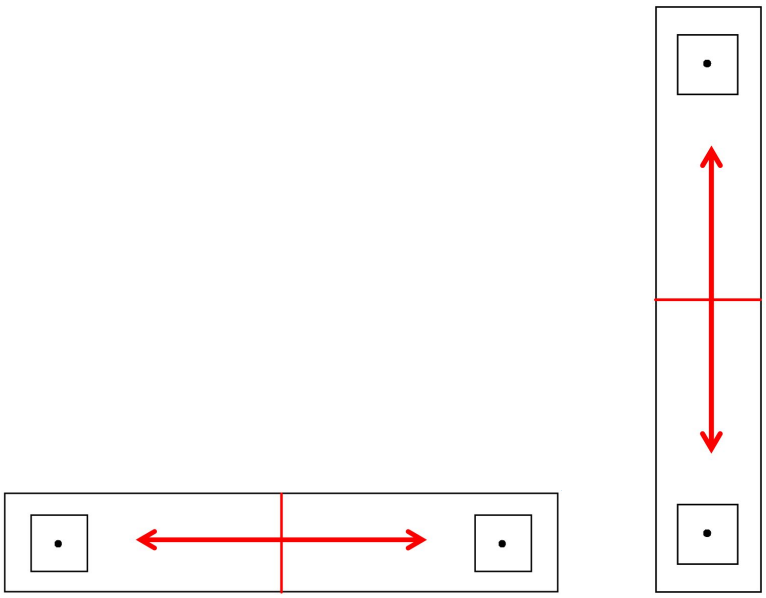


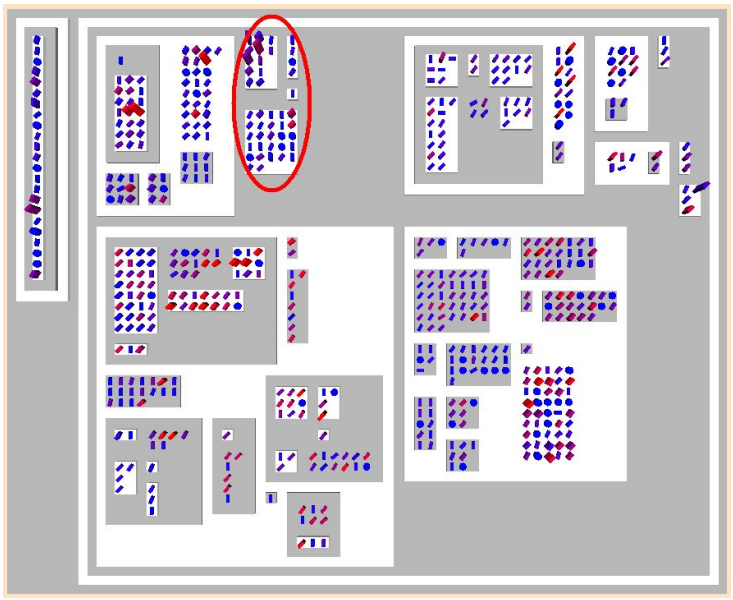
Figure 3.11 – Illustration du principe d'éloignement des paquetages. Les images A) et B) montrent, respectivement, comment placer les paquetages lorsque le parent est séparé verticalement ou horizontalement, de façon à maximiser l'espace qu'il y a entre eux.

Malgré les améliorations apportées par cet algorithme, certaines de ses lacunes sont déjà apparentes dans l'exemple montré. Sa principale faiblesse est qu'il peut éloigner un paquetage de son frère (son voisin immédiat), mais seulement pour le coller à d'autres paquetages qui sont situés dans une autre branche de la hiérarchie.

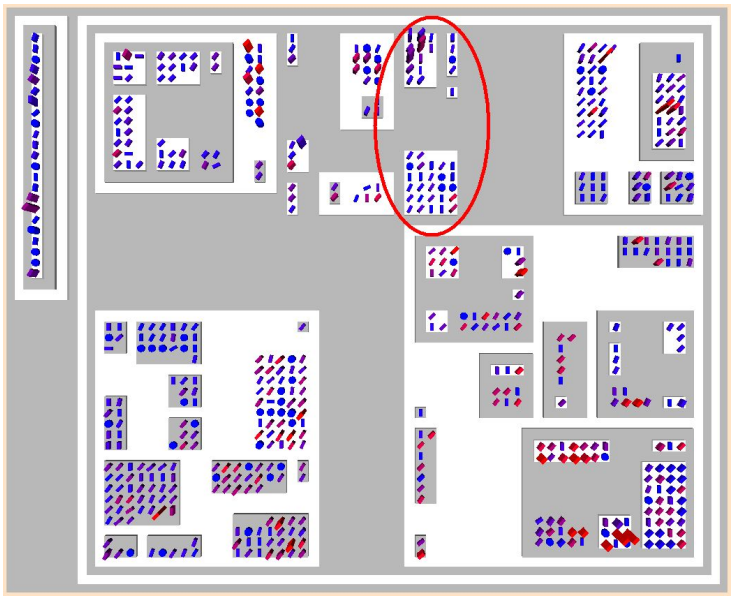
Il est évident qu'il existe différents algorithmes qui permettraient de placer de façon plus uniforme les paquetages dans leur parent, ce qui règlerait la faiblesse que nous venons de mentionner. Par contre, dans notre cas, un des aspects les plus importants du placement est la qualité de la visualisation des liens d'adjacence qu'on peut obtenir avec celui-ci. Dans cette optique, un bon placement ne cherche pas seulement à espacer autant que possible les paquetages en général, mais plus précisément à espacer autant que possible les paquetages ayant une grande quantité de liens entre eux. La section 4.4 décrit un algorithme heuristique qui permet d'améliorer le placement selon cette ligne directrice. Cet algorithme reste toutefois limité par les contraintes qu'impose la hiérarchie, signifiant qu'il serait encore possible d'améliorer davantage le placement avec des algorithmes plus sophistiqués. Néanmoins, notre algorithme de placement basé sur le *Treemap* nous a donné jusqu'à maintenant des résultats amplement satisfaisants avec les logiciels testés, autant pour la visualisation de la hiérarchie de ces logiciels que pour la visualisation des liens d'adjacence entre leurs éléments.

3.3 Placement radial

Comme il le sera montré au chapitre 4, les liens d'adjacence qui seront ajoutés au placement décrit dans la section précédente seront affichés au-dessus de la visualisation de la hiérarchie. Ceci fait en sorte que, lorsque nous visualisons un logiciel avec une vue de dessus, ce qui est nécessaire pour avoir une vue d'ensemble de celui-ci, les liens d'adjacence cachent en partie la représentation hiérarchique. Ceci crée donc une certaine interférence entre l'affichage des liens d'adjacence et l'affichage de la hiérarchie. Il est vrai que, avec notre outil, il est possible de changer le point de vue comme l'on souhaite afin d'afficher avec le moins d'obstructions que possible les régions de la hiérarchie qui nous intéressent. Néanmoins, lorsque nous voulons avoir une vue d'ensemble du



(A)



(B)

Figure 3.12 – Exemple d'utilisation plus efficace de l'espace supplémentaire. L'image A) montre le placement qu'on avait à la fin de la section précédente. L'image B) montre le placement qu'on obtient lorsqu'on essaie d'espacer les paquetages autant que possible. Les paquetages encerclés se sont éloignés de leurs voisins immédiats (le paquetage à leur droite), mais seulement pour se retrouver collés à d'autres paquetages dans le placement.

logiciel, une certaine occultation de la hiérarchie est inévitable, et ce peu importe le point de vue adopté. Le placement radial, introduit dans [10] et repris dans [25], est une façon de représenter la hiérarchie en incluant un espace vide dans le centre du placement, et dans lequel tous les liens d'adjacence sont dessinés sans empiéter sur l'affichage de la hiérarchie, réglant ainsi ce problème d'interférence.

L'idée à la base de ce placement est de représenter les éléments de la hiérarchie avec des arcs (ayant une certaine épaisseur) plutôt qu'avec des rectangles et de les placer sur différents anneaux concentriques collés les uns sur les autres, où chacun de ces anneaux représente un niveau de la hiérarchie. L'anneau le plus externe représente la racine, chaque anneau interne subséquent représente le prochain niveau et les feuilles sont toutes placées sur l'anneau le plus à l'intérieur, c'est-à-dire celui qui est le plus près du centre du placement. L'espace vide délimité par cet anneau, c'est-à-dire l'espace vide au milieu du placement, est ensuite utilisé pour afficher les liens d'adjacence entre les feuilles, faisant ainsi en sorte qu'aucun lien ne passe par-dessus des éléments de la hiérarchie.

Les paquetages sont représentés en divisant radialement chaque anneau en autant de secteurs qu'il y a de paquetages à ce niveau. Chaque secteur occupe une portion de l'anneau qui correspond au ratio entre la taille du paquetage qu'il représente, calculée selon son nombre de descendants, et la taille de son parent. Ceci permet de diviser chaque anneau de façon équitable entre les paquetages qui s'y trouvent. Les relations hiérarchiques sont représentées en plaçant les sous-paquetages d'un paquetage donné à l'intérieur du secteur délimité par celui-ci dans l'anneau inférieur subséquent, secteur qui sera ensuite divisé pour représenter ses sous-paquetages. Ceci crée un effet d'imbrication des secteurs des différents anneaux afin de représenter la hiérarchie. Pour placer les paquetages, on commence donc par placer la racine (qui occupe tout son anneau), puis on divise l'anneau interne subséquent selon le nombre d'enfants qu'elle a et selon leur taille respective, et ainsi de suite de façon récursive pour chacun des enfants. Voir la figure 3.13 pour un exemple en deux dimensions de ce placement.

L'algorithme utilisé dans notre outil pour construire le placement radial diffère un peu de celui proposé dans [10]. L'algorithme proposé dans [10] commence par construire

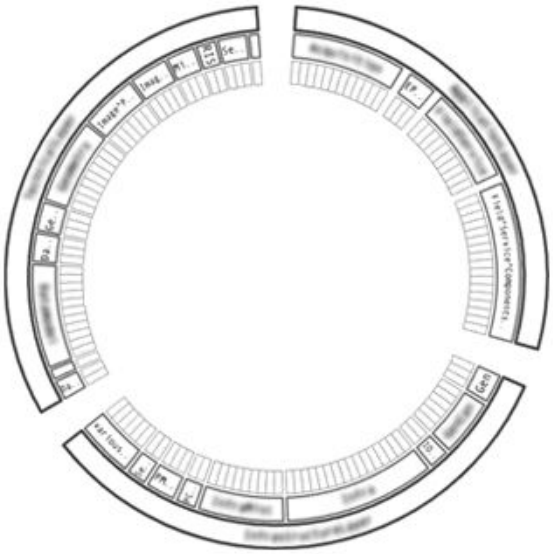


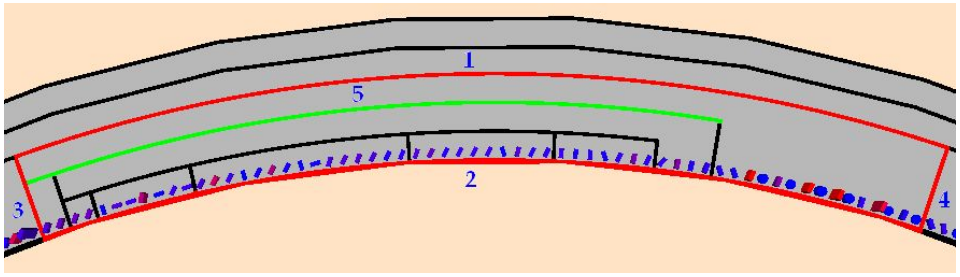
Figure 3.13 – Exemple de placement radial en deux dimensions (figure tirée de [10]).

un arbre radial standard représentant la hiérarchie puis, avec un effet de miroir appliqué sur l'anneau le plus interne, on obtient la hiérarchie externe à partir de laquelle on fait le placement. Notre algorithme procède dans l'ordre inverse, c'est-à-dire que nous commençons par placer la hiérarchie externe et ensuite nous obtenons, par effet de miroir, la hiérarchie interne servant à guider les liens d'adjacence (voir la section 4.3 pour plus de détails sur l'utilité de cette hiérarchie interne dans la création des liens d'adjacence). Malgré tout, exceptées quelques petites différences, le placement qu'on obtient avec notre algorithme est essentiellement identique à celui que produit le leur.

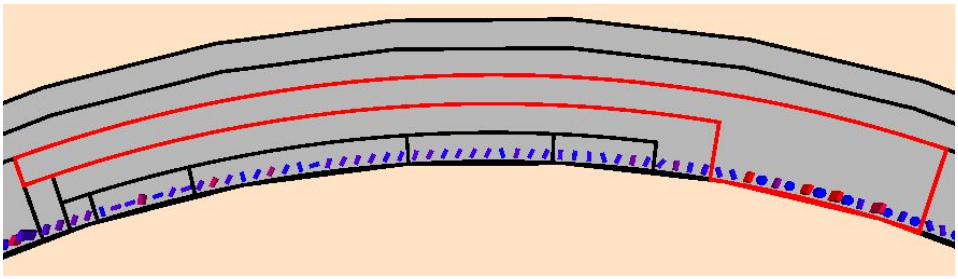
Dans notre outil, les paquetages sont affichés en délimitant avec une bordure le secteur les représentant. Les classes sont toutes placées dans la partie inférieure de leur parent, sur l'anneau le plus interne du placement. Tous les paquetages qui ne sont pas au plus bas niveau de la hiérarchie sont donc prolongés jusqu'à cet anneau, de sorte que leurs classes puissent y être placées. Ainsi, un paquetage englobe visuellement ses enfants non seulement en terme d'angle, mais aussi en terme d'épaisseur lorsque celui-ci est fermé (détails et exemples sur ce sujet dans la section 4.5.1). La figure 3.14 montre comment un paquetage est affiché dans notre placement radial et décrit les différentes parties qui le composent. La figure 3.15 montre un exemple de placement obtenu avec

notre algorithme.

L'exemple montré dans la figure 3.15 met en relief la principale faiblesse de ce placement, c'est-à-dire l'imposant espace nécessaire pour l'afficher, et ce malgré la taille relativement petite du logiciel visualisé. Ceci s'explique facilement par le fait que, étant donné que toutes les classes sont placées sur l'anneau le plus interne, la circonférence du placement dépend directement du nombre de classes que contient le logiciel. Par exemple, en supposant que chaque classe occupe une unité d'espace, un logiciel comportant 3000 classes aura un anneau interne de 3000 unités de circonférence, ce qui correspond à un placement ayant un rayon interne d'environ 477 unités. Un tel logiciel serait bien trop grand pour qu'on puisse en avoir une vue d'ensemble tout en étant capable de voir assez clairement les détails de la hiérarchie, c'est-à-dire les classes et les bordures délimitant les paquetages. La solution que nous employons pour limiter cette faiblesse est d'afficher les classes sur plusieurs rangées à l'intérieur de leur parent. En effet, chaque paquetage a ce qu'on pourrait appeler une « devanture », c'est-à-dire le nombre de classes qu'il doit placer sur l'anneau interne du placement, et qui ainsi détermine la taille, en terme de longueur d'arc, nécessaire à son affichage. La circonférence de l'anneau interne du placement étant obtenue en additionnant la devanture de tous les paquetages, cela signifie que si nous réduisons la taille de ces devantures, alors la circonférence du placement sera réduite aussi. Lorsque toutes les classes sont placées sur une seule rangée, la devanture de chaque paquetage correspond au nombre de classes qu'il contient. Par contre, si nous les plaçons sur plusieurs rangées, l'une derrière l'autre, alors la taille de chaque devanture se retrouve, dans le meilleur des cas, divisée par le nombre de rangées utilisées pour placer ces classes, de ce fait divisant aussi la circonférence de l'anneau interne par le nombre de rangées. Par contre, pour qu'une telle situation idéale se présente, il faut que tous les paquetages du logiciel aient un nombre de classes qui soit un multiple du nombre de rangées voulues, ce qui est à toute fin pratique impossible avec des logiciels réels (voir la figure 3.16 pour un exemple de ce phénomène). Malgré tout, cette façon de procéder a permis de réduire considérablement la taille du placement des logiciels visualisés avec notre outil. La figure 3.17 montre jusqu'à quel point placer les classes sur plusieurs rangées réduit la taille du placement présenté à la



(A)



(B)

Figure 3.14 – Façon d'afficher un paquetage dans le placement radial. Dans l'image A), nous avons mis en rouge les bordures qui limitent le paquetage (toutes les bordures sont noires dans l'affichage réel). Les bordures numéro 1 et 2 correspondent respectivement aux bordures externe et interne du paquetage, tandis que les bordures numéro 3 et 4 correspondent aux limites de l'angle qu'il occupe dans le placement. La bordure numéro 5, en vert, est obtenue par l'agrégation de l'affichage des bordures externes de ses sous-paquetages. Celle-ci, conjointement avec sa bordure externe, permet de dessiner l'anneau concentrique sur lequel il est placé. Ceci nous permet d'obtenir visuellement une série d'anneaux concentriques, comme mentionné plus haut. En ne mettant aucune séparation entre les classes du paquetage, qui sont placées sur l'anneau le plus interne, et la section d'anneau le représentant, cela permet à l'utilisateur, par continuité, de facilement associer les classes à leur parent. L'image B) montre, à l'aide de bordures rouges, la forme du paquetage perçue par l'utilisateur, forme qui permet de lui associer assez facilement le bon groupe de classes.

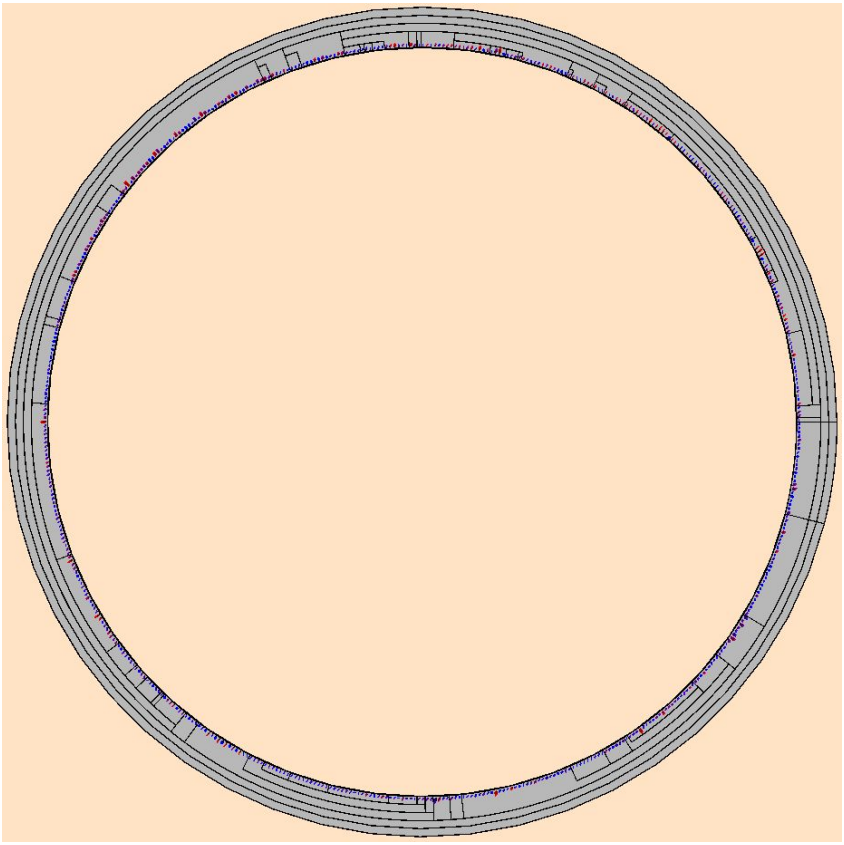
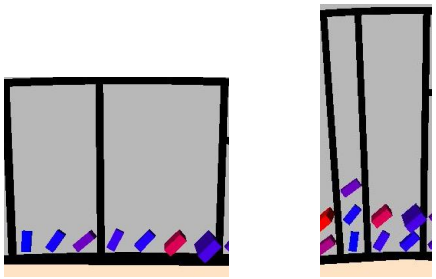


Figure 3.15 – Exemple de placement radial standard. Le logiciel représenté dans cet exemple, *JHotDraw*, contient 587 classes.

figure 3.15. Le seul inconvénient de cette méthode est qu'elle crée un peu d'interférence entre la visualisation des liens d'adjacence et la visualisation de la hiérarchie, ce qui est contraire au principe qui est à la base même de ce type de placement. Par contre, les différentes rangées de classes étant placées le plus près possible de l'anneau interne du placement, l'interférence que cela ajoute à la visualisation reste limitée et est localisée au niveau des classes. En fin de compte, il revient à l'utilisateur de choisir un nombre de rangées qui offre un bon compromis entre l'interférence créée et la réduction de la taille du placement obtenue.



(A) (B)

Figure 3.16 – Illustration de la réduction de la devanture des paquetages. L'image A) montre deux paquetages contenant respectivement trois et quatre classes et où celles-ci sont placées sur une seule rangée. L'image B) montre le résultat obtenu pour ces deux paquetages lorsque nous plaçons les classes sur trois rangées. Le paquetage contenant trois classes, ayant un nombre de classes qui est un multiple du nombre de rangées, a diminué sa devanture par trois. Par contre, le paquetage contenant quatre classes ne l'a réduite que de moitié, étant donné que placer une seule classe par rangée, sur trois rangées, n'aurait évidemment pas suffit à placer toutes ses classes. Il fallait donc, dans ce cas-ci, en placer deux par rangée, sur deux rangées seulement.

Une autre faiblesse de ce placement, lorsque nous le comparons à notre placement *Treemap* présenté à la section 3.2.1, est qu'il est moins efficace pour représenter les relations hiérarchiques. Par exemple, il est beaucoup plus difficile de voir où commence et où se termine un paquetage qui occupe une grande partie de son anneau, surtout si celui-ci est de haut niveau. Il suffit d'observer les deux paquetages du plus haut niveau dans la figure 3.16 pour s'en rendre compte. Afin de remédier à cela, nous avons appliqué à notre placement radial les mêmes modifications que celles que nous avons appliquées

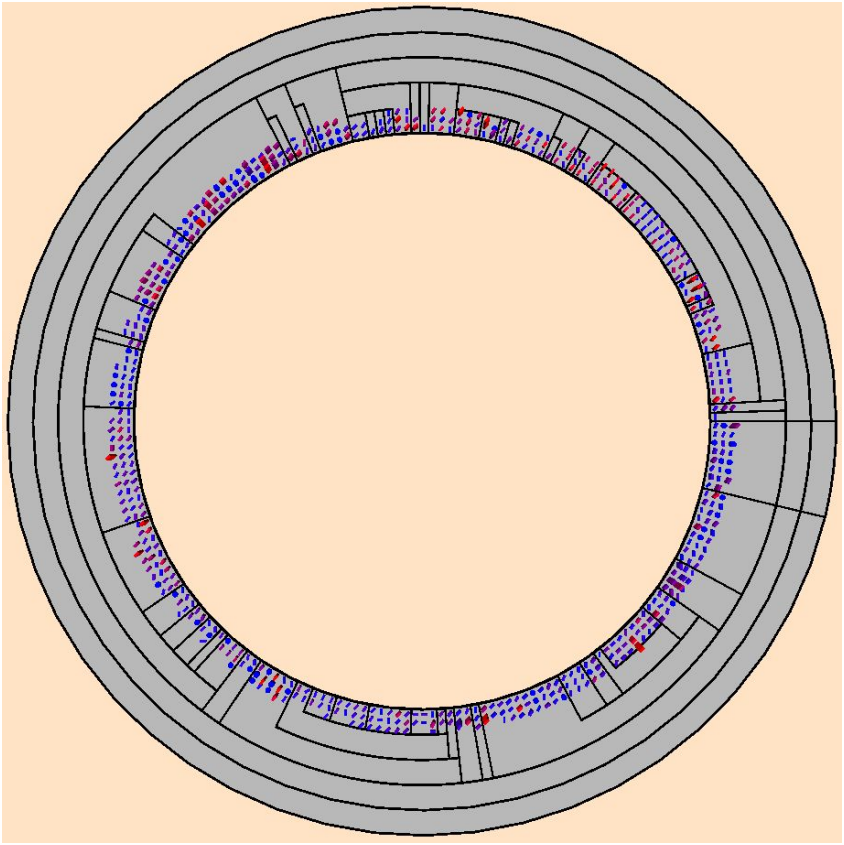


Figure 3.17 – Exemple d'un placement radial avec plusieurs rangées pour placer ses classes. Ce placement a une taille de loin inférieure au placement radial standard.

au placement *Treemap* et qui sont décrites dans la section 3.2.2, c'est-à-dire l'alternance de couleur entre les niveaux, l'empilement des paquetages et l'ajout d'espaces (de bordures) entre eux. Ces modifications procurent au placement radial les mêmes avantages qu'elles procuraient au placement *Treemap* concernant l'amélioration de la visualisation de la hiérarchie et des liens d'adjacence. La seule modification à notre algorithme de placement est qu'il doit prendre en considération la contribution des espaces à la circonférence de l'anneau interne. La figure 3.18 montre le même placement radial présenté précédemment, mais après y avoir appliqué les modifications susmentionnées.

3.4 Placement de type Colisée

Comme présenté à la section précédente, la modification apportée au placement radial standard afin qu'il puisse afficher les classes de chaque paquetage sur plus d'une rangée a permis de rendre sa taille raisonnable pour le logiciel montré en exemple, surtout comparée à la taille qu'il avait avant d'y apporter cette modification. Par contre, celle-ci s'est avérée insuffisante pour que le placement ait une taille raisonnable avec certains logiciels plus imposants que nous avons testés, qui eux illustrent mieux le logiciel moyen auquel notre outil pourrait être confronté. La cause principale à cela est que le placement a l'obligation de placer les devantures des paquetages côte à côte sur l'anneau interne du placement, sans créer d'intersections entre eux. L'imposant espace d'affichage qu'il occupe étant pour nous l'aspect le plus problématique que présente ce type de placement, nous avons décidé de modifier l'algorithme en lui enlevant l'obligation susmentionnée de sorte qu'il puisse disposer les classes différemment dans le placement. Plus précisément, ce nouvel algorithme place les classes directement sur l'anneau représentant leur niveau dans la hiérarchie, sans prolonger les paquetages jusqu'à l'anneau interne du placement et sans obliger les groupes de classes à être côte à côte en terme d'angle, ce qui évite d'agrandir inutilement certains paquetages. Même si cette façon de placer les classes augmente l'interférence entre la représentation de la hiérarchie et les liens d'adjacence, celle-ci reste limitée et ne pose pas vraiment de problème en pratique. De toute façon, la réduction de l'espace d'affichage qu'offre ce nouveau placement, comparé au

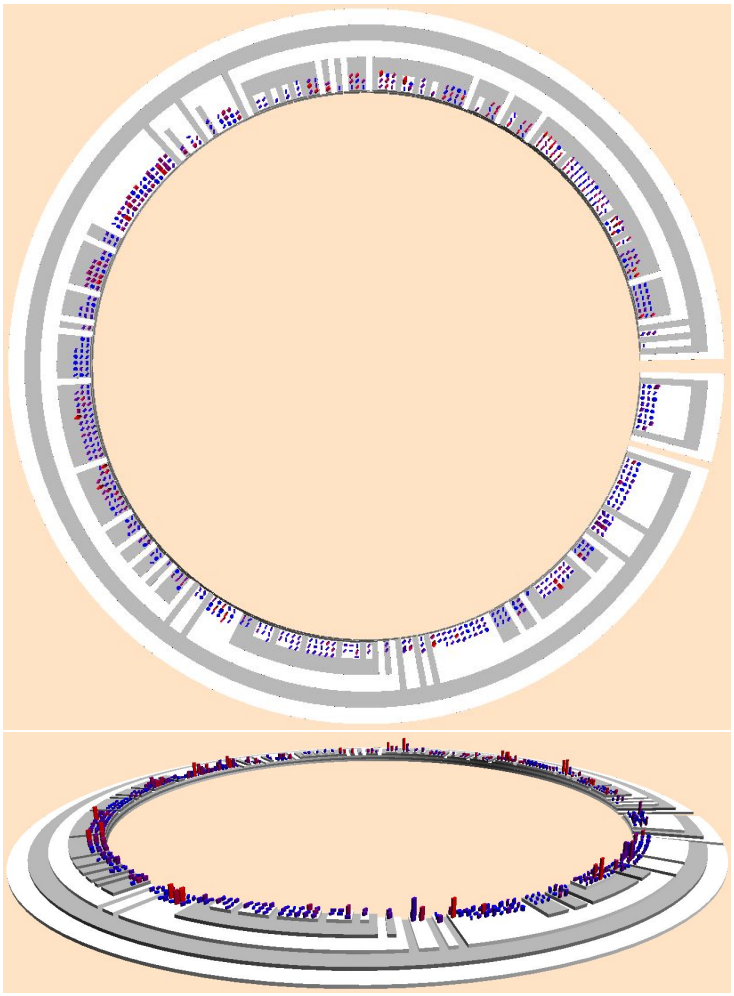


Figure 3.18 – Exemple du placement radial utilisé dans notre outil. Ce placement correspond au placement radial montré précédemment, mais auquel nous avons ajouté les mêmes modifications que celles apportées au placement *Treemap* standard, c'est-à-dire des espaces entre les éléments, l'alternance de couleur entre les niveaux et l'empilement des paquetages.

placement radial, compense largement la légère augmentation d'interférence qu'il crée. En voyant les résultats obtenus par cette façon de procéder, une métaphore s'est imposée à nous : celle du Colisée de Rome.

En effet, les classes du placement sont disposées de façon similaire aux spectateurs dans le Colisée, donc si, à l'instar de celui-ci, nous disposions les paquetages en gradins, cela permettrait à chaque classe d'avoir une bonne visibilité du reste du logiciel, tout comme ce principe permet à chaque spectateur d'avoir une bonne vue sur le reste du Colisée et sur l'action qui s'y déroule. Cette façon de procéder améliorerait la visualisation des liens d'adjacence en évitant plus naturellement l'intersection entre ceux-ci et la hiérarchie, et en faisant en sorte que les liens entre des éléments plus hauts dans la hiérarchie passent plus facilement au-dessus des liens entre des éléments moins hauts. De plus, la visualisation de la hiérarchie s'en trouverait améliorée aussi, notamment par l'inclusion plus naturelle des paquetages entre eux et par la plus grande facilité pour un utilisateur de déterminer le niveau d'un groupe de classes dans la hiérarchie ainsi que le paquetage auquel il appartient.

L'algorithme utilisé pour créer ce type de placement est essentiellement le même que celui utilisé pour le placement radial, excepté évidemment pour les particularités qui le distinguent de ce dernier. Premièrement, les paquetages ne sont pas prolongés systématiquement jusqu'à l'anneau interne du placement, mais seulement jusqu'à leur plus profond descendant. Ensuite, comme déjà mentionné, les classes sont placées sur leur paquetage parent, au niveau de l'anneau interne correspondant à son niveau dans la hiérarchie. La zone où les placer correspond à la surface complète du secteur qu'occupe leur paquetage parent sur son anneau, plutôt qu'à une partie de celui-ci. Cette surface pouvant être beaucoup plus grande que nécessaire, cela permet, lorsque c'est le cas, d'espacer davantage les classes comparé au placement radial, où celles-ci sont toujours placées côte à côte.

L'algorithme d'affichage procède aussi de la même manière que pour le placement radial, excepté pour deux différences fondamentales. Premièrement, il associe des hauteurs décroissantes aux paquetages, en commençant par la racine, afin de les disposer en gradin, ce qui est essentiel pour que le placement ressemble au Colisée duquel il s'ins-

pire. Deuxièmement, il affiche les paquetages ouverts de sorte qu'ils occupent seulement le secteur qu'ils utilisent sur l'anneau leur correspondant, sans les prolonger jusqu'à leur plus profond descendant, sans quoi ils cacheraient leurs sous-paquetages. Les principes d'alternance de couleur entre les niveaux et d'espacement des éléments, décrits à la section 3.2.2 pour le placement *Treemap* et réutilisés pour le placement radial, ont aussi été incorporés tels quels dans le placement Colisée. Il est toutefois à noter que l'alternance de couleur fonctionne moins bien dans ce cas-ci pour ce qui est de regrouper facilement les paquetages frères. Ceci s'explique par l'absence d'une couleur de fond commune à tous les paquetages frères, couleur correspondant à celle de leur parent et alternant entre les niveaux. Il faut donc se fier aux espaces qu'il y a entre les éléments du niveau supérieur pour déterminer les groupes de paquetages frères à un niveau donné du placement, ce qui est moins efficace que la façon de faire dans les autres types de placements. La figure 3.19 montre le résultat obtenu avec le placement Colisée, pour le même logiciel que celui montré en exemple à la figure 3.18.

Comme le montre la figure 3.19, pour un même nombre de rangées et une même quantité d'espace entre les éléments, le placement Colisée utilise beaucoup moins d'espace d'affichage que le placement radial, tout en gardant intacts les principes fondamentaux à ce genre de placement circulaire. Par contre, disposer les classes sur les différents anneaux concentriques augmente l'interférence entre les liens d'adjacence et la hiérarchie, ce que nous voulions éviter autant que possible avec le placement radial. Pour alléger ce problème, lorsqu'un paquetage offre plus d'espace que nécessaire pour y placer ses classes, au lieu de les disperser uniformément à travers tout le paquetage parent, nous allons plutôt les regrouper dans une de ses extrémités. Le placement offre ainsi une façon de placer les classes qui se rapproche davantage du décalage entre les groupes de classes, en terme de l'angle qu'ils occupent, que l'on retrouve dans le placement radial. La figure 3.20 montre un exemple de cette nouvelle façon de placer les classes.

Au final, en ce qui concerne l'utilisation efficace de l'espace d'affichage, la réduction de l'interférence entre l'affichage des liens d'adjacence et la représentation de la hiérarchie, le placement Colisée offre une solution mitoyenne entre le placement *Treemap* et le placement radial.

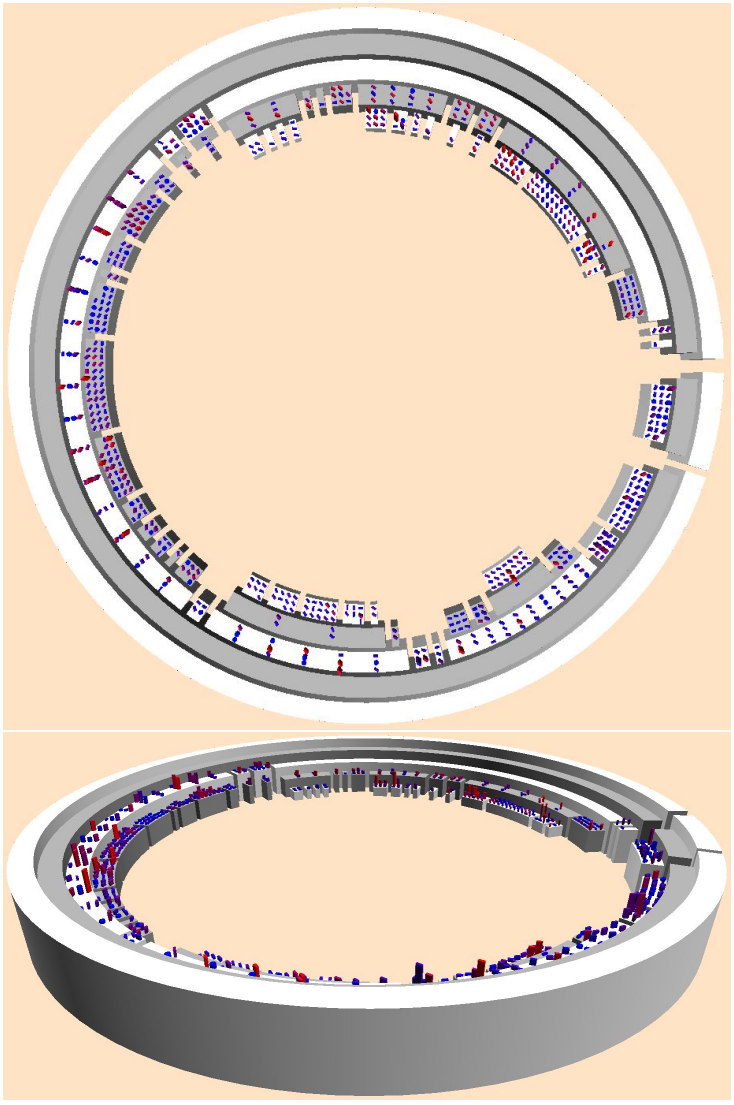


Figure 3.19 – Exemple du placement Colisée.

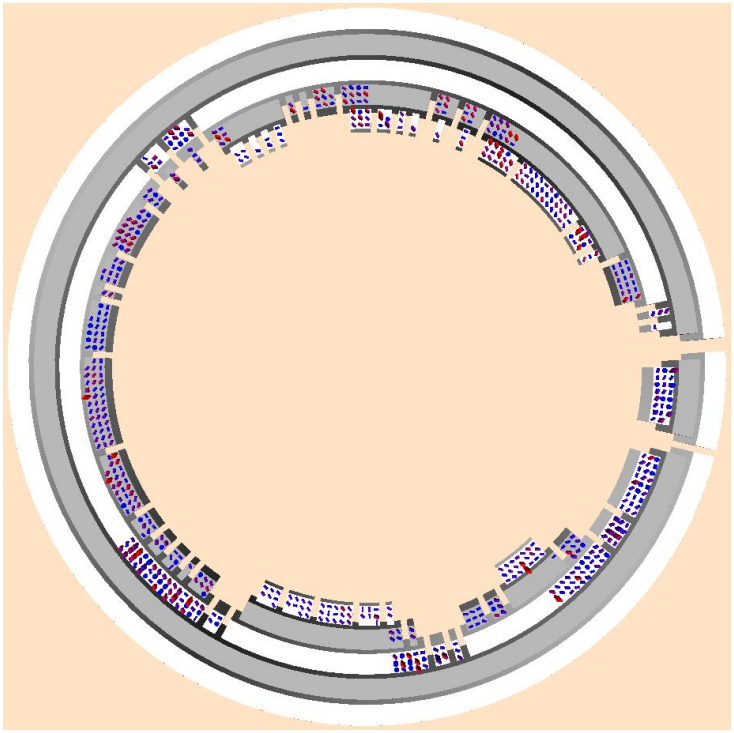


Figure 3.20 – Exemple de la nouvelle façon de placer les classes dans le placement Colisée.

CHAPITRE 4

VISUALISATION DES LIENS D'ADJACENCE

Le chapitre précédent a montré diverses façons de disposer les éléments dans l'espace afin de présenter les relations hiérarchiques entre ceux-ci de la façon la plus claire et la moins encombrante que possible. Ceci nous a donné différents types de placements, chacun avec ses forces et ses faiblesses, nous permettant de visualiser l'entièreté de la structure hiérarchique du logiciel sous analyse et des éléments le composant. Comme expliqué dans l'introduction du présent mémoire, en plus de présenter de façon visuelle la structure hiérarchique du logiciel, nous nous intéressons aussi à présenter simultanément les liens d'adjacence existant entre les éléments de celui-ci. Les données utilisées pour déterminer les liens existant entre les éléments et, ultimement, ce qu'ils représentent comme information, par exemple les liens d'invocation ou les liens d'héritage, nous importent peu. Le travail de ce présent mémoire s'intéresse avant tout à représenter visuellement ces liens de la façon la plus compréhensible que possible, et ce peu importe le type d'information qu'ils représentent. Nous prenons donc pour acquis que toutes les données relatives aux liens entre les éléments ont déjà été calculées, c'est-à-dire que nous connaissons déjà les liens existant entre les classes du logiciel. Dans notre système, un lien ne peut exister qu'entre exactement deux entités : l'entité de départ et l'entité d'arrivée (la cible). La tâche qu'il nous reste à faire consiste donc à trouver une représentation visuelle convenable à tous ces liens et de l'ajouter à la visualisation des liens hiérarchiques déjà existante, de façon à ce qu'on puisse visualiser ces deux ensembles de données simultanément avec le moins d'interférence possible. Le présent chapitre explique notre démarche pour accomplir cette tâche. La section 4.1 explique les principaux problèmes auxquels nous sommes confrontés lorsqu'on veut afficher des liens d'adjacence. La section 4.2 décrit la métaphore sur laquelle se fonde la solution utilisée pour résoudre ces problèmes. La section 4.3 décrit les méthodes utilisées pour créer et représenter visuellement les liens d'adjacence ainsi que la façon dont nous les incorporons dans la visualisation déjà existante. La section 4.4 présente une méthode heuristique

permettant d'optimiser le placement des éléments afin qu'il offre une meilleure visualisation des liens d'adjacence. Finalement, la section 4.5 décrit les différents mécanismes d'interaction dynamique que notre système offre à l'utilisateur pour réduire l'encombrement visuel et ainsi pour lui permettre d'aller chercher les informations qui l'intéressent. Comme dans le chapitre 3, tous les exemples de notre visualisation qui seront présentés dans ce chapitre utilisent, sauf avis contraire, le logiciel *JHotDraw*, qui est composé de 583 classes réparties dans 67 paquetages, qui eux sont étalés sur 6 niveaux de profondeur (incluant la racine).

4.1 Problématique de la représentation des liens d'adjacence

Comme il l'a été montré à la section 2.4, il existe déjà un mécanisme d'affichage des liens dans VERSO. Celui-ci fonctionne sur demande, c'est-à-dire que l'utilisateur doit d'abord sélectionner une classe et, ensuite, le système affiche les liens d'adjacence reliés à celle-ci, ne permettant donc pas de visualiser simultanément l'ensemble des liens d'adjacence du logiciel. Cette façon de procéder s'avère ainsi largement insuffisante pour une analyse plus approfondie des liens d'adjacence d'un logiciel, car il est important d'en avoir une vue d'ensemble, et donc de pouvoir tous les visualiser simultanément, si nous voulons observer, par exemple, les tendances dans les liens ou les zones à forte concentration de liens entrants ou sortants. De plus, dans l'état qu'il est actuellement, ce mécanisme n'affiche pas le sens des liens d'adjacence, le rendant ainsi encore plus limitatif. Ce mécanisme étant inadéquat pour les tâches d'analyses envisagées, il nous faut donc incorporer à VERSO une nouvelle façon de visualiser les liens d'adjacence. La première question à se poser dans le processus d'élaboration de cette nouvelle méthode est si nous voulons représenter les liens de façon implicite, c'est-à-dire sans utiliser d'entité graphique pour représenter visuellement les liens, ou à l'inverse les représenter de façon explicite.

La méthode de représentation implicite utilisée par le mécanisme décrit ci-haut, qui utilise la saturation des classes pour montrer lesquelles sont liées, est tout de suite à écarter car, comme déjà mentionné, on ne pourra jamais réussir à afficher tous les liens

d'adjacence en procédant de cette façon. Les méthodes de représentation par inclusion décrites à la section 2.1.2, c'est-à-dire celles qui utilisent le positionnement relatif des éléments les uns par rapport aux autres afin de représenter les liens qui existent entre ceux-ci, ne peuvent être utilisées non plus, et ce pour deux raisons évidentes. Premièrement, la position des éléments étant déjà utilisée pour représenter les liens hiérarchiques existant entre ceux-ci, il est impossible de s'en servir pour représenter un autre type de liens. Deuxièmement, même si cet attribut visuel était disponible, comme expliqué à la section 2.1.2 il est en pratique inutilisable pour les liens d'adjacence.

La meilleure façon de procéder est donc d'utiliser une représentation graphique explicite de type nœuds-liens pour afficher les liens d'adjacence, surtout que, les liens hiérarchiques étant représentés de façon implicite, ceci évite de créer trop d'interférence entre ces deux représentations. La représentation nœuds-liens la plus simple que l'on peut imaginer pour représenter les liens d'adjacence est tout simplement de tracer une ligne droite entre chaque paire d'éléments liés. Le sens du lien est indiqué avec un dégradé de couleurs allant du noir (point de départ) au jaune (point d'arrivée). Le choix des couleurs de départ et d'arrivée est arbitraire, mais il est quand même important de les choisir de façon à ce qu'elles se démarquent suffisamment des couleurs utilisées dans la représentation de la hiérarchie. La figure 4.1 montre un exemple du résultat obtenu en utilisant cette méthode dans VERSO.

Comme nous le montre de façon éloquente cette figure, même avec un logiciel relativement petit (dans ce cas-ci, 587 classes et 1 607 liens), la représentation utilisant des lignes droites comporte beaucoup d'encombrement visuel, ce qui limite les tâches d'analyse que l'on peut faire sur les liens d'adjacence avec cette visualisation. Les seules informations utiles que l'on peut en tirer sont les points sensibles, c'est-à-dire les zones à forte concentration de liens entrants et/ou sortants, et les convergences importantes de liens qui partent d'une zone élargie pour aller vers un point plus précis du logiciel. Une tâche importante que nous voulons pouvoir être capable de réaliser avec notre système est de visualiser les tendances générales dans les liens d'adjacence, dans le but d'être en mesure de voir clairement les liens qui existent entre les différentes parties du logiciel. Ceci nous permettrait de pouvoir déduire le schéma des collaborations existant entre les

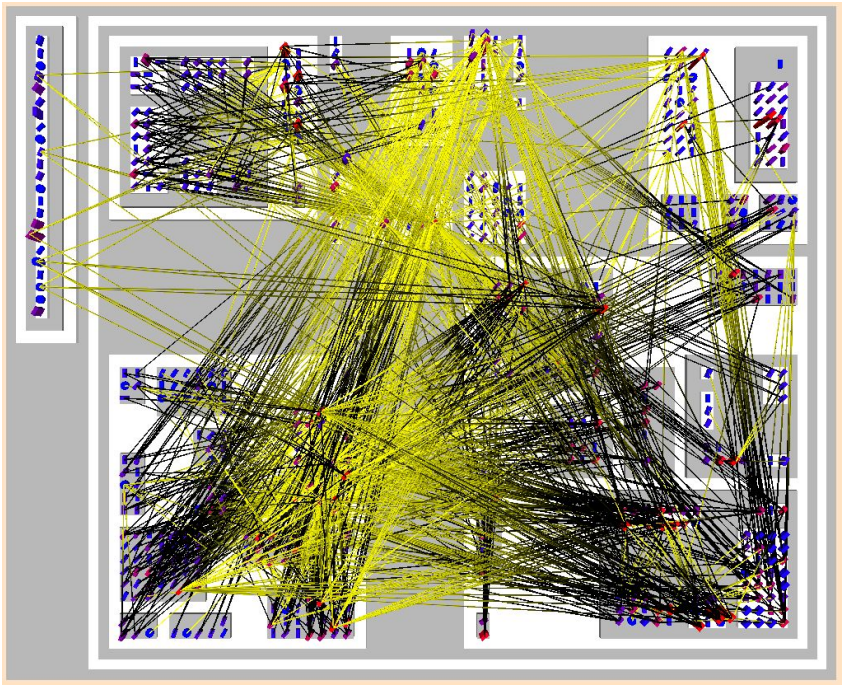


Figure 4.1 – Exemple de représentation des liens d'adjacence avec des lignes droites.

paquetages du logiciel, ce qui est impossible à réaliser avec une visualisation qui utilise des lignes droites. La figure 4.2 montre à quel point le problème d'encombrement visuel s'accentue lorsque le nombre de classes et de liens augmente.

Les deux exemples précédents montrent qu'il est impératif de diminuer de façon significative l'encombrement visuel si nous voulons que notre visualisation soit en mesure de faire ressortir de l'information de plus haut niveau, telle les tendances dans les liens. Une façon de faire dans la vie courante nous procure une excellente métaphore pour résoudre ce problème : l'installation de câbles dans un immeuble.

4.2 Métaphore de l'installation de câbles

Prenons le problème d'installation de câbles réseaux dans un immeuble, en le regardant sous l'angle où l'on veut relier tous les appareils réseaux présents dans l'immeuble au réseau Internet global. La solution qu'on applique en général consiste à faire sortir les câbles réseaux d'une pièce où se trouvent des appareils que l'on veut connecter, en

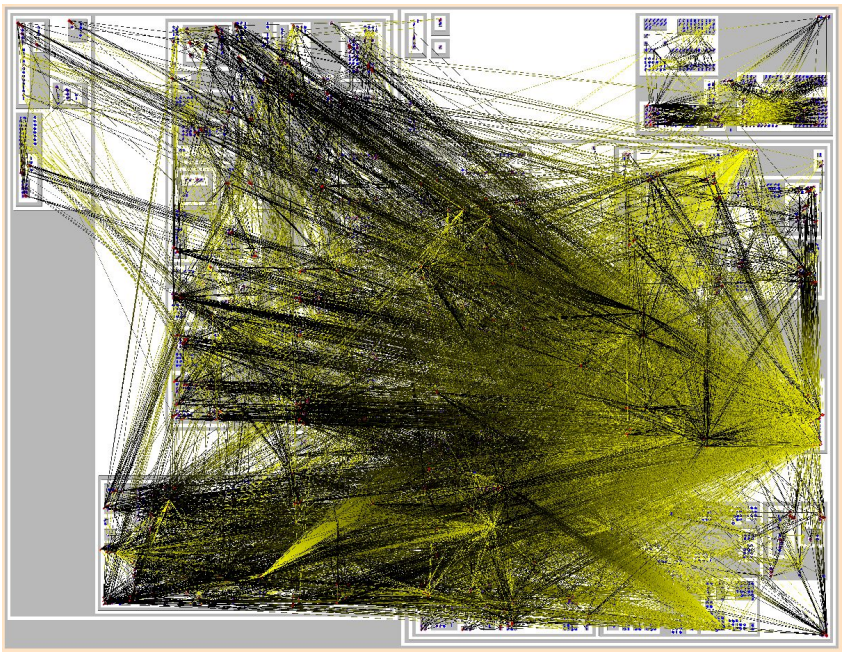


Figure 4.2 – Exemple de représentation avec des lignes droites appliquée à un plus grand logiciel, dans ce cas-ci *Azureus*.

les rassemblant afin qu'ils forment un plus gros câble, qui lui sera ensuite sorti de la pièce par un point précis avant d'aller rejoindre un appareil de distribution (commutateur, routeur, etc.). Nous faisons de même avec les autres pièces, raccordant leur câble de sortie au même appareil de distribution, permettant ainsi de centraliser la sortie des câbles de l'étage en un seul point, à l'instar de ce que nous faisons dans chaque pièce. Le même procédé est donc répété pour faire sortir de l'étage l'ensemble des câbles de toutes les pièces présentes sur celui-ci, puis, en raccordant les sorties de tous les étages à un autre appareil de distribution, de les faire sortir de l'immeuble, avant d'aller finalement rejoindre le réseau Internet global.

Si nous regardons ce procédé dans l'autre sens, prenant par exemple la façon de faire pour amener l'alimentation électrique dans un immeuble, alors le procédé consiste à faire entrer l'alimentation électrique, depuis une centrale externe, par le point d'entrée de l'immeuble. Ce point d'entrée correspond au boîtier de distribution principal de l'immeuble, et l'électricité y est amenée de la centrale à l'aide d'un câble de grande taille, c'est-à-dire d'un câble à haut débit. Ce boîtier de distribution va ensuite s'occuper de

subdiviser l'électricité entrante dans plusieurs câbles à plus petit débit, où chacun d'eux est relié au boîtier de distribution d'un étage de l'édifice, permettant ainsi de desservir tous les étages en électricité. Finalement, on répète le même procédé pour amener l'électricité dans les pièces de chaque étage, et ensuite pour alimenter les appareils électriques présents dans chaque pièce.

Il est évident que, en général, les installations réelles de câbles sont beaucoup plus complexes que ce que nous venons de décrire. Par exemple, il peut y avoir plusieurs appareils de distribution pour l'immeuble ou pour chaque étage, offrant ainsi différents points d'entrée par où amener l'alimentation électrique ou les données réseaux, ou à l'inverse avoir un appareil de distribution qui dessert plusieurs étages. Il peut y avoir aussi différents points de sortie par lesquels faire sortir les câbles d'une pièce, les regroupant ainsi en plusieurs câbles de sortie différents. Malgré tout, afin que la métaphore utilisée pour guider notre visualisation reste simple et permette ainsi de produire une visualisation efficace pour montrer l'information voulue à un utilisateur, nous utilisons comme métaphore une vision idéalisée de la situation réelle. L'essentiel de cette métaphore, dépendamment du sens dans lequel on la considère, se résume donc ainsi :

- Les entrées d'un certain niveau (immeuble, étage ou pièce) y entrent par un seul endroit, puis, une fois le point d'entrée passé, sont séparées afin d'aller desservir tous les sous-niveaux du niveau courant.

- Les sorties d'un certain niveau (immeuble, étage ou pièce) sont rassemblées afin d'en sortir par un seul endroit avant d'aller rejoindre le niveau supérieur.

Transposant la métaphore décrite ci-haut dans la visualisation qu'offre notre système de la hiérarchie d'un logiciel, comme décrit dans le chapitre 3, le paquetage racine du système devient alors l'immeuble, les paquetages descendant de cette racine deviennent les subdivisions de l'immeuble, c'est-à-dire les étages et les pièces, les classes deviennent les appareils à brancher et les liens d'adjacence deviennent les câbles. Pour respecter les deux principes essentiels de la métaphore, nous devons définir pour chaque paquetage un point par lequel faire passer les liens entrant et sortant de celui-ci. Les

liens sortant d'un paquetage seront ainsi rassemblés en son point d'entrée-sortie avant de continuer plus haut dans la hiérarchie. Chaque lien devra donc passer successivement par le point d'entrée-sortie des ancêtres de son élément de départ jusqu'à ce qu'il atteigne le niveau souhaité dans la hiérarchie. Les liens qui entrent dans un paquetage seront rassemblés en son point d'entrée-sortie avant d'entrer dans celui-ci, pour ensuite se disperser vers leur cible respective une fois passé ce point, répétant le même processus à chaque paquetage rencontré sur le chemin menant à cette cible. Ainsi, un lien devra passer successivement par le point d'entrée-sortie de chaque paquetage qu'il doit traverser avant d'atteindre sa cible.

Contrairement à la métaphore utilisée, nous voulons uniquement afficher dans notre visualisation les liens qui existent entre deux éléments du logiciel sous analyse. Cela signifie que, dans notre système, les éléments peuvent uniquement être liés entre eux, plutôt que d'être liés à une entité externe au logiciel, comme c'est le cas pour les appareils électriques dans un immeuble, qui reçoivent leur alimentation électrique d'une centrale externe et qui ne sont donc pas branchés entre eux. Il est donc nécessaire, pour lier ensemble deux éléments de notre visualisation, d'appliquer successivement, dans l'ordre, le principe de sortie et le principe d'entrée. Premièrement, à partir de l'élément de départ, il faut appliquer le principe de sortie des liens en faisant passer le lien par les points de sortie des ancêtres de cet élément, et ce jusqu'à ce qu'on monte assez haut dans la hiérarchie pour que le lien puisse aller rejoindre le point d'entrée d'un paquetage ancêtre de sa cible. Ensuite, il faut appliquer le principe d'entrée des liens pour chaque paquetage qu'on doit traverser avant d'aller rejoindre l'élément cible.

La méthode *Hierarchical Edge Bundles*, développée par Holten [10] et que nous nommerons *HEB* pour le reste du présent mémoire, est basée sur la métaphore décrite précédemment. Celle-ci permet d'afficher les liens d'adjacence de façon flexible et efficace, tout en respectant les principes essentiels de la métaphore d'origine en rassemblant les liens sur la partie de chemin qu'ils ont en commun et en les séparant par la suite. Cette méthode est en plus conçue pour être intégrée avec la plupart des techniques de visualisation d'arbres déjà existantes, y compris celles qu'utilise notre système, et ce avec une quantité minime d'effort. Toutes ces qualités font d'elle la méthode idéale pour nous, et

c'est donc évidemment celle-là que nous avons choisie d'implémenter dans notre système pour afficher les liens d'adjacence.

4.3 Création et représentation des liens d'adjacence

4.3.1 Création des liens d'adjacence avec l'algorithme *HEB*

L'objectif principal de l'auteur de cette méthode est essentiellement le même que le nôtre, c'est-à-dire d'être capable d'afficher simultanément la hiérarchie d'un logiciel et des liens d'adjacence entre les éléments de celle-ci, en limitant suffisant l'encombrement visuel et l'interférence générée par ces liens de sorte que la visualisation résultante puisse fournir des informations de haut niveau utiles à ceux qui doivent faire la maintenance de ce logiciel. Étant donné qu'il existe déjà plusieurs représentations efficaces pour afficher la hiérarchie d'un logiciel, et que le principal problème se situait jusqu'à maintenant au niveau de l'affichage des liens d'adjacence, plus précisément de leur affichage simultané avec la hiérarchie, l'algorithme développé par l'auteur s'occupe uniquement de créer et d'afficher les liens d'adjacence, puis de les ajouter à une représentation hiérarchique déjà existante. En fait, l'algorithme *HEB* repose entièrement sur la représentation hiérarchique utilisée : celle-ci lui permet non seulement d'être très efficace pour réduire l'encombrement visuel dans les liens d'adjacence et pour en faire ressortir des informations de plus haut niveau, mais, dans les faits, l'algorithme ne pourrait même pas fonctionner sans la représentation hiérarchique sur laquelle il se base. Cet algorithme ne se contente donc pas seulement de créer séparément les liens d'adjacence puis de les ajouter bêtement à une représentation quelconque des liens hiérarchiques d'un logiciel. Au contraire, celui-ci travaille de pair avec la représentation hiérarchique utilisée afin d'appliquer les principes de la métaphore mentionnée précédemment et, de cette façon, de produire une visualisation des liens d'adjacence de qualité. Évidemment, l'inconvénient qu'engendre une telle dépendance est que l'algorithme *HEB* ne peut pas être utilisé pour afficher les liens d'adjacence entre des éléments qui ne sont pas organisés de façon hiérarchique. Malgré tout, cet inconvénient n'est pas à prendre en considération étant donné que cet algorithme a été développé justement dans le but de représenter simultanément des liens

hiérarchiques et des liens d'adjacence, et donc d'être utilisé uniquement sur des données organisées de façon hiérarchique.

L'idée de base de l'algorithme, qui guide sa façon de créer les liens d'adjacence, consiste essentiellement à réunir, plus ou moins fortement, les liens qui partagent un même chemin. Plus précisément, l'idée est de les réunir en paquet seulement sur la partie de chemin qu'ils ont en commun, puis de les séparer lorsque leurs chemins divergent. Dans cet algorithme, le chemin entre deux éléments est basé sur la hiérarchie du logiciel. En considérant celle-ci comme étant un arbre où chaque nœud représente un élément du logiciel (paquetage ou classe), le chemin entre deux nœuds, donc entre deux éléments, se définit comme étant le plus court chemin existant dans l'arbre entre le nœud de départ et le nœud d'arrivée. Plus précisément, le chemin est constitué, dans l'ordre qu'on les rencontre, des nœuds suivants :

1. Le nœud de départ, puis, en remontant dans l'arbre, tous les nœuds qu'on rencontre jusqu'au plus petit commun ancêtre (PPCA).

2. Le PPCA.

3. À partir du PPCA, tous les nœuds qu'on rencontre en descendant dans l'arbre jusqu'au nœud d'arrivée, incluant celui-ci.

L'idée maîtresse de l'algorithme est de représenter les liens d'adjacence par des courbes, et d'utiliser la position, dans la représentation hiérarchique, des nœuds du chemin entre deux éléments liés comme points de contrôle de la courbe représentant le lien d'adjacence entre ceux-ci. Cela signifie que la représentation hiérarchique utilisée doit absolument définir, pour chaque nœud de la hiérarchie, un point indiquant la position de celui-ci dans la visualisation, sans quoi l'algorithme ne pourrait pas définir les points de contrôle des courbes et, ainsi, ne pourrait pas les tracer. Cette contrainte est essentiellement la seule modification à apporter à une représentation hiérarchique pour qu'on puisse y ajouter des liens d'adjacence avec l'algorithme *HEB*. La position des nœuds est triviale à déterminer dans les représentations de type nœuds-liens standards, comme celles décrites à la section 2.2.1, où celles-ci correspondent tout simplement à la position

de l'entité représentant le nœud. Par contre, pour des représentations plus complexes, comme celles utilisées dans notre système, la façon de définir la position des nœuds n'est pas toujours aussi simple. Les détails sur la façon de procéder pour chaque type de placement offert par notre système seront donnés à la section 4.3.3.1. Malgré tout, cet algorithme peut s'adapter sans effort à la plupart des représentations hiérarchiques existantes, ce qui le rend d'autant plus utile étant donné qu'il nous laisse le champ libre sur la méthode à utiliser pour afficher les liens hiérarchiques.

Avant de pouvoir tracer les liens, il faut décider quel type de courbe nous voulons utiliser pour cette tâche. Après en avoir essayé différentes sortes, l'auteur a décidé d'utiliser des courbes B-splines cubiques (degré = 3), car celles-ci offrent un bon niveau de contrôle local, ce qui permet de resserrer efficacement les courbes (les liens) en paquets seulement sur les sections de chemin voulues, et ce sans affecter le reste de chaque courbe. De plus, ce type de courbe ne présente pas une complexité de calcul énorme, ce qui est important si nous voulons les utiliser dans un système interactif qui doit afficher les résultats en temps réel, comme c'est le cas dans l'outil développé par l'auteur et ainsi que dans le nôtre. La figure 4.3 montre un exemple de chemin entre deux éléments d'un « arbre de ballons », ainsi que la courbe B-spline obtenue pour représenter le lien entre ceux-ci.

Afin de donner plus de flexibilité à la façon dont l'algorithme *HEB* rassemble les liens en paquets, l'auteur de cet algorithme a ajouté un paramètre β, où $\beta \in [0,1]$, servant à contrôler la force du resserrement des liens en influençant la position des points de contrôle utilisés pour les créer. Pour y arriver, avant de créer un lien, il applique la formule suivante sur chacun de ses points de contrôle afin de les déplacer :

$$P'_i = \beta \cdot P_i + (1 - \beta)(P_0 + \frac{i}{N-1} (P_{N-1} - P_0))$$

où :

- N est le nombre total de points de contrôle qu'utilise le lien.
- i est l'indice du point de contrôle ($i \in \{0, ..., N - 1\}$).
- P_i est le $i^{ème}$ point de contrôle.

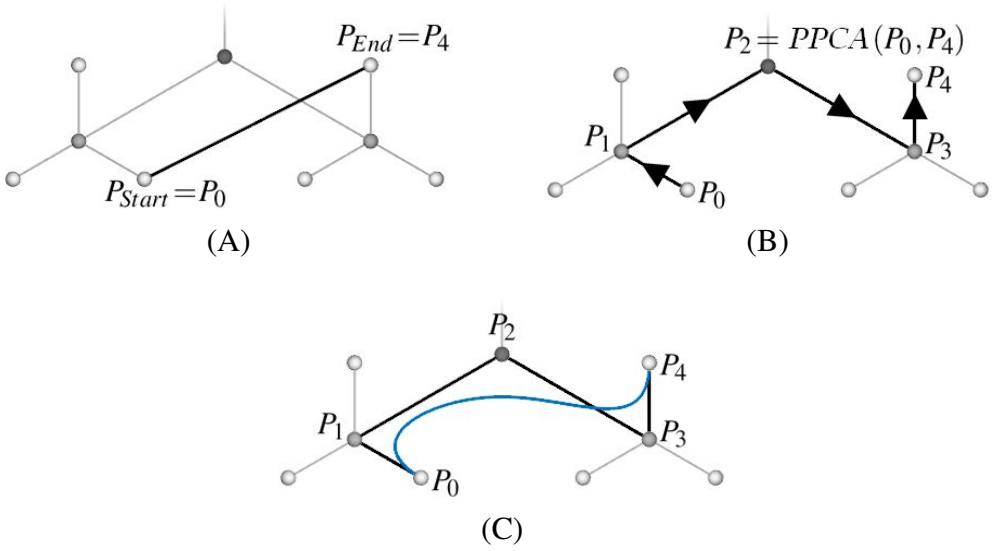


Figure 4.3 – L'image A) montre une partie d'un « arbre de ballons », où P_{Start} (P_0) est l'élément de départ et P_{End} (P_4) l'élément cible, et où le lien d'adjacence entre ces deux éléments est représenté par une ligne droite. L'image B) montre le chemin reliant ces deux éléments. Du point de vue de la hiérarchie, pour aller de P_0 à P_4, on doit commencer, à partir de P_0, par monter dans la hiérarchie jusqu'à P_2, qui est le PPCA de P_0 et de P_4. Ensuite, à partir de P_2, on doit descendre dans la hiérarchie jusqu'à P_4, l'élément cible. Le chemin reliant P_0 à P_4 est donc constitué, dans l'ordre, des nœuds P_0, P_1, P_2, P_3 et P_4. L'image C) montre comment, en utilisant la position de ces nœuds dans la visualisation comme points de contrôle, on crée la courbe représentant le lien d'adjacence entre les éléments P_0 et P_4. Ces images sont toutes tirées de [10].

- P'_i est le point obtenu en déplaçant le point de contrôle P_i original.
- β est le paramètre de resserrement ($\beta \in [0, 1]$).

Pour comprendre l'effet de cette formule, prenons les deux valeurs extrêmes de β. Si $\beta = 1$, alors $P'_i = P_i$, et donc les points de contrôle utilisés restent exactement les mêmes. À l'autre extrémité, si $\beta = 0$, alors P'_i correspond à un point situé sur la ligne droite reliant le point de départ (P_0) et le point d'arrivée (P_{N-1}). Sa position sur cette ligne va correspondre au rapport entre sa position, dans la liste de points de contrôle, et le nombre total de points de contrôle utilisés. En résumé, cela signifie que si $\beta = 1$, alors les liens seront resserrés en paquets de façon maximale, et que plus la valeur de β diminue, plus les paquets de liens vont se relâcher, rapprochant ainsi de plus en plus la visualisation de celle utilisant des lignes droites, jusqu'à y correspondre parfaitement lorsque $\beta = 0$.

Utiliser une valeur de β élevée, c'est-à-dire resserrer fortement les liens en paquets compacts, présente évidemment d'indéniables avantages, notamment en ce qui concerne la réduction de l'encombrement visuel dans l'affichage des liens d'adjacence, ce qui était au départ notre plus grand problème. Aussi, plus on resserre les liens, plus ceux-ci font ressortir de l'information de haut niveau, c'est-à-dire de l'information concernant les liens entre les paquetages de haut niveau dans la hiérarchie. Ce genre d'information s'avère très utile pour quelqu'un qui n'est pas familier avec le logiciel et qui veut, comme première approche, se donner une idée globale des liens existant entre les parties principales de celui-ci. Cette abstraction de l'information s'explique par le fait que, lorsque des liens sont rassemblés en un paquet très compact, ceux-ci deviennent indiscernables, ce qui nous empêche d'associer de façon précise les éléments de départ avec les éléments cibles correspondants. Dans cette situation, nous pouvons donc seulement associer de façon grossière une *région* de départ, où commence un groupe de liens, avec la *région* cible vers où celui-ci se dirige, chacune de ces régions correspondant à un paquetage de haut niveau dans la hiérarchie. Ces groupes de liens, une fois fusionnés en un paquet, forment ainsi un lien dit *implicite* entre les paquetages contenant les éléments qu'ils relient. Par *implicite*, nous voulons dire que, étant donné que les liens peuvent se faire uniquement entre les classes du système, il ne peut y avoir de lien défini de façon explicite entre deux

paquetages. Malgré tout, les paquetages sont quand même liés par l'entremise des liens existant entre leurs éléments descendants, et c'est à cela que correspondent les liens implicites entre eux. En résumé, resserrer fortement les liens d'adjacence permet de cacher les détails de ceux-ci dans la visualisation et ainsi de faire ressortir la façon dont les paquetages principaux du logiciel, c'est-à-dire ceux situés dans les premières couches de la hiérarchie, sont liés, tout en réduisant de façon considérable l'encombrement visuel. Au final, cela a pour effet de résumer l'information que la visualisation nous donne sur les liens d'adjacence présents dans le logiciel en ne présentant que le schéma global des tendances dans ceux-ci.

Malgré tous les avantages que procure un fort resserrement des liens, donner au paramètre β sa valeur maximale ne résulte pas en une visualisation parfaite et idéale des liens d'adjacence, même si cette valeur permet de réduire au maximum l'encombrement visuel. Au contraire, une visualisation utilisant des liens fortement resserrés, surtout des liens resserrés au maximum, contient son lot de défauts. Un premier reproche qu'on peut lui faire est en fait le défaut de sa qualité, c'est-à-dire qu'elle simplifie trop l'affichage des liens d'adjacence et cache ainsi trop de détails pour être vraiment utile. En effet, même pour une analyse globale du logiciel, une visualisation utilisant une valeur de β extrêmement élevée résume trop les informations que l'on peut en tirer sur les liens d'adjacence présents dans celui-ci, celles-ci s'avérant trop sommaires pour être vraiment utiles. Il est aussi plus difficile de quantifier les paquets de liens lorsque ceux-ci sont trop compacts, car dans de tels cas les liens sont tellement serrés qu'ils s'affichent les uns par-dessus les autres. Les paquets de liens qu'ils forment (les liens implicites) sont donc tous très minces, souvent aussi minces qu'un lien simple, ce qui nous empêche de se faire une idée approximative du nombre de liens formant chaque paquet et du sens de ceux-ci. En relâchant un peu les paquets de liens, la largeur de ceux-ci nous donne une meilleure approximation de la quantité de liens qui les composent, du moins en comparaison avec les autres paquets de liens de la visualisation. De plus, les liens devenant plus faciles à discerner, leurs couleurs ne se mélangent pas autant, nous donnant ainsi une meilleure idée sur le sens des liens formant le paquet. Finalement, des liens trop resserrés peuvent aussi donner lieu à de l'ambiguïté dans les connexions entre les éléments, comme le

montre la figure 4.4.

En fin de compte, un trop fort resserrement des liens présente certains avantages et certains inconvénients, tout comme c'est le cas pour un resserrement trop faible. Il convient donc, pour avoir une visualisation des liens d'adjacence de qualité, de trouver le juste milieu entre ces deux extrêmes, juste milieu qui peut varier selon divers facteurs, tels la tâche d'analyse à accomplir ou le type de logiciel visualisé. C'est pour cette raison que, afin que l'outil qu'il a développé soit flexible à ce niveau, l'auteur de l'algorithme *HEB* a inclus dans celui-ci la possibilité de changer dynamiquement la valeur du paramètre β et d'afficher en temps réel les résultats de ce changement. Ainsi, un utilisateur peut facilement changer le niveau d'information présenté, en commençant par exemple avec un β élevé afin de voir les tendances générales dans les liens, puis de le diminuer graduellement afin de faire ressortir les liens existant entre les paquetages de plus bas niveaux. Notre système aussi offre la possibilité de changer dynamiquement la valeur de β, en plus d'offrir divers autres outils interactifs qui permettent de faire ressortir de l'information plus détaillée. Il est en général préférable d'utiliser une valeur de β assez élevée lorsqu'on veut observer l'ensemble des liens du système et d'utiliser cette gamme d'outils pour investiguer plus en détail les zones d'intérêts dans la visualisation plutôt que de diminuer la valeur de β. La section 4.5 donnera plus de détails sur les outils interactifs offerts par notre système, ainsi que sur la façon de les utiliser pour investiguer les données. Nous avons déterminé qu'affecter au paramètre β une valeur de 0.9 donnait des résultats satisfaisants pour l'ensemble des logiciels testés et, à moins d'avis contraire, pour le reste de ce mémoire c'est cette valeur qui sera utilisée dans les exemples de visualisation obtenue avec notre système.

Une dernière étape qu'effectue l'algorithme *HEB* avant de tracer les courbes est qu'il enlève le PPCA du chemin entre deux nœuds liés si celui-ci en contient plus de trois. La raison à cela est que la courbure des liens vers le parent commun (le PPCA) des nœuds qu'ils relient ne procure aucune information de haut niveau supplémentaire sur la connexion entre les paquetages. Au contraire, cela ne fait qu'encombrer inutilement la région autour du PPCA. La figure 4.5 montre un exemple illustrant ce procédé, tandis que la figure 4.6 montre pourquoi il est important de ne pas enlever le PPCA lorsqu'il

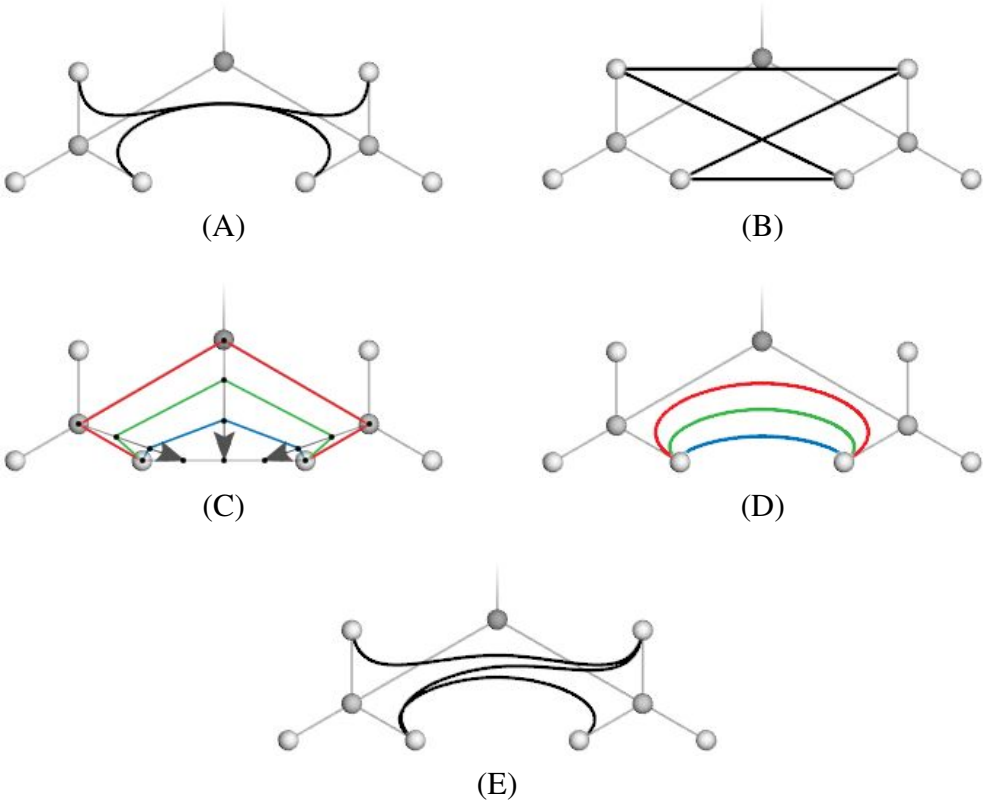


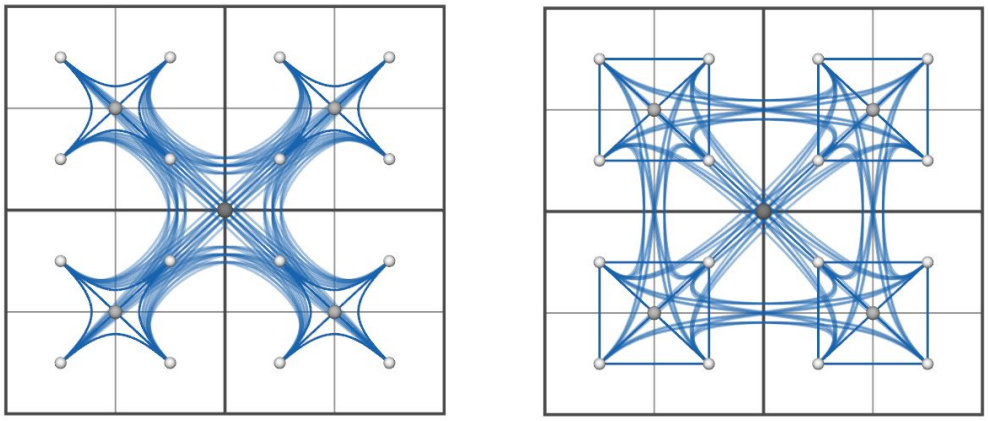
Figure 4.4 – Résoudre l'ambiguïté dans les connexions entre les éléments. L'image A) montre des liens créés avec $\beta = 1$. Les courbes qui en résultent sont tellement resserrées que leurs parties centrales coïncident parfaitement. Cela fait en sorte qu'il nous est impossible de déterminer s'il existe un lien seulement entre la paire de nœuds du haut et entre celle du bas ou s'il y a quatre liens en tout, comme montré dans l'image B). L'image C) montre les points de contrôle qu'on obtient avec différentes valeurs de β, tandis que l'image D) montre la courbe que ceux-ci génèrent (courbe rouge : $\beta = 1$, courbe verte : $\beta = 0.66$, courbe bleue : $\beta = 0.33$). Ces images montrent comment, en diminuant la valeur de β, on rapproche les points de contrôle, et du même coup la courbe qu'ils génèrent, de la ligne droite reliant les nœuds, ce qui permet de séparer les liens les uns des autres. L'image E) montre les mêmes liens que dans l'image A), mais avec $\beta = 0.8$. Cette valeur de β est assez élevée pour resserrer les liens en paquets et assez basse pour éviter qu'ils se confondent, ce qui permet de résoudre l'ambiguïté présente dans l'image A) et ainsi de nous montrer qu'il y a trois liens en tout. Ces images sont toutes tirées de [10].

y a seulement trois nœuds dans le chemin. Enlever le PPCA permet de mieux discerner les paquets de liens et ainsi d'obtenir de l'information plus détaillée sur les connexions entre les paquetages du logiciel, tout cela au coût d'une légère augmentation de l'encombrement visuel. Même si cela ressemble à première vue au même effet que de diminuer la valeur du paramètre β, une importante différence entre les deux est que, même en enlevant le PPCA, les paquets de liens sont encore resserrés avec la force que l'on veut : ceux-ci vont seulement plus directement vers leur paquetage cible. La raison est que, lorsqu'on diminue la valeur de β, ce sont les liens individuels qui deviennent plus directs entre les nœuds qu'ils relient (i.e. deviennent progressivement des lignes droites), tandis qu'en enlevant le PPCA, ce sont les paquets de liens qu'on rend plus directs. Autrement dit, lorsqu'on baisse la valeur de β, on relâche graduellement tous les paquets de liens, et ce à tous les niveaux, tandis qu'en enlevant le PPCA on ne fait que séparer la première couche de paquets de liens en leurs sous-paquets respectifs, nous donnant ainsi un niveau d'information plus détaillé sur les connexions entre les paquetages du logiciel. Les exemples de notre visualisation montrés dans le reste du mémoire utiliseront ce procédé, sauf avis contraire.

La figure 4.7 montre un exemple de visualisation utilisant les *HEB*. Celle-ci provient de l'outil que l'auteur a développé, et l'exemple présente la même visualisation plusieurs fois, mais avec une valeur de β différente pour chacune afin de montrer l'effet que le changement de la valeur de ce paramètre produit sur les liens.

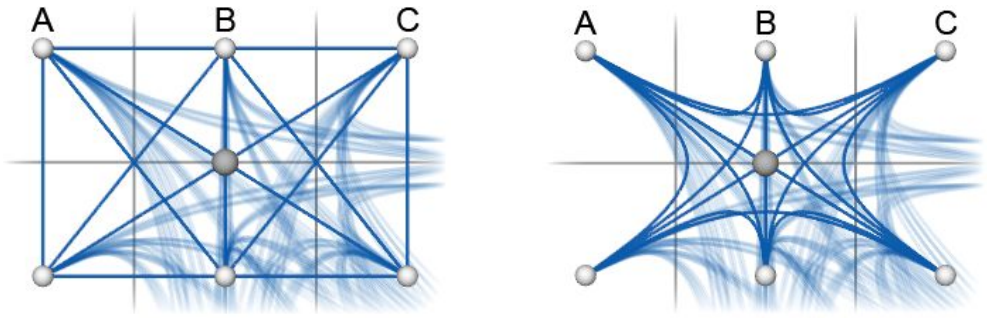
4.3.2 Caractéristiques des liens affichés dans la visualisation

Avant de continuer, il est important de souligner que notre système représente tous les liens existant entre deux éléments avec une seule représentation graphique, c'est-à-dire qu'un lien est représenté avec une seule courbe, décision qui est justifiée par le fonctionnement même des *HEB*. En effet, comme décrit à la section 4.3.1, l'algorithme *HEB* utilise le chemin de nœuds existant entre chaque paire d'éléments liés comme points de contrôle servant à créer la courbe qui les unit. Cela signifie que, étant donné que tous les liens existant entre deux mêmes éléments ont exactement le même chemin de nœuds, les courbes créées pour les représenter sont identiques et coïncident parfaitement sur toute



(A) (B)

Figure 4.5 – Effet sur les liens de l'enlèvement du PPCA. L'image A) montre un exemple où beaucoup de liens ont un chemin qui passe par la racine du logiciel. Ceci crée une courbure vers la racine dans les paquets de liens reliant les quatre paquetages principaux du logiciel, ce qui a pour effet d'encombrer le centre de la visualisation et de rendre moins évidentes les connexions existant entre ces paquetages. L'image B) montre la visualisation obtenue après avoir enlevé le PPCA dans les chemins entre les nœuds. Le centre de la visualisation est moins encombré, et les paquets de liens vont directement d'un paquetage à l'autre, ce qui permet de mieux cerner le niveau d'interconnexion entre ceux-ci. Ces images sont toutes tirées de [10].



(A) (B)

Figure 4.6 – Ambiguïté causée par l'enlèvement du PPCA. L'image A) montre un exemple d'ambiguïté que cause l'enlèvement du PPCA lorsque le chemin contient seulement trois nœuds. Dans cette situation, si on enlève le PPCA, alors cela veut dire qu'il reste seulement deux nœuds dans le chemin : le nœud de départ et le nœud d'arrivée. Lorsqu'on trace une courbe avec seulement deux points de contrôle, cela revient à tracer une ligne droite entre ces points, ce qui est le cas dans l'image A) entre les nœuds A, B et C. Comme ces trois nœuds sont alignés, cela crée de l'ambiguïté, car on ne peut être certain de la façon dont ces nœuds sont liés. Il pourrait tout aussi bien y avoir un seul lien entre A et C ou, au contraire, y avoir deux liens, un entre A et B et l'autre entre B et C. L'image B) montre comment, en conservant le PPCA lorsque le chemin contient seulement trois nœuds, on parvient à résoudre cette ambiguïté. Ces images sont toutes tirées de [10].

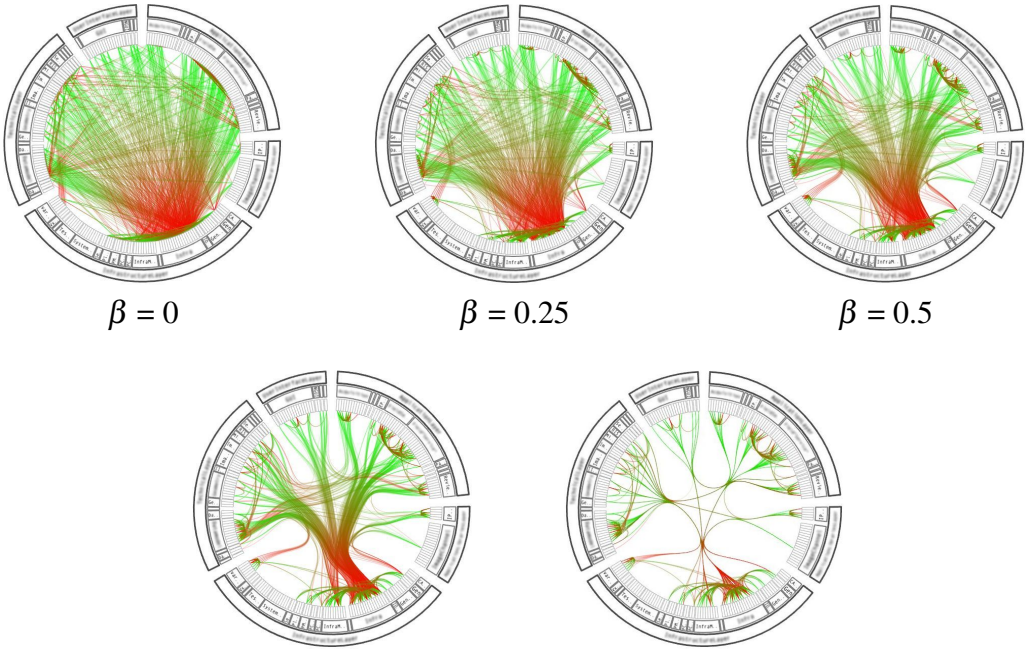


Figure 4.7 – Exemple de visualisation utilisant les *HEB* dans un placement radial. Les différentes visualisations, en utilisant une valeur croissante de β, nous montrent comment les liens se resserrent en paquets au fur et à mesure que nous augmentons la valeur de ce paramètre. Cela nous fait voir aussi comment ce resserrement permet de diminuer l'encombrement visuel et de faire ressortir les liens implicites entre les paquetages, cachant ainsi les détails dans les liens d'adjacence afin de permettre à l'utilisateur de se concentrer sur de l'information de plus haut niveau. Nous pouvons voir aussi les inconvénients qu'apporte une trop forte valeur de β dans la visualisation utilisant $\beta = 1$, où les liens entre les paquetages sont difficiles à quantifier, c'est-à-dire qu'on ne peut savoir combien de liens les composent. Les deux premières visualisations ($\beta = 0$ et $\beta = 0.25$) montrent qu'une valeur de β trop faible donne lieu à des visualisations comportant trop d'encombrement visuel pour être vraiment utiles. La visualisation utilisant $\beta = 0.75$ représente en ce qui nous concerne le juste milieu, montrant ainsi qu'il est préférable d'utiliser un β relativement élevé pour obtenir une visualisation satisfaisante.

leur longueur, ce qui les rend indiscernables aux yeux de l'utilisateur. Il serait donc inutile de vouloir représenter ces liens avec des courbes distinctes. De plus, même si nous modifions l'algorithme *HEB* afin de rendre ces liens discernables, par exemple en les décalant un peu de leur chemin normal de façon aléatoire, cela ferait exploser la quantité de liens à afficher, du moins pour certains types de liens. Par exemple, si on considère chaque appel fait par une classe à une autre comme un lien distinct, la quantité totale de liens à afficher, même pour un système de taille modeste, serait énorme, ce qui serait beaucoup trop exigeant en termes de ressources matérielles pour être affiché en un temps raisonnable. Notre solution à ce problème est tout simplement de créer une seule courbe entre chaque paire d'éléments liés, et d'utiliser certains attributs de cette courbe pour afficher le nombre de liens représentés par celle-ci ainsi que la direction générale de ceux-ci. Les attributs choisis pour représenter ces données ainsi que la façon de les utiliser pour y parvenir sont décrits à la section 4.3.4.

Notre système est conçu principalement de façon à traiter des liens dirigés, même si des liens non dirigés peuvent aussi y être affichés. Un lien dirigé entre deux éléments définit un de ceux-ci comme étant l'élément de départ, duquel il « sort », et l'autre comme étant l'élément d'arrivée, dans lequel il « entre ». La notion d'un lien comme étant « sortant » ou « entrant » est donc relative à chacune des deux classes entre lesquelles celui-ci existe, et leur point de vue est nécessairement inversé. En effet, si le lien « sort » d'une des classes, il doit nécessairement « entrer » dans l'autre, et vice-versa. Comme nous venons de le mentionner, les liens affichés dans notre visualisation peuvent représenter l'agrégation de plusieurs liens dirigés distincts, ceux-ci pouvant aller dans des directions opposées. Dans cette situation, la notion « d'entrée » et de « sortie » d'un lien par rapport à une classe n'est donc pas définie de façon absolue, car le lien affiché peut en fait, du point de vue de la même classe, contenir à la fois des liens sortants et des liens entrants. Nous ne pouvons donc pas qualifier un lien simplement comme étant « entrant » ou « sortant » d'une classe, car celui-ci peut être les deux à la fois. Afin de prendre cette notion en considération, nous calculons pour chaque lien, et ce pour chacun des deux éléments situés à ses extrémités, deux valeurs que nous appelons respectivement le « degré de sortie » et le « degré d'entrée » du lien par rapport à cet élément :

$$D_s = \frac{L_s}{L_s+L_e} \qquad D_e = \frac{L_e}{L_s+L_e}$$

où :

- D_s est le degré de sortie du lien.
- D_e est le degré d'entrée du lien.
- L_s est le nombre de liens sortants, par rapport à l'élément concerné, contenus dans le lien affiché.
- L_e est le nombre de liens entrants, par rapport à l'élément concerné, contenus dans le lien affiché.

Ces valeurs, comprises entre 0 et 1, représentent le rapport existant entre les liens sortants et les liens entrants contenus dans le lien, du point de vue de la classe située à l'extrémité où celles-ci sont calculées. Plus le degré de sortie est élevé, plus le lien veut « sortir » de la classe, tandis que plus le degré d'entrée est élevé, plus le lien veut « entrer » dans celle-ci. Ces deux valeurs sont complémentaires, c'est-à-dire que $D_s + D_e = 1$. Par exemple, si le degré de sortie est de 75%, alors le degré d'entrée sera de 25%. Les degrés de sortie calculés à chaque extrémité sont aussi complémentaires, de même que les degrés d'entrée. Par exemple, pour un lien existant entre les éléments A et B, si le degré de sortie du lien à l'élément A est de k, alors celui à l'élément B sera de $1 - k$. Ces deux valeurs sont aussi inversées d'une extrémité à l'autre, c'est-à-dire que le degré de sortie calculé à une extrémité correspond au degré d'entrée calculé à l'autre et vice-versa. Ceci est cohérent avec ce que nous venons de mentionner ci-haut, c'est-à-dire que tout ce qui « sort » d'un des deux éléments liés doit nécessairement « entrer » dans l'autre. Finalement, les valeurs extrêmes, c'est-à-dire 0 ou 1, sont obtenues seulement lorsque tous les liens que représente le lien affiché vont dans la même direction.

4.3.3 Intégration de l'algorithme *HEB*

Comme mentionné à la section 4.3.1, l'algorithme *HEB* doit absolument s'appuyer sur une représentation hiérarchique dans laquelle chaque nœud de la hiérarchie a une

position bien définie. Comme décrit au chapitre 3, notre système utilise trois représentations hiérarchiques différentes pour afficher les éléments d'un logiciel. La principale tâche que nous avons à effectuer pour adapter l'algorithme *HEB* à notre système est donc de modifier les représentations hiérarchiques utilisées afin que celles-ci associent une position, dans la visualisation, à chaque nœud de la hiérarchie, c'est-à-dire à chaque élément du logiciel visualisé. La position d'un nœud de la hiérarchie est indépendante de la représentation graphique utilisée pour afficher l'élément que le nœud représente. Par contre, si nous voulons obtenir une visualisation de qualité des liens d'adjacence, qui soit cohérente avec la représentation hiérarchique sur laquelle elle s'appuie et qui exploite au maximum les avantages offerts par l'algorithme *HEB*, notamment pour réduire l'encombrement visuel et pour faire ressortir de l'information de plus haut niveau, la représentation graphique des éléments dans le placement doit être prise en considération lorsque nous décidons comment placer les nœuds qui leur sont associés. La section 4.3.3.1 explique comment nous positionnons les nœuds dans chacune des représentations hiérarchiques offertes par notre système afin d'obtenir la meilleure visualisation possible des liens d'adjacence. La section 4.3.3.2 explique le principe d'ajout d'un nœud de sortie aux groupes de classes du logiciel.

4.3.3.1 Positionnement des nœuds de la hiérarchie dans la visualisation

Un logiciel *Java* peut être vu comme étant constitué de deux sortes d'éléments : les classes, qui représentent le contenu du logiciel, et les paquetages, qui servent à structurer de façon hiérarchique ce contenu. Tout comme les éléments d'un logiciel remplissent une fonction différente au sein de celui-ci selon la catégorie d'éléments à laquelle ils appartiennent, les nœuds ont un rôle différent à jouer dans la création des liens d'adjacence selon le type d'élément auquel ils sont associés. Les nœuds se retrouvent ainsi divisés en deux catégories : ceux associés à une classe et ceux associés à un paquetage, et la façon de les positionner est différente d'une catégorie à l'autre. La section 4.3.3.1.1 explique comment positionner les nœuds associés aux classes. La section 4.3.3.1.2 explique comment positionner les nœuds associés aux paquetages. Finalement, la section 4.3.3.1.3 explique comment positionner le nœud d'un paquetage lorsque celui-ci est fermé.

4.3.3.1.1 Position des nœuds associés aux classes

Contrairement aux paquetages, qui servent à structurer le logiciel les contenant, les classes d'un logiciel peuvent être liées de façon explicite par des liens d'adjacence. Dans cette situation, les nœuds associés à ces classes (les feuilles de la hiérarchie) servent de points de départ et d'arrivée aux liens d'adjacence, tandis que les nœuds associés aux paquetages (les nœuds internes de la hiérarchie) servent à les regrouper en paquets et à les guider jusqu'à leur cible. Les classes correspondent aux entités graphiques d'où commencent et où se terminent les liens d'adjacence : les nœuds qui leur sont associés doivent donc être placés de façon à ce que les courbes créées respectent ce principe. Ainsi, dans chacune des représentations hiérarchiques utilisées, un nœud associé à une classe est tout simplement placé sur la boîte représentant celle-ci (ou sur le cylindre si c'est une interface), comme montré dans la figure 4.8. Ceci fait en sorte que les courbes représentant les liens d'adjacence dans la visualisation « touchent » les classes qu'elles sont sensées relier, nous permettant ainsi de déterminer facilement, pour un lien donné, entre quelles classes de la représentation hiérarchique celui-ci existe.

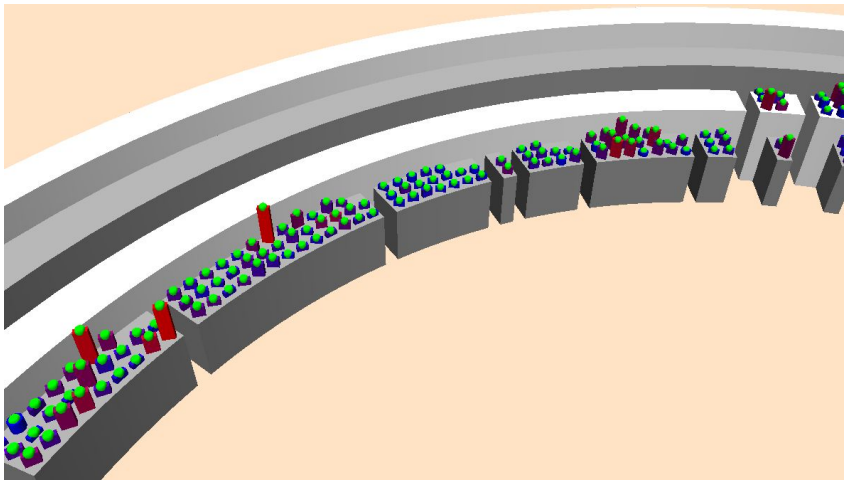


Figure 4.8 – Position dans la visualisation des nœuds associés aux classes. Les sphères vertes représentent les nœuds associés aux classes. Cet exemple utilise le placement Colisée, mais le principe est le même dans les deux autres types de placement.

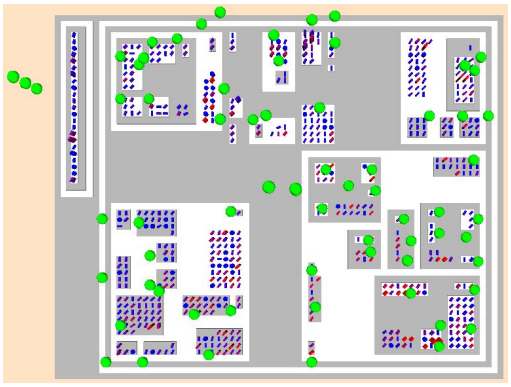
4.3.3.1.2 Position des nœuds associés aux paquetages

Comme nous venons de le mentionner à la section précédente, les nœuds associés à un paquetage sont utilisés pour guider les liens à travers la hiérarchie jusqu'à leur cible, les rassemblant en paquets et les séparant selon les paquetages traversés avant d'atteindre celle-ci. Ces nœuds peuvent ainsi être vus comme étant le squelette de l'algorithme *HEB*, leur position influençant grandement la qualité de la visualisation obtenue : il est donc primordial, dans chaque type de placement, de choisir judicieusement leur position. Il est aussi important de les positionner de sorte que la visualisation des liens d'adjacence s'intègre naturellement à la représentation de la hiérarchie et que ces deux visualisations travaillent de concert afin de nous permettre de recouper sans effort les informations données par chacune d'elles. Pour y arriver, il est important de prendre en considération la façon dont les paquetages de la hiérarchie sont représentés graphiquement lorsque nous positionnons les nœuds qui leur sont associés. La représentation graphique des paquetages étant différente entre le placement *Treemap* et les placements radial et Colisée, la façon de positionner les nœuds associés aux paquetages ne sera donc pas la même dans tous les types de placements. De plus, notre visualisation étant en trois dimensions, il est important de positionner ces nœuds de façon à profiter de la dimension supplémentaire dont nous disposons par rapport aux autres outils utilisant les *HEB* qui ont été développés jusqu'à maintenant [4, 9, 10, 24]. Afin de simplifier le procédé utilisé dans chaque type de placement pour positionner les nœuds associés aux paquetages, nous commençons par déterminer la position des nœuds dans le plan 2D sur lequel repose la représentation hiérarchique (les axes X et Z) avant de déterminer leur hauteur (l'axe Y).

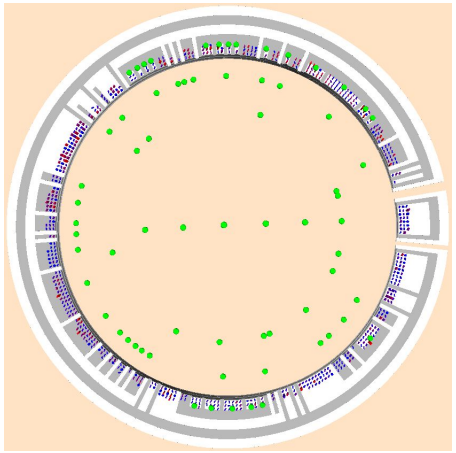
La position des nœuds dans le plan supportant la représentation hiérarchique correspond à celle qu'ils auraient si la visualisation était en deux dimensions, comme c'est le cas dans l'article sur les *HEB* [10]. La façon dont l'auteur de cet article positionne les nœuds (en 2D) dans les différents types de placements qu'il utilise, façon qui varie selon le type de placement, est pour nous dans chaque cas la façon la plus naturelle de les positionner. Ces différentes façons de positionner les nœuds sont donc celles qui, selon nous, intègrent le plus naturellement les liens d'adjacence à la représentation hié-

rarchique utilisée et qui produisent la plus grande cohérence entre les deux types de visualisation. Pour chacun de nos placements, nous positionnons donc les nœuds associés aux paquetages de la même façon que dans le placement correspondant dans [10]. Pour le placement *Treemap*, le nœud de chaque paquetage est tout simplement placé au milieu de la zone rectangulaire définie par le paquetage qui lui est associé. Dans le cas du placement radial, même si la façon dont nous le construisons n'est pas exactement la même que celle utilisée par l'auteur des *HEB*, la représentation hiérarchique qui en résulte est quand même identique, et il en va de même pour la façon de placer les nœuds dans celle-ci. Dans ce type de placement, les nœuds associés aux paquetages sont disposés sur différents anneaux dans le cercle interne du placement, où chaque anneau correspond à un niveau de la hiérarchie. Pour déterminer la position de ces nœuds, nous commençons par placer le nœud associé à la racine du logiciel au milieu du placement : celui-ci correspond au point central du placement tout entier. Ensuite, afin de déterminer la distance à laisser entre chaque anneau de nœuds, nous déterminons le rayon du cercle vide situé au centre du placement, puis nous le divisons par le niveau le plus profond de la hiérarchie, ce qui nous permet de laisser la même distance entre chaque anneau. Nous plaçons ensuite chaque nœud associé à un paquetage sur l'anneau correspondant à son niveau dans la hiérarchie, en le positionnant de façon à ce qu'il soit situé sur la bissectrice de l'angle qu'occupe la représentation graphique de son paquetage dans le placement. En ce qui concerne le placement Colisée, comme celui-ci est fortement basé sur le placement radial, la façon utilisée dans ce type de placement pour positionner les nœuds associés aux paquetages est la même que dans ce dernier. La figure 4.9 montre la position des nœuds en 2D (vue de dessus) pour chaque type de placement inclus dans notre outil.

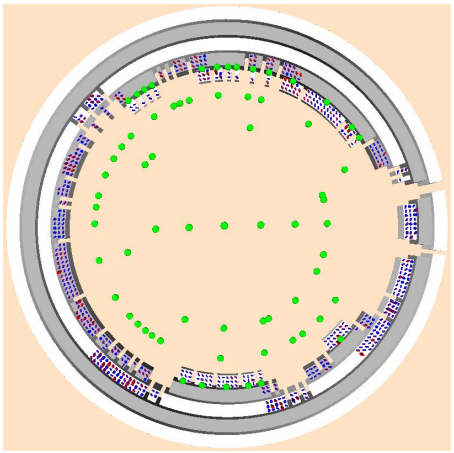
Après avoir déterminé la position des nœuds associés aux paquetages selon les axes *X* et *Z*, il nous reste encore à décider comment les positionner selon l'axe *Y*, c'est-à-dire à quelle hauteur les positionner. Si nous voulons utiliser la troisième dimension offerte par notre outil de façon à améliorer la visualisation des liens d'adjacence, nous devons positionner ces nœuds selon des hauteurs variables plutôt que de les placer tous à la même hauteur, car sinon cela reviendrait à déposer une visualisation bidimensionnelle des liens



(A)



(B)



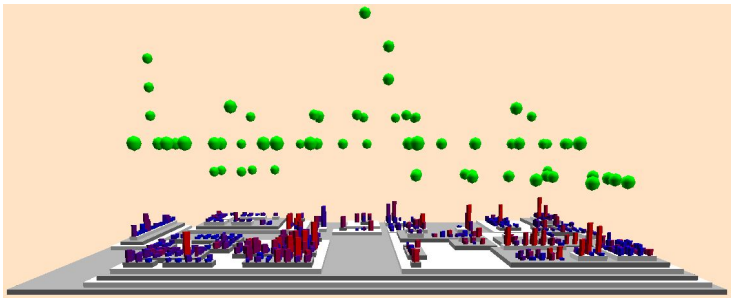
(C)

Figure 4.9 – Position en 2D des nœuds associés aux paquetages. Les sphères vertes représentent les nœuds associés aux paquetages. L'image A) montre la position de ces nœuds dans le placement *Treemap*, l'image B) celle dans le placement radial et l'image C) celle dans le placement Colisée.

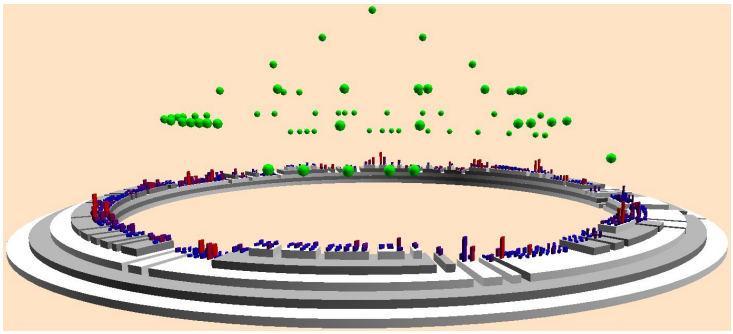
d'adjacence par-dessus notre représentation hiérarchique. Pour obtenir une visualisation des liens d'adjacence de qualité, il est aussi important que les différentes hauteurs associées à ces nœuds soient cohérentes avec la hiérarchie du logiciel. Afin d'obtenir cette cohérence, la hauteur d'un nœud associé à un paquetage est déterminée par le niveau du paquetage dans la hiérarchie : plus le paquetage est haut dans la hiérarchie, plus le nœud qui y est associé est haut dans la visualisation. Ainsi, les nœuds sont placés sur des couches successives de hauteur croissante, chaque couche correspondant à un niveau de la hiérarchie. La distance entre chaque couche de nœuds est déterminée de façon arbitraire par l'utilisateur. Afin qu'il n'y ait pas d'intersection entre les liens d'adjacence et les entités de la représentation hiérarchique, nous commençons par trouver, dans chaque type de placement, à quelle hauteur se situe le sommet de la plus haute entité graphique présente dans celui-ci. Nous ajoutons à cette hauteur maximale la distance inter-couche définie par l'utilisateur : la hauteur résultante correspond à celle de la première couche de nœuds, c'est-à-dire aux nœuds les plus profonds de la hiérarchie. La hauteur de chaque couche de nœuds successive correspond à la hauteur de la couche précédente additionnée de la distance inter-couche. Au final, positionner les nœuds associés aux paquetages de cette façon fait en sorte que plus un lien doit monter dans la hiérarchie avant d'atteindre sa cible, plus il montera dans la visualisation, permettant ainsi de mieux séparer les liens existant entre des éléments qui sont proches dans la hiérarchie de ceux existant entre des éléments qui sont situés dans deux branches éloignées de celle-ci. La figure 4.10 montre la position des nœuds en 3D (vue de côté) pour chaque type de placement inclus dans notre outil.

4.3.3.1.3 Position des nœuds associés à des paquetages fermés

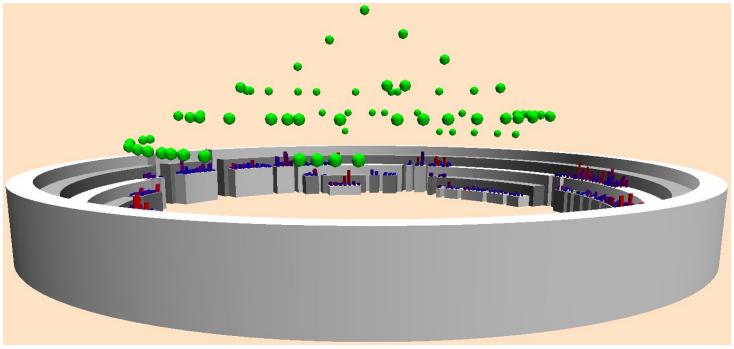
Comme nous l'avons déjà expliqué à la section 2.5, notre système offre la possibilité de fermer les paquetages afin de réduire la complexité de la visualisation. Les paquetages fermés sont alors considérés comme un élément de base, au même titre que les classes et les interfaces, et sont représentés par une entité graphique différente des paquetages ouverts. Ainsi, les paquetages fermés peuvent être liés explicitement avec



(A)



(B)



(C)

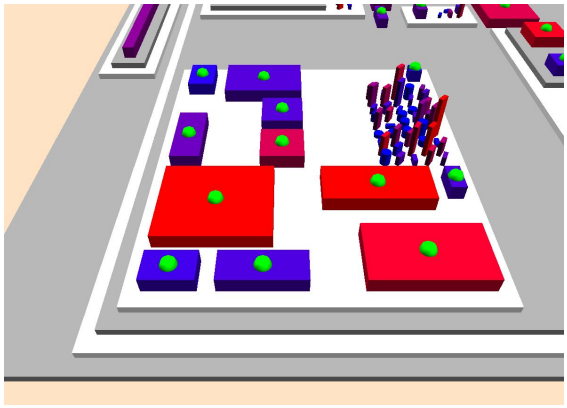
Figure 4.10 – Position en 3D des nœuds associés aux paquetages. Les sphères vertes représentent les nœuds associés aux paquetages. L'image A) montre la position de ces nœuds dans le placement *Treemap*, l'image B) celle dans le placement radial et l'image C) celle dans le placement Colisée.

d'autres éléments, ce qui nécessite d'ajouter un nœud sur ces paquetages afin que les courbes représentant les liens les impliquant les « touchent », tout comme elles le font pour une classe ou une interface. La figure 4.11 montre, pour chaque type de placement, comment nous plaçons ces nouveaux nœuds sur les entités graphiques représentant les paquetages fermés.

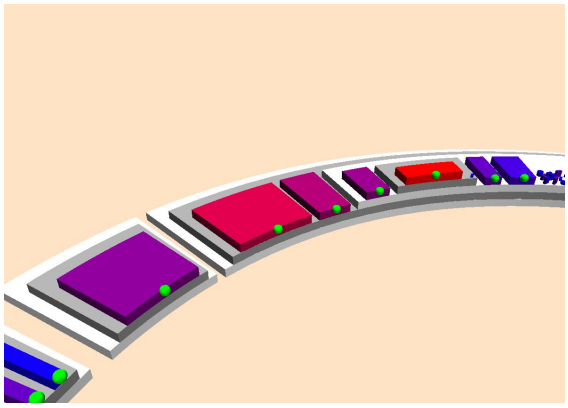
4.3.3.2 Ajout d'un nœud de sortie pour les groupes de classes

Comme nous l'avons vu au chapitre 3, dans tous les placements que nous utilisons, les ensembles de classes sœurs sont placés en groupes compacts à l'intérieur de leur parent, et les liens entrant ou sortant de cette zone compacte doivent passer par environ le même point. Cela a pour inconvénient que, lorsqu'il y a une grande quantité de liens entrant et sortant d'un tel groupe, ces deux types de liens se retrouvent mélangés, et il devient alors difficile pour l'utilisateur de bien distinguer le sens des liens allant vers ces classes. Comme il sera expliqué à la section 4.3.4.1, le sens d'un lien est représenté par la couleur de celui-ci. La figure 4.12 montre comment il est difficile de se faire une idée, pour le groupe de classes montré en exemple, sur la quantité de liens allant dans un sens ou dans l'autre, les liens entrants et sortants étant trop entremêlés pour cela. Une solution serait d'augmenter l'espace que le placement laisse entre les classes : ceci laisserait plus d'espace entre les liens rejoignant ces éléments et ainsi aiderait à les séparer. Par contre, augmenter de façon significative l'espace entre les classes ferait exploser la taille du placement, et ce peu importe le type de placement utilisé, sans oublier que le problème persisterait toujours pour les classes d'où partent et arrivent une grande quantité de liens. La solution que nous employons pour réduire ce problème consiste à créer pour chaque groupe de classes un nœud spécial, que nous appellerons « nœud de sortie » pour le reste du présent mémoire, qui est utilisé afin de guider différemment les liens selon qu'ils entrent ou qu'ils sortent du groupe de classes auquel celui-ci est associé.

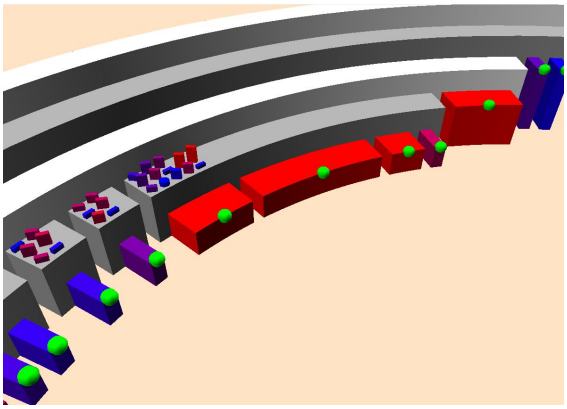
La première étape est de déterminer comment positionner les nœuds de sortie associés aux groupes de classes. Chaque nœud de sortie est positionné au centre de la zone délimitée par le groupe de classes auquel il est associé, à la même hauteur que le nœud associé au paquetage parent de ces classes. La figure 4.13 montre un exemple, pour



(A)

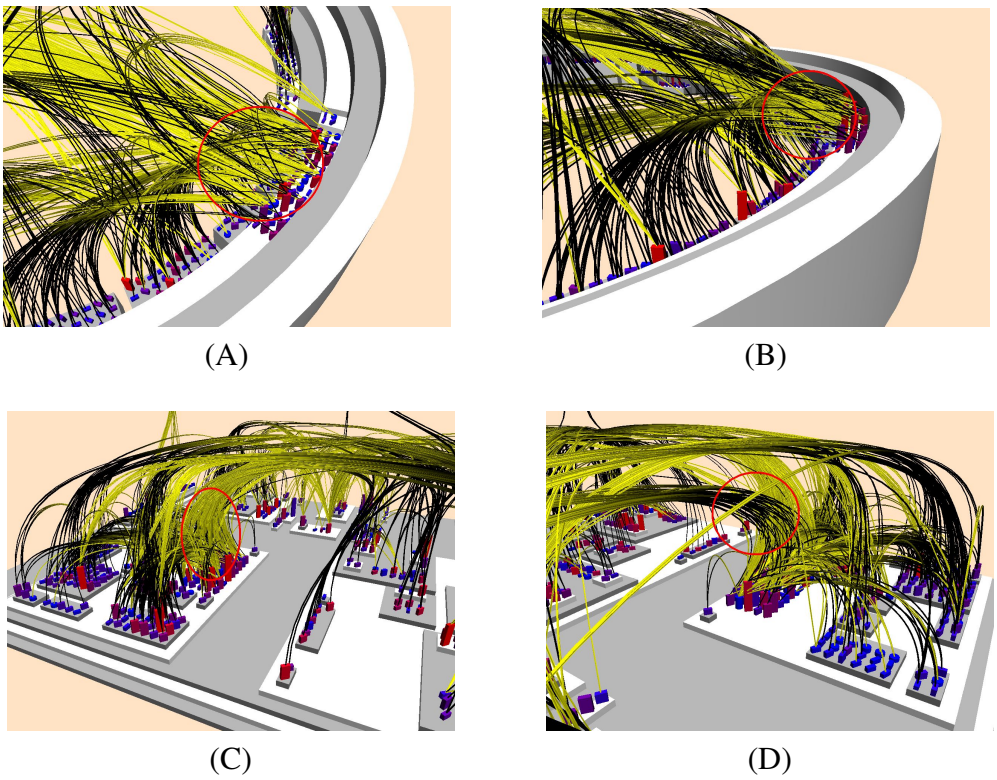


(B)



(C)

Figure 4.11 – Position des nœuds associés aux paquetages fermés. Les sphères vertes représentent les nœuds associés aux paquetages fermés. L'image A) montre la position de ces nœuds dans le placement *Treemap*, l'image B) celle dans le placement radial et l'image C) celle dans le placement Colisée.



(A) (B)

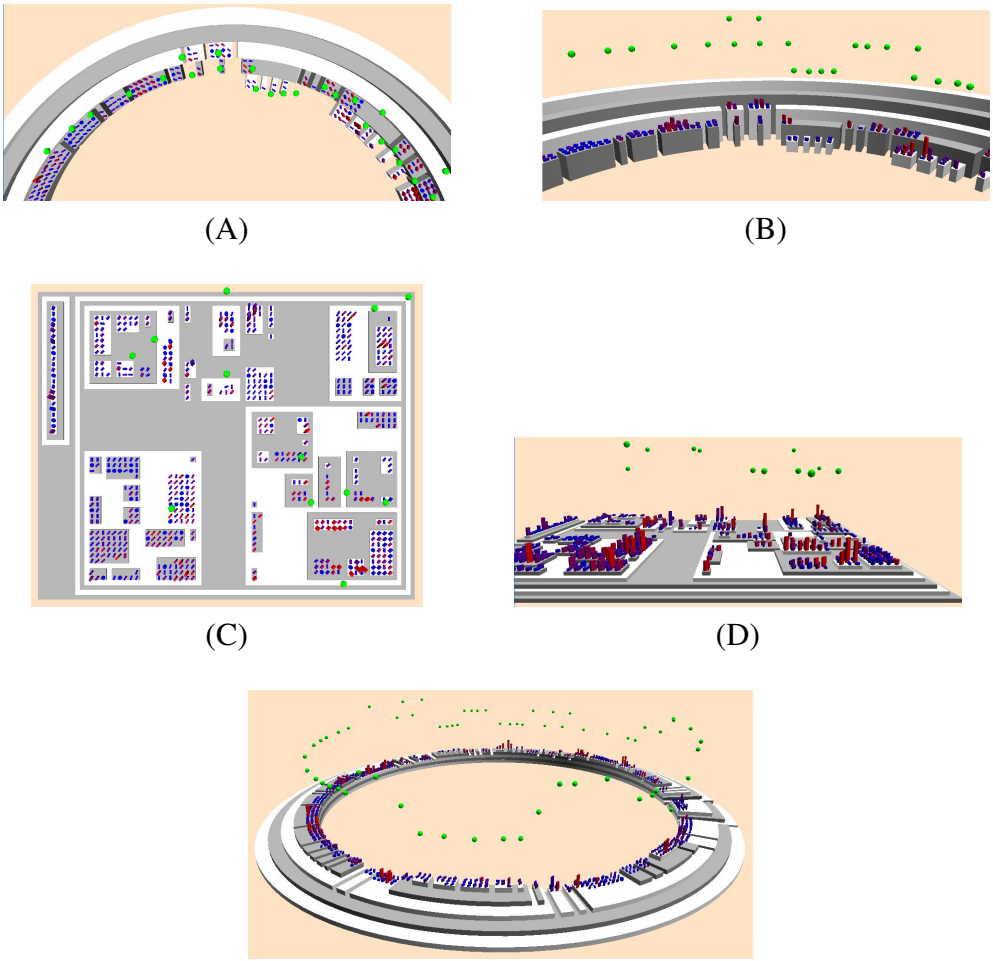
(C) (D)

Figure 4.12 – Enchevêtrement des liens entrant et sortant des groupes de classes. L'image A) montre un exemple d'un tel enchevêtrement dans le placement Colisée. Les liens en jaune sont des liens entrant dans le groupe de classes, tandis que ceux en noir sont des liens sortant de celui-ci. Il est difficile de bien juger, en se basant sur la couleur des liens, la quantité de liens entrants par rapport à la quantité de liens sortants. D'ailleurs, dans l'image B), la présence plus élevée de couleur noire dans le même amas de liens, lorsque nous le regardons sous un autre angle, laisse penser qu'il y a plus de liens sortants que ce que nous pouvions voir dans l'image A). L'image C) montre les liens entrant et sortant du même groupe de classes que celui dans les images A) et B), mais cette fois-ci en utilisant le placement *Treemap*. Vu sous cet angle, il semble qu'il y ait seulement des liens entrant dans ce groupe de classes, mais si nous les regardons sous un autre angle, comme dans l'image D), nous pouvons voir clairement qu'il y a un important groupe de liens sortant de celui-ci.

chaque type de placement, de la façon dont nous positionnons ces nœuds dans celui-ci.

Une fois la position des nœuds de sortie déterminée, il faut décider de quelle façon nous voulons les utiliser pour guider les liens entrant et sortant des groupes de classes. L'influence qu'a le nœud de sortie sur un lien donné dépend du degré de sortie de celui-ci : plus il est élevé, plus le lien passe proche du nœud de sortie. Le but de notre algorithme est donc de calculer, à chaque extrémité de chaque lien, à quel point celui-ci va passer près (ou non) du nœud de sortie pour aller rejoindre la classe concernée. Pour ce faire, notre algorithme utilise, pour un lien et pour une extrémité donnés, le degré de sortie D_s du lien à cette extrémité afin de créer un nouveau nœud. Ce nouveau nœud est situé sur la droite reliant le nœud associé à la classe concernée et le nœud de sortie du groupe de classes auquel celle-ci appartient. Plus D_s est élevé, plus ce nouveau nœud est situé près du nœud de sortie du groupe de classes, tandis que plus D_s est bas, plus il est près du nœud placé sur la classe. Par exemple, si $D_s = 1$, alors le nouveau nœud correspond exactement au nœud de sortie, tandis que si $D_s = 0$, alors il correspond exactement au nœud situé sur la classe que doit rejoindre le lien. Le nouveau nœud est ensuite ajouté dans le chemin du lien, juste avant le nœud de la classe concernée : ce nouveau nœud devient donc le « vrai » nœud de sortie de ce lien pour cette classe. Le degré de sortie D_s, calculé à une extrémité donnée d'un lien, représente donc tout simplement à quel point le lien va passer près, à cette extrémité, du nœud de sortie global au groupe de classes avant d'aller rejoindre la classe concernée. Au final, ceci a pour effet que les liens sortant d'un groupe de classes montent plus haut dans la visualisation que les liens entrant dans celui-ci, et ainsi qu'ils passent par-dessus ces derniers, ce qui évite de mélanger ces deux groupes de liens. La figure 4.14 montre quelques exemples du résultat final.

L'algorithme que nous venons de décrire s'occupe seulement d'améliorer la visualisation des liens entrant et sortant des groupes de classes, sans s'occuper des groupes de paquetages frères fermés. Cela s'explique par le fait que les groupes de paquetages frères risquent rarement d'être placés de façon assez compacte pour créer des problèmes à ce niveau, contrairement aux groupes de classes qui eux le sont presque toujours. Malgré tout, nous avons quand même décidé d'étendre cet algorithme aux paquetages fermés, ce qui permet d'améliorer un peu plus la visualisation et de s'assurer que les paquetages ne

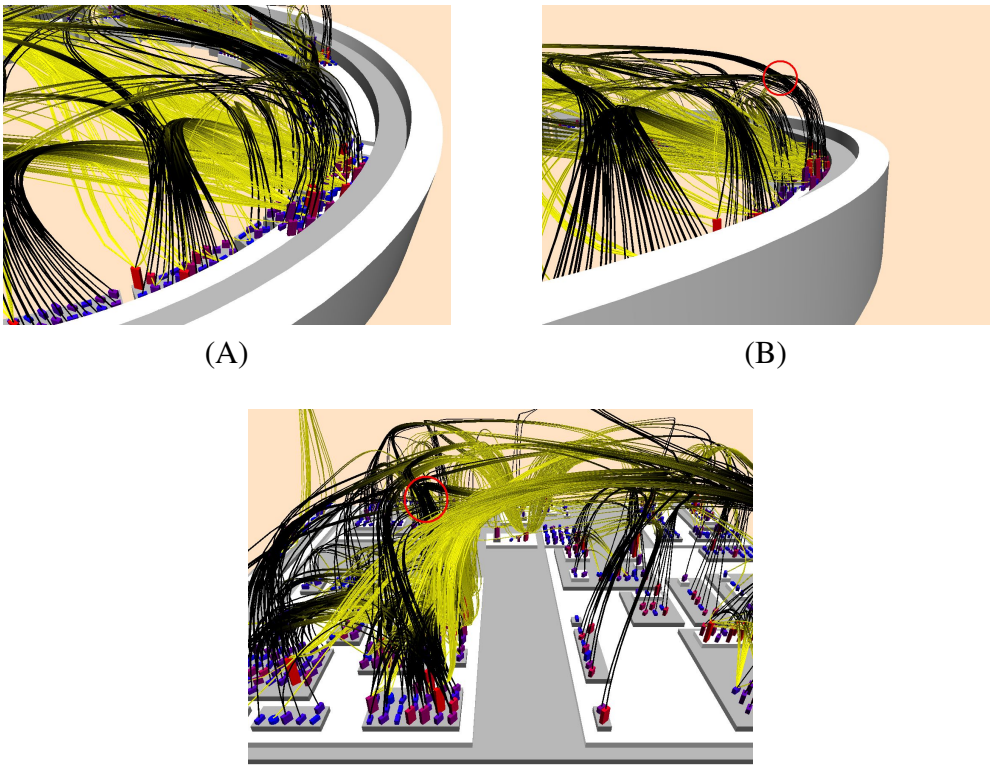


(A) (B)

(C) (D)

(E)

Figure 4.13 – Position des nœuds de sortie associés aux groupes de classes. L'image A) montre une partie d'un placement Colisée vue de dessus, où l'on voit quelques nœuds de sortie, en vert, positionnés au milieu de la zone délimitée par le groupe de classes auquel chacun d'eux est associé. L'image B) montre les mêmes nœuds avec une vue de côté, montrant ainsi qu'ils ont la même hauteur que le nœud associé au paquetage parent de leur groupe de classes. L'image C) montre l'ensemble des nœuds de sortie que nous avons ajoutés à un placement *Treemap*. Le fait qu'il y en ait si peu comparé au placement Colisée montré précédemment, qui représente le même logiciel, est que nous ajoutons seulement un nœud de sortie aux groupes de classes dont le paquetage parent n'est pas une feuille, c'est-à-dire qu'il n'a pas de sous-paquetage comme enfant. Pour les groupes de classes faisant partie de tels paquetages, le nœud de sortie que nous leur ajouterions coïnciderait parfaitement avec le nœud associé à leur paquetage parent. Dans cette situation, nous prenons donc le nœud associé au paquetage parent comme nœud de sortie au lieu d'en créer un nouveau. L'image D) montre le même placement vu de côté. L'image E) montre une vue d'ensemble des nœuds de sortie que nous avons ajoutés à un placement radial. Ceux-ci sont placés de façon identique à ceux du placement Colisée.



(A) (B)

(C)

Figure 4.14 – Exemple de séparation des liens entrants et sortants. Les images A) et B) montrent le même exemple que celui montré aux images A) et B) de la figure 4.12, mais après avoir appliqué notre nouvel algorithme de création des liens qui utilise les nœuds de sortie. Nous pouvons maintenant distinguer clairement les liens sortants des liens entrants, ces premiers passant en-dessous de ces derniers. L'image B) montre clairement le paquet de liens sortant du groupe de classes (celui qui est encerclé) monter plus haut dans la visualisation afin de passer par le nœud de sortie. L'image C) montre le même exemple que celui montré à l'image C) de la figure 4.12. L'amélioration est moins flagrante que pour le premier exemple, mais nous distinguons tout de même plus facilement le paquet de liens sortant du groupe de classes que dans l'exemple qui n'utilise pas notre nouvel algorithme.

causent pas de problèmes au niveau de l'affichage des liens entrant et sortant de ceux-ci.

L'algorithme que nous appliquons sur les paquetages fermés est exactement le même que celui décrit précédemment. La seule différence est que, au lieu de créer un nouveau nœud de sortie pour chaque paquetage fermé, nous utilisons simplement, comme nœud de sortie, le nœud qui leur est associé dans le placement. Les « vrais » nœuds de sortie que l'on crée pour guider les liens entrant et sortant d'un paquetage fermé sont donc situés sur la droite reliant le nœud placé sur celui-ci et le nœud qui lui est associé dans le placement. La figure 4.15 montre un exemple de liens entrant et sortant d'un paquetage fermé.

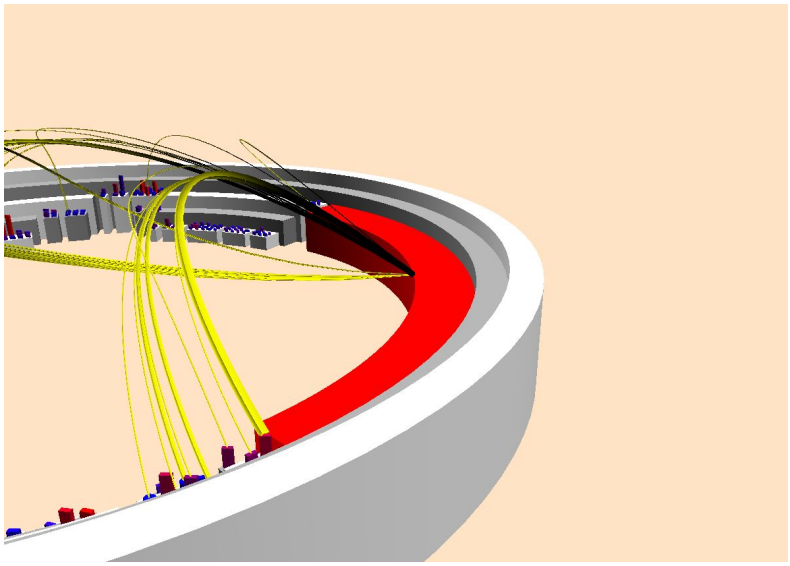


Figure 4.15 – Exemple de séparation des liens entrant et sortant d'un paquetage fermé.

4.3.4 Attributs visuels des liens

Comme nous l'avons expliqué à la section 4.3.2, les liens affichés dans la visualisation peuvent représenter l'agrégation de plusieurs liens distincts, ceux-ci pouvant aller dans des directions opposées. Afin que l'utilisateur puisse avoir une meilleure idée de l'ensemble des liens existant réellement entre les éléments du logiciel visualisé, nous utilisons certains attributs visuels des courbes afin d'afficher la quantité de liens qu'elles représentent, ainsi que la direction générale de ceux-ci. La section 4.3.4.1 explique com-

ment nous utilisons la couleur de la courbe pour afficher la direction générale des liens qu'elle représente. La section 4.3.4.2 explique comment nous utilisons l'épaisseur de la courbe pour afficher la quantité de liens qu'elle représente.

4.3.4.1 Variation de la couleur en fonction du sens des liens

Comme mentionné à la section 4.3.2, notre système est conçu principalement pour afficher des liens dirigés : il est donc primordial d'afficher clairement le sens de ceux-ci afin que l'utilisateur puisse distinguer sans effort les éléments appelants des éléments appelés. Pour afficher cette information, nous utilisons la couleur du lien, qui est selon nous l'attribut visuel le plus approprié pour accomplir cette tâche. Premièrement, la couleur est un des attributs visuels perçus le plus rapidement et le plus facilement par le système visuel humain [7] : faire la distinction entre deux couleurs est donc une tâche qui ne demande pas d'effort cognitif de la part de l'utilisateur, à condition bien sûr que ces couleurs soient suffisamment différentes (voir figure 4.16). Finalement, et pour nous cela est le plus important, la couleur d'une courbe est affichée sur la surface de celle-ci : cet attribut n'utilise donc pas d'espace supplémentaire, ni ne modifie la taille ou la forme de la courbe. Cela nous évite de créer davantage d'encombrement visuel ou d'occultation dans la visualisation, ce qui serait le cas si nous devions ajouter des entités graphiques pour représenter le sens des liens, par exemple en ajoutant une pointe de flèche à une des extrémités, ou le long de la courbe.

La première étape pour afficher le sens des liens est de choisir deux couleurs, idéalement de teintes et/ou d'intensités largement différentes, et d'associer chacune d'elle à une des deux extrémités du lien. Il est aussi important de choisir deux couleurs qui ne se confondent pas avec les couleurs utilisées dans la représentation hiérarchique, qui en fin de compte sont les couleurs d'arrière-plan des liens d'adjacence. Nous avons donc une « couleur de départ », qui indique le début du lien (l'élément appelant), et une « couleur d'arrivée », qui en indique la fin (l'élément appelé). Le sens d'un lien est indiqué en appliquant, sur la courbe le représentant, un dégradé de couleurs allant de la couleur de départ jusqu'à la couleur d'arrivée. Une particularité de notre algorithme est qu'il sépare chaque courbe en trois segments de longueur égale : le segment de départ, le segment

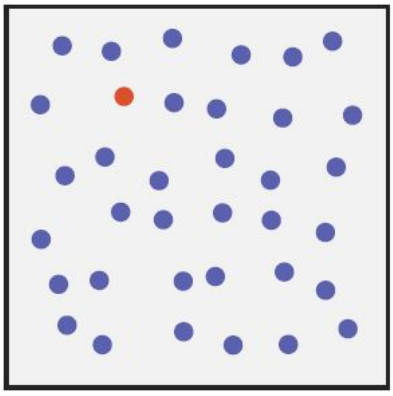
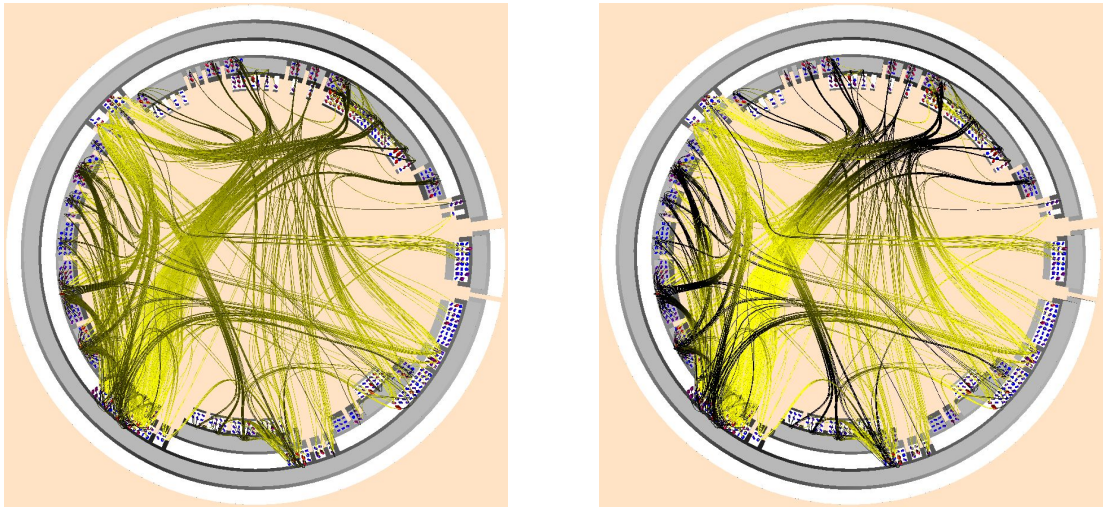


Figure 4.16 – Exemple de détection pré-attentive de la couleur. Il est extrêmement facile dans cette image de repérer le cercle rouge parmi les cercles bleus. Image tirée de [7].

de transition et le segment d'arrivée. Le segment de départ correspond au premier tiers de la courbe, à partir de l'élément de départ (l'élément appelant), tandis que le segment d'arrivée correspond au dernier tiers de la courbe, qui va jusqu'à l'élément d'arrivée (l'élément appelé). Chacun de ces segments est coloré uniquement avec la couleur associée à l'extrémité du lien qui leur correspond. Le segment de transition correspond au tiers central de la courbe : c'est sur cette partie que nous effectuons une transition linéaire entre la couleur de départ et la couleur d'arrivée. Le but de cette séparation de la courbe est d'offrir le contraste maximal possible, avec les couleurs de départ et d'arrivée choisies, entre le début et la fin de celle-ci, et ainsi de faire ressortir chacune de ses extrémités plus vivement dans la visualisation. Ceci permet à l'utilisateur de mieux voir les groupes de segments de départ et d'arrivée, et ainsi de repérer plus facilement dans la visualisation les groupes d'éléments appelants et d'éléments appelés qui leur sont respectivement associés. De plus, comme il le sera expliqué sous peu, la couleur associée à chaque extrémité de la courbe représente le ratio entre la quantité de liens entrant et de liens sortant de l'élément situé à cette extrémité. Afficher cette couleur sur le tiers de la courbe, à chaque extrémité, permet donc à l'utilisateur de voir clairement la couleur associée à chacune d'elle, et ainsi d'avoir une meilleure idée sur le ratio entre les liens entrants et sortants associé à chaque élément lié. La figure 4.17 montre un exemple de liens existant entre les classes d'un logiciel, avec et sans la séparation des liens en trois

segments.



(A) (B)

Figure 4.17 – Exemple de liens entre les classes d'un logiciel. Nous avons choisi les couleurs noir et jaune comme couleurs de départ et d'arrivée, respectivement. L'image A) montre le dégradé de couleurs obtenu sur les courbes lorsque nous appliquons sur celles-ci un dégradé linéaire, allant de la couleur de départ jusqu'à la couleur d'arrivée, s'effectuant sur toute la longueur de la courbe. L'image B) montre le résultat obtenu lorsque nous séparons les courbes en trois segments égaux colorés de façons différentes, comme expliqué ci-haut. La démarcation entre le segment de départ (en noir) et d'arrivée (en jaune) de chaque courbe est plus prononcée, et ceux-ci ressortent plus vivement dans la visualisation que dans l'image A). Ceci permet à l'utilisateur de repérer plus facilement ces segments, ainsi que les groupes d'éléments appelants et d'éléments appelés qui leur sont liés.

Comme mentionné à la section 4.3.2, les liens affichés dans la visualisation peuvent représenter plusieurs liens pouvant aller dans des directions opposées. Il est donc nécessaire de modifier l'algorithme que nous venons de décrire de sorte qu'il soit capable de traiter de tels liens. Pour y arriver, il nous a suffit seulement de modifier la façon d'associer une couleur à chaque extrémité. Nous choisissons toujours de façon arbitraire une couleur de départ et une couleur d'arrivée, mais au lieu de les associer directement aux extrémités correspondantes, nous les utilisons conjointement avec le degré d'entrée des éléments situés à chaque extrémité de la courbe afin de calculer une nouvelle couleur que nous associons à l'extrémité correspondante. Plus précisément, la nouvelle couleur

correspond à celle située au pourcentage donné par le degré d'entrée sur le spectre du dégradé entre la couleur de départ et la couleur d'arrivée. La figure 4.18 montre ce spectre avec les couleurs que l'on obtiendrait à chaque extrémité si le degré d'entrée était de 25% à une de celles-ci. Une fois ces nouvelles couleurs de départ et d'arrivée déterminées, l'algorithme colorie la courbe exactement de la même façon que celle décrite précédemment. Au final, les couleurs qu'on obtient pour le segment de départ et le segment d'arrivée représentent le ratio entre les liens entrant et les liens sortant de l'élément situé à cette extrémité. Le dégradé de couleurs appliqué sur la courbe représente donc le sens dominant de l'ensemble des liens contenus dans celle-ci, tandis que les couleurs aux extrémités, n'étant pas pures, indiquent avec quelle force (ou faiblesse) ce sens est dominant. La figure 4.19 montre un exemple de cela en utilisant les couleurs calculées à la figure 4.18. S'il y a autant de liens allant dans un sens que dans l'autre, signifiant qu'il n'y a pas de sens dominant, alors la courbe sera coloriée de façon uniforme avec la couleur située à 50% sur le spectre du dégradé de couleurs.

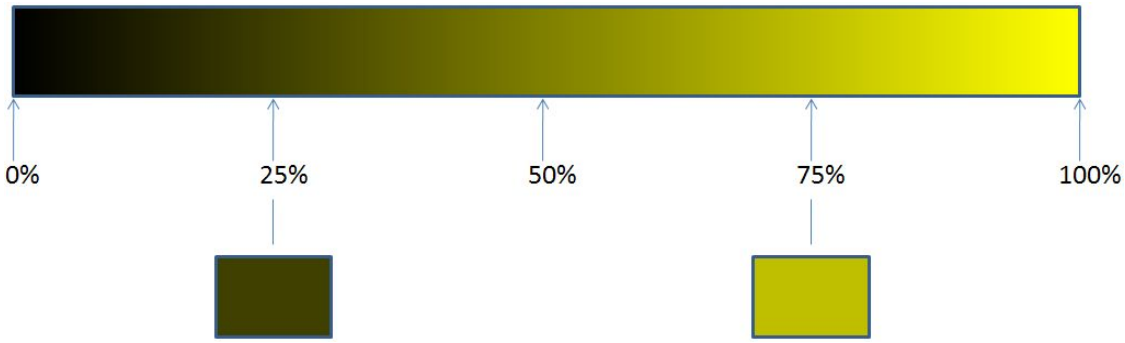


Figure 4.18 – Exemple du calcul des nouvelles couleurs de départ et d'arrivée. La bande ci-haut montre le spectre d'un dégradé de couleurs allant du noir (couleur de départ) au jaune (couleur d'arrivée). Les pourcentages situés en-dessous indiquent la couleur obtenue selon la valeur du degré d'entrée de l'élément. Par exemple, si le degré d'entrée d'un élément à une des extrémités est de 25%, alors celui de l'élément situé à l'autre extrémité est nécessairement de 75% (comme expliqué à la section 4.3.2). Les carrés de couleur situés à 25% et 75% du spectre représentent les couleurs que nous calculerions respectivement pour ces deux extrémités avec ces degrés d'entrée. Le lien montré à la figure 4.19 utilise ces couleurs comme couleur de départ et couleur d'arrivée.

Figure 4.19 – Exemple de la dégradation de couleurs d'un lien agrégé. L'élément situé à l'extrémité gauche de ce lien a un degré d'entrée de 75%, et celui de droite a un degré d'entrée de 25%. Les couleurs attribuées aux extrémités de ce lien correspondent donc à celles montrées en exemple à la figure 4.18. Le sens dominant du lien est facilement visible ; de plus, les couleurs moins vives aux extrémités (car moins pures) que celles utilisées pour les liens montrés à la figure 4.18 indiquent que l'ensemble des liens représentés par cette courbe ne vont pas tous dans le même sens. Il est à noter que ce lien a été créé artificiellement pour les besoins de l'exemple.

4.3.4.2 Variation de l'épaisseur d'une courbe en fonction du nombre de liens

Nous venons de voir à la section précédente comment afficher le sens global des groupes de liens représentés par les courbes de la visualisation en utilisant la couleur de celles-ci. Une autre donnée importante à afficher dans notre visualisation est le nombre de liens représentés par chaque courbe. L'attribut des courbes que nous avons choisi d'utiliser à cette fin est leur épaisseur, l'association entre celle-ci et le nombre de liens contenus dans la courbe étant la plus naturelle que l'on puisse imaginer. En effet, cela nous ramène à la métaphore des câbles décrite à la section 4.2, où la grosseur d'un câble (son épaisseur) est déterminée par le nombre de liens le composant. Cela suit donc la logique la plus simple et la plus naturelle que peuvent s'imaginer la plupart des utilisateurs : plus une courbe est épaisse, plus celle-ci contient de liens.

Essentiellement, la seule chose à faire pour afficher cette nouvelle information est de déterminer la façon de calculer la largeur d'une courbe en fonction du nombre de liens qu'elle représente. Premièrement, nous commençons par fixer de façon arbitraire un plafond sur l'épaisseur des courbes. Cela nous permet de s'assurer autant que possible que, peu importe le nombre de liens qu'elles représentent, les courbes ne deviendront pas

grosses au point de nuire grandement à la visualisation en créant trop d'encombrement visuel et d'occultation dans celle-ci. L'épaisseur maximale est choisie en fonction de la taille du logiciel visualisé : plus celui-ci est grand, plus l'épaisseur maximale autorisée sera élevée. Ceci s'explique par le fait que plus un logiciel est grand, plus la représentation de sa hiérarchie sera étendue. Ainsi, un lien qui semble énorme lorsqu'affiché dans une petite représentation hiérarchique aura l'air significativement plus petit s'il est affiché dans une représentation hiérarchique beaucoup plus étendue. Aussi, comme la représentation hiérarchique est plus étendue, augmenter de façon proportionnelle l'épaisseur maximale des courbes ne risque pas, en général, de créer significativement plus d'encombrement visuel ou d'occultation. La figure 4.20 montre un exemple de ce phénomène. L'épaisseur maximale idéale pour un logiciel donné doit donc être assez basse pour éviter de créer trop d'encombrement visuel dans celui-ci, mais assez grande pour que la taille des liens soit significative lorsque vue par rapport à l'ensemble de la représentation hiérarchique.

Une fois l'épaisseur maximale déterminée, nous calculons l'épaisseur de chaque courbe comme suit :

$$E = \frac{N}{MAX_N} \cdot MAX_E$$

où :

- E est l'épaisseur de la courbe
- N est le nombre de liens représentés par la courbe
- MAX_N est le nombre maximum de liens
- MAX_E est l'épaisseur maximale autorisée.

Le nombre maximum de liens peut être calculé de façon automatique ou déterminé de façon manuelle. S'il est calculé de façon automatique, alors notre algorithme trouve le plus gros lien existant entre les éléments du logiciel, incluant les paquetages, et MAX_N est affecté du nombre de liens agrégés dans celui-ci. Il est à noter que le plus gros lien

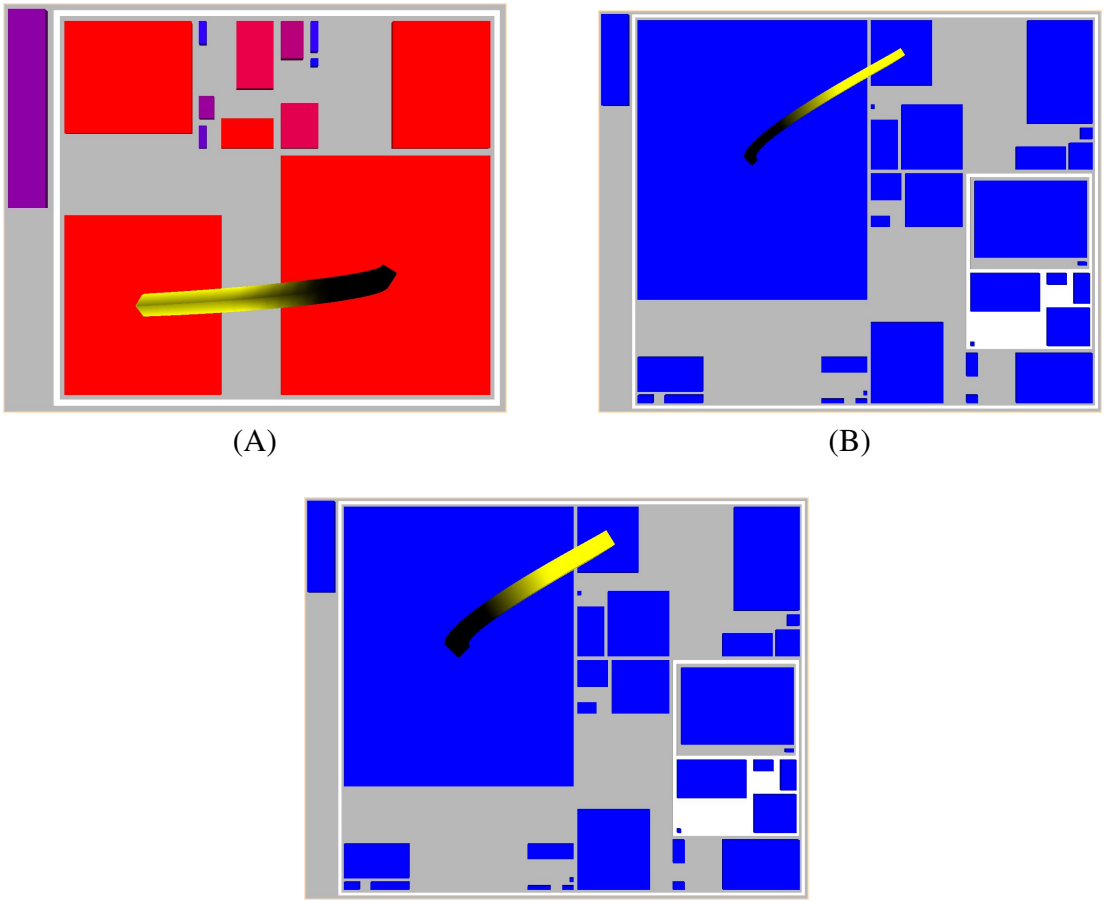


Figure 4.20 – Perception relative de l'épaisseur maximale autorisée. L'image A) montre le plus gros lien existant dans le logiciel *JHotDraw*, affiché avec l'épaisseur maximale autorisée. L'image B) montre le plus gros lien existant dans le logiciel *ArgoUML*, affiché avec la même épaisseur maximale que celle utilisée dans l'image A). Vu par rapport à l'ensemble du logiciel, le lien dans l'image B) semble nettement plus petit que celui dans l'image A), et ne représente pas adéquatement la quantité de liens qui existent entre ces deux paquetages. L'image C) montre le même lien que dans l'image B), mais après avoir doublé l'épaisseur maximale autorisée. La nouvelle taille du lien est plus significative lorsque vue par rapport à l'ensemble de la représentation hiérarchique, considérant que c'est le plus gros lien existant dans le logiciel. Cela montre l'importance d'adapter l'épaisseur maximale autorisée en fonction de la taille du logiciel visualisé.

du logiciel a de très fortes chances de se trouver entre deux paquetages de très haut niveau dans la hiérarchie. Si MAX_N est déterminé de façon manuelle, alors l'utilisateur choisit ce qu'il considère comme étant le nombre maximum de liens que peut contenir une courbe. La valeur choisie par l'utilisateur devrait refléter ce qu'il considère comme étant des liens énormes, soit par rapport au logiciel visualisé, soit dans l'absolu, au-delà desquelles il n'est plus important de faire la distinction entre les différentes tailles de liens. Par exemple, si l'utilisateur considère les liens qui en représentent 500 comme étant énormes, alors tous ceux qui dépassent cette limite seront affichés dans la visualisation avec l'épaisseur maximale. L'utilisateur pourra ainsi trouver facilement tous les liens qu'il considère énormes, et ce peu importe la taille des plus gros liens existant dans le logiciel visualisé. De plus, l'utilisateur peut adapter manuellement la valeur choisie en fonction du logiciel, un peu comme dans la façon automatique, mais en ayant la liberté de choisir une valeur étant représentative de l'ensemble des liens de taille importante existant dans le logiciel, et non pas seulement du lien le plus gros. Ceci pourrait éviter à cette valeur d'être faussée par un seul lien de taille aberrante, phénomène qui pourrait se produire avec la façon automatique. Par contre, pour choisir une valeur adéquate, l'utilisateur devrait idéalement connaître déjà suffisamment bien le logiciel qu'il visualise, ce qui va un peu à l'encontre de l'utilité de notre outil. L'avantage de la façon automatique est donc qu'elle permet à la visualisation des liens d'adjacence de s'adapter en fonction de la taille du logiciel visualisé, sans nécessiter de l'utilisateur qu'il ait au préalable une quelconque connaissance de celui-ci. En résumé, la façon automatique est idéale pour une première approche du logiciel, tandis que la façon manuelle l'est plus pour des tâches de maintenance, alors que l'utilisateur connaît mieux la structure de celui-ci. Sauf avis contraire, pour le reste du présent mémoire la façon automatique sera utilisée dans tous les exemples de liens d'adjacence qui seront présentés.

Une fois MAX_N déterminé, nous calculons l'épaisseur de la courbe avec la formule donnée ci-haut. La figure 4.21 montre quelques exemples de liens de tailles différentes, déterminées de façon automatique ou de façon manuelle. Le plus grand inconvénient que l'on a de représenter le nombre de liens avec l'épaisseur de la courbe est que cette méthode est mal adaptée pour être appliquée sur les liens existant entre les classes du

logiciel. La raison principale à cela est que les classes, contrairement aux paquetages, sont toujours placées en groupes compacts, et il en va de même pour les paquets de liens les unissant. Augmenter l'épaisseur de seulement quelques liens dans un de ces paquets créerait beaucoup d'occultation sur les autres présents dans celui-ci. C'est pourquoi, pour l'instant, nous ne traitons que des liens unitaires entre les classes, c'est-à-dire des liens qui en représentent seulement un. Une solution à cela serait d'utiliser la couleur des courbes pour afficher à la fois la quantité de liens présents dans celles-ci ainsi que la direction globale dans laquelle ils vont, comme dans [3]. Par contre, selon nous la solution idéale serait mixte : appliquer la méthode décrite dans [3] seulement sur les liens ayant au moins une classe à une de leurs extrémités, et appliquer conjointement cette méthode avec la nôtre, utilisant l'épaisseur des courbes, sur tous les liens existant entre deux paquetages. De cette façon, nous éviterions de créer trop d'occultation entre les liens. De plus, les liens entre les paquetages, paquetages pouvant être représentés chacun par une très grande entité graphique, auront une taille représentant mieux la quantité de liens réels existant entre ceux-ci.

4.3.5 Affichage des liens d'adjacence

Les sections précédentes ont décrit la façon de construire les liens ainsi que la façon de déterminer la couleur et la taille à leur donner. Pour les afficher, il ne nous reste plus qu'à évaluer un nombre arbitraire de points sur les courbes les représentant, puis à créer les polygones servant à les afficher. Tous les exemples que nous avons présentés jusqu'à maintenant, et tous ceux que nous présenterons jusqu'à la fin de ce mémoire évaluent 31 points sur chaque courbe, incluant les points de départ et d'arrivée, et donc utilisent 30 segments pour l'approximer. Ce nombre de segments est suffisant pour que l'apparence des courbes ne soit pas trop « cassée », tout en nous permettant d'obtenir des performances raisonnables lorsqu'on les affiche en temps réel. Chaque courbe est constituée de quatre faces, au lieu des six ou huit faces qui seraient nécessaires pour lui donner un aspect cylindrique raisonnable. Utiliser seulement quatre faces permet d'obtenir de meilleures performances pour le rendu, sans compter que même si les liens ont un aspect cubique au lieu d'un aspect cylindrique, cela n'a aucun impact significatif

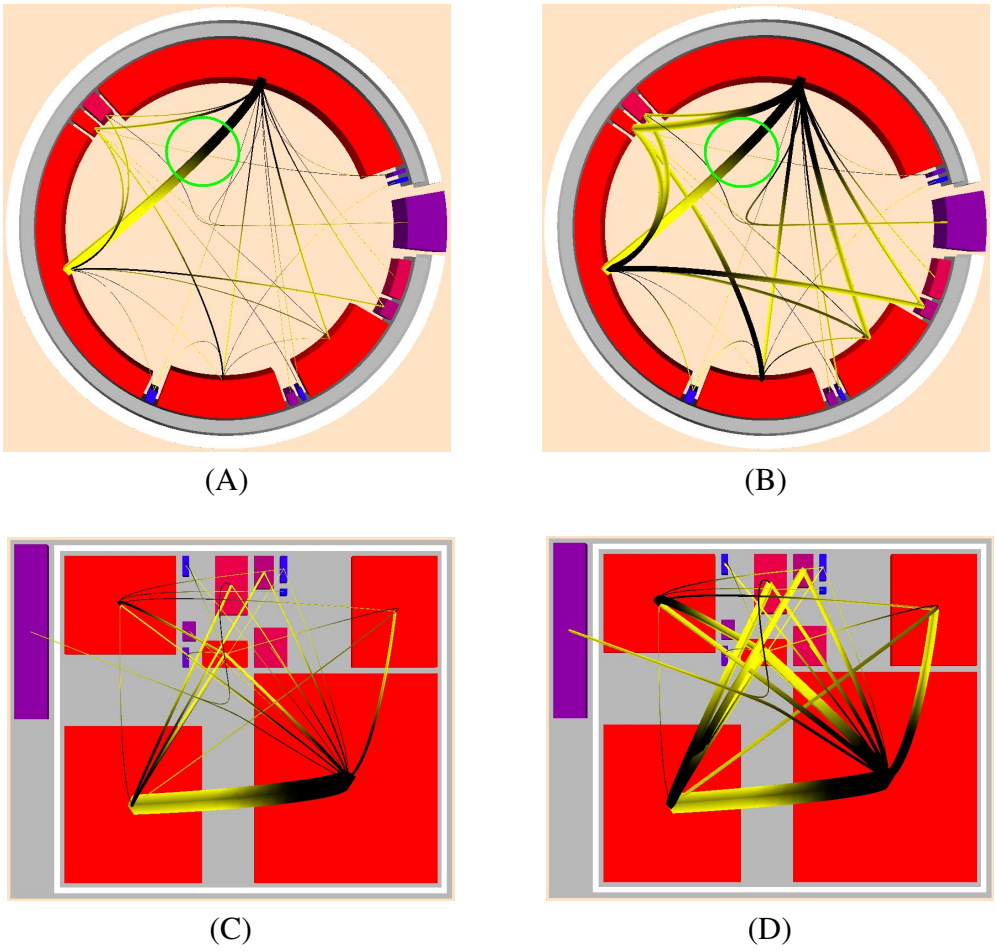
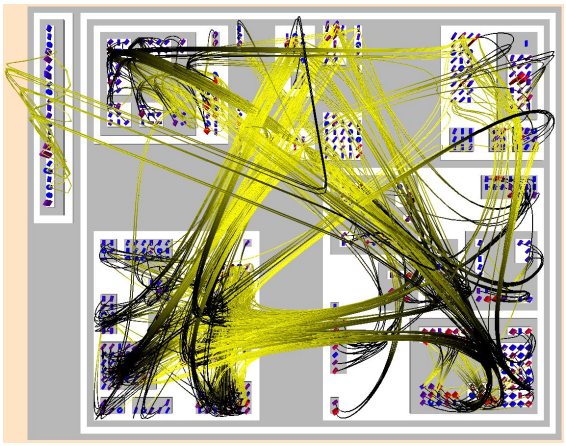


Figure 4.21 – Comparaison entre les façons automatique et absolue de déterminer MAX_N. Les images A) et B) montrent les liens existant entre des paquetages situés au même niveau dans la hiérarchie du logiciel, niveau qui est assez élevé. L'image A) utilise la façon automatique pour calculer MAX_N, tandis que l'image B) utilise la façon manuelle, avec $MAX_N = 100$. Dans chacune de ces images, le lien encerclé en vert est le plus gros lien existant dans tout le logiciel, et donc son épaisseur correspond à MAX_E. Dans l'image A), tous les liens affichés sont très minces comparés au lien le plus gros. Ce lien dominant nous permet de constater que la majorité des liens à ce niveau se font entre les deux paquetages unis par celui-ci. La plupart des liens dans l'image B) sont nettement plus larges, car ils sont comparés avec ce que l'utilisateur considère comme étant un lien de grande taille, et non avec le plus gros lien présent dans le logiciel. Il est ainsi plus facile pour l'utilisateur de repérer les liens considérés comme importants selon ses critères, et ce peu importe comment ceux-ci se comparent avec le lien le plus gros du logiciel. Par contre, en procédant ainsi il devient impossible de comparer avec certitude deux courbes ayant atteint l'épaisseur maximale autorisée, car celles-ci, même si elles sont d'épaisseur égale, pourraient très bien représenter deux quantités de liens largement différentes, allant au-delà de MAX_N. Les images C) et D) montrent les mêmes liens, mais en utilisant le placement *Treemap*.

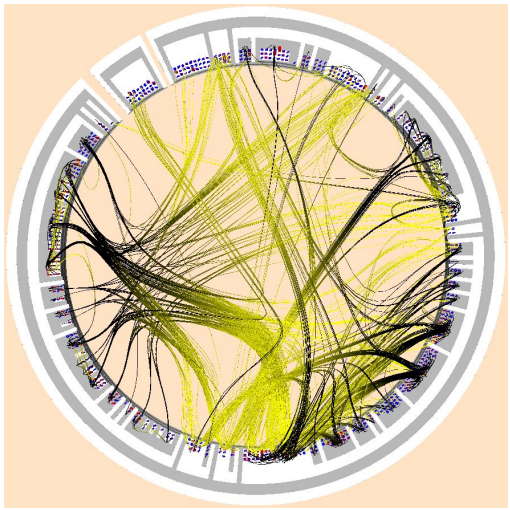
sur la qualité de la visualisation. La figure 4.22 montre un exemple des liens d'adjacence obtenus dans chaque type de placement, avec noir comme couleur de départ et jaune comme couleur d'arrivée.

4.4 Réduction de l'encombrement visuel par métaheuristique

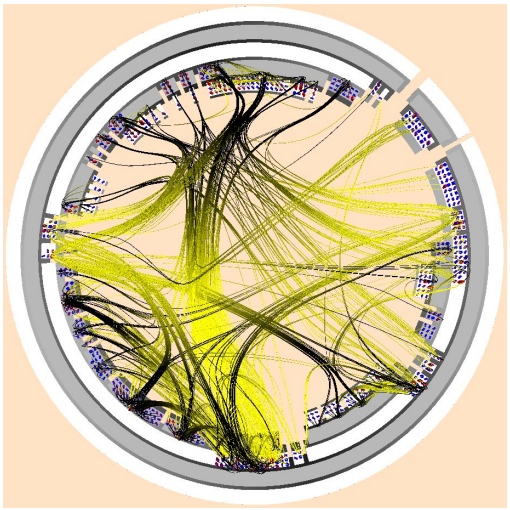
Il est évident que le type de placement utilisé pour afficher la hiérarchie d'un logiciel a une grande influence sur la visualisation des liens d'adjacence obtenue. Par exemple, comme le montre la figure 4.22, la visualisation des liens d'adjacence que nous obtenons en utilisant le placement *Treemap*, décrit à la section 3.2, est largement différente de celle obtenue en utilisant le placement Colisée, décrit à la section 3.4. De plus, avec un type de placement donné, l'ordre dans lequel on place les éléments de la hiérarchie peut aussi avoir une certaine influence, positive ou négative, sur la visualisation des liens d'adjacence obtenue. Prenons par exemple ce qui arrive dans le placement Colisée lorsque nous plaçons côte à côte deux paquetages dont les sous-éléments sont fortement liés. Dans cette situation, la courbure des liens entre ces éléments est très prononcée, comme le montre la figure 4.23 : ceci les rend plus difficiles à suivre, comme le stipule la loi de la bonne continuité, une des lois principales de la Gestalt. Une façon d'améliorer la visualisation de ces liens d'adjacence serait d'éloigner ces paquetages, ce qui réduirait la courbure des liens à mesure que la distance angulaire les séparant augmente. Le cas idéal serait alors qu'ils soient placés l'un en face de l'autre (distance de 180^{o}), réduisant ainsi au maximum la courbure des liens. Lorsque deux éléments sont situés face à face sur un cercle, la distance les séparant en ligne droite correspond au diamètre du cercle, ce qui est en fait la distance maximale pouvant les séparer. Ainsi, vouloir réduire la courbure des liens entre deux groupes d'éléments revient donc à vouloir augmenter autant que possible la distance en ligne droite les séparant. Cette observation sera utilisée comme critère fondamental dans l'évaluation de la valeur d'un placement Colisée, comme il sera expliqué sous peu. Il est à noter que tous les exemples présentés dans cette section utilisent le logiciel *ArgoUML*, qui est constitué de 2291 classes, réparties dans 142 paquetages qui sont étalés sur 10 niveaux de profondeur (incluant la racine).



(A)

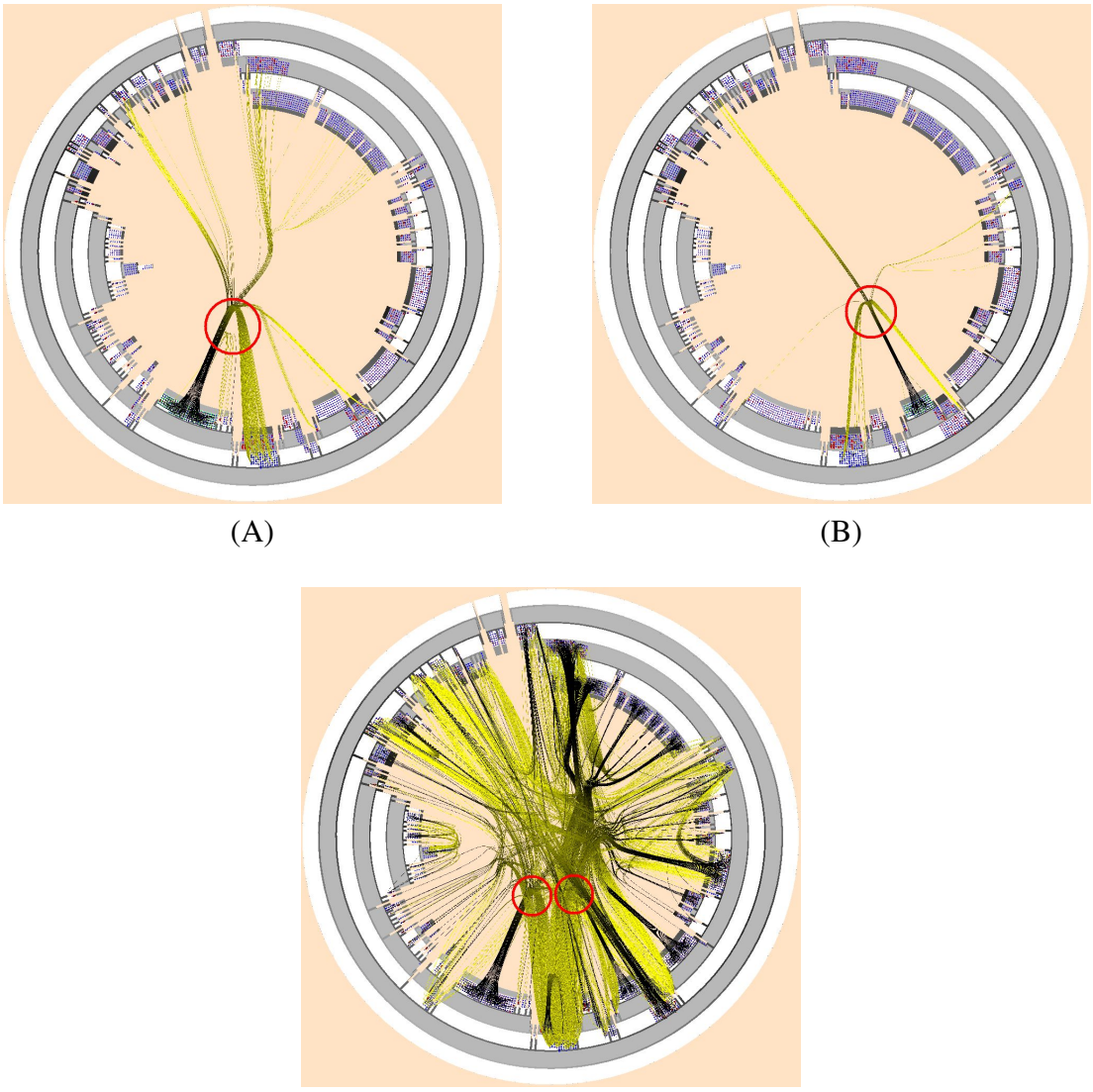


(B)



(C)

Figure 4.22 – Exemples des liens d'adjacence obtenus dans chaque type de placement. L'image A) montre les liens d'adjacence obtenus dans le placement *Treemap*, l'image B) ceux obtenus dans le placement radial et l'image C) ceux obtenus dans le placement Colisée.



(A) (B)

(C)

Figure 4.23 – Exemple de liens avec une courbure très prononcée dans le placement Colisée. Les images A) et B) montrent chacune un groupe de classes plus ou moins lié avec un groupe de classes voisin. Le chemin de nœuds que doivent emprunter les liens unissant des éléments voisins les oblige à aller plus ou moins vers le centre du placement avant de revenir pour rejoindre l'élément correspondant. S'ils veulent respecter les règles de l'algorithme *HEB*, les liens ne peuvent donc pas aller plus directement d'un élément à un autre, les obligeant ainsi à avoir une courbure prononcée. L'image C) montre l'ensemble des liens présents dans le placement : il est difficile de voir que certains paquets de liens commençant à côté du groupe de classes situé dans le bas du placement reviennent abruptement vers celui-ci.

Par défaut, nos algorithmes de placement choisissent de façon arbitraire l'ordre dans lequel placer les éléments, en choisissant toujours le même ordre. Nous les avons donc modifiés afin qu'ils essaient de trouver l'ordre de placement des éléments de la hiérarchie qui offre la meilleure visualisation possible des liens d'adjacence. L'exemple donné précédemment, celui sur les paquetages fortement liés qui sont placés côte à côte dans le placement Colisée, représente pour ce type de placement le principal problème causé par un choix arbitraire de l'ordre de placement des paquetages. Le placement idéal dans cette situation serait donc celui qui atténue autant que possible ce problème, bref celui qui éloigne autant que possible les paquetages fortement liés. Il est à noter que, dans chaque placement utilisé, nous ne nous attardons pas à modifier l'ordre dans lequel les classes sont placées, car celles-ci sont toujours rassemblées en groupes compacts, et ainsi modifier leur ordre de placement n'influencerait pas de manière significative la visualisation des liens d'adjacence.

Les contraintes imposées par la hiérarchie font qu'on peut seulement changer l'ordre avec lequel l'algorithme place des paquetages frères dans leur parent. Malgré tout, même avec ces contraintes, et même avec un système de taille relativement petite, comme par exemple *JHotDraw*, le nombre de placements différents possibles est bien trop grand pour qu'on puisse tous les évaluer et ensuite utiliser celui qu'on estime être le meilleur. C'est pourquoi nous avons décidé d'utiliser l'algorithme du recuit simulé [14, 28], une métaheuristique, afin non pas de trouver nécessairement le meilleur placement possible, mais d'en trouver au moins un très bon.

L'algorithme du recuit simulé est inspiré du recuit réel, procédé de métallurgie qui consiste à contrôler le refroidissement d'un métal avec des apports de chaleur externes, dans le but de minimiser l'énergie du matériau. Par analogie avec le recuit réel d'une pièce de métal, l'objectif de cet algorithme est de trouver la solution ayant l'énergie E la plus basse, où l'énergie d'une solution donnée correspond à la valeur qu'on lui accorde. Pour ce faire, celui-ci parcourt l'espace de solutions, à partir d'une solution initiale choisie arbitrairement, en se déplaçant aléatoirement d'une solution voisine à une autre. Il utilise un paramètre T représentant la température courante, qui décroît plus ou moins vite selon le degré de précision voulu dans la recherche d'une bonne solution et selon le

temps que l'on est prêt à accorder à l'algorithme pour s'exécuter. La température T influence la probabilité qu'a l'algorithme de faire la transition vers une solution voisine de la solution courante qui est moins bonne que cette dernière, c'est-à-dire qui a une énergie plus élevée : plus la température est élevée, plus cette transition a des chances de s'effectuer. Une température élevée permet donc à l'algorithme d'explorer un large spectre de l'ensemble des solutions, ce qui permet d'éviter de rester pris dans un optimum local. Ensuite, à mesure que la température diminue, l'algorithme réduit son champ de recherche autour de la solution courante, la transition de moins en moins probable vers les solutions voisines l'empêchant d'aller plus loin dans l'espace des solutions.

Le pseudo-code 4.1 décrit l'algorithme de recuit simulé utilisé dans notre outil, dans lequel :

- T est la température courante.
- t_{init} est la température initiale.
- t_{min} est la température la plus basse jusqu'à laquelle on permet de descendre.
- S_{init} est la solution initiale.
- S_C est la solution courante.
- E_C est l'énergie (la valeur) de la solution courante.
- S_V est la solution voisine.
- E_V est l'énergie (la valeur) de la solution voisine.
- S_M est la meilleure solution trouvée jusqu'à maintenant.
- E_M est l'énergie (la valeur) de la meilleure solution trouvée jusqu'à maintenant.
- p est la probabilité d'acceptation d'une solution voisine de moins bonne valeur que la solution courante, où $p \in [0,1]$.
- k est un nombre entier positif choisi arbitrairement, qui représente le nombre de solutions essayées à chaque pas de température.

- α est un nombre réel positif choisi arbitrairement, où $\alpha \in [0,1]$.
- La fonction random() retourne un nombre réel aléatoire r, où $r \in [0,1]$.

Chaque itération de la boucle principale de l'algorithme correspond à une température donnée. À chacun de ces pas de température, l'algorithme essaie k solutions différentes avant de passer au pas de température suivant. La valeur de α indique à quelle vitesse nous voulons abaisser la température : une valeur élevée de α la fait diminuer tranquillement, laissant à l'algorithme plus de temps pour trouver la meilleure solution possible. À chaque fois que l'algorithme se déplace à une solution voisine, il la compare toujours avec la solution à laquelle il se trouve à ce moment-là. Si la solution voisine est meilleure, c'est-à-dire que son énergie est plus basse, alors il se déplace tout de suite vers celle-ci : sinon, il évalue la probabilité p de se déplacer vers cette solution, même si celle-ci est moins bonne que la solution courante. Cette probabilité est influencée par la différence d'énergie existant entre la solution courante et la solution voisine, et surtout, par la valeur de la température T : plus la température est élevée, plus la probabilité d'accepter de mauvaises solutions sera élevée aussi. Ensuite, l'algorithme vérifie si la nouvelle solution courante est meilleure que la meilleure solution trouvée jusqu'à maintenant et, si oui, la prend en note. Finalement, une fois que la température a été abaissée en-dessous du niveau minimal permis, alors l'algorithme termine et retourne la meilleure solution qu'il a trouvée durant toute son exécution.

Deux aspects importants sont à définir dans l'algorithme que nous venons de décrire :

1. Qu'est-ce qu'une solution ?
2. Qu'est-ce qu'une solution voisine de la solution courante ?

Ces aspects varient selon le problème que nous cherchons à optimiser avec cet algorithme. Dans notre cas, une solution correspond à une hiérarchie de paquetages où les groupes de paquetages frères ont un ordre de placement précis, tandis qu'une solution voisine correspond à la même hiérarchie que la solution courante, mais où deux paquetages frères quelconques ont été échangés dans l'ordre de placement de leur groupe.

Algorithm 4.1 Algorithme du recuit simulé utilisé dans notre système

$T \leftarrow t_{init}$
$S_C \leftarrow S_{init}$
$E_C \leftarrow$ évaluer(S_C)
$S_M \leftarrow S_C$
$E_M \leftarrow E_C$

while $T \geq t_{min}$ **do**

    for $i = 0 \rightarrow k$ **do**
        $S_V \leftarrow$ trouver_voisin(S_C)
        $E_V \leftarrow$ évaluer(S_V)

        if $E_V < E_C$ **then**
            $S_C \leftarrow S_V$
            $E_C \leftarrow E_V$
        else
            $p \leftarrow exp((E_C - E_V)/T)$

            if random() $\leq p$ **then**
                $S_C \leftarrow S_V$
                $E_C \leftarrow E_V$
            end if

        end if

        if $E_C < E_M$ **then**
            $S_M \leftarrow S_C$
            $E_M \leftarrow E_C$
        end if

    end for

    $T \leftarrow T \bullet \alpha$
end while

return S_M

Pour obtenir une solution voisine, nous commençons par sélectionner aléatoirement un paquetage parmi ceux de la hiérarchie qui possèdent au moins deux sous-paquetages. Ensuite, nous sélectionnons aléatoirement deux paquetages parmi les sous-paquetages de celui-ci, et nous inversons leur position dans la liste qui représente l'ordre dans lequel ces sous-paquetages sont placés à l'intérieur de leur parent. Les placements utilisés dans notre système définissent ces aspects exactement comme nous venons de le décrire. La seule différence entre ces placements est que, dans le placement *Treemap*, les groupes de classes sont considérés comme un sous-paquetage supplémentaire, mais cela ne change en rien la façon de définir ce qu'est une solution ni la façon de trouver ses voisins.

Un autre aspect important à définir pour exécuter cet algorithme est la façon dont nous évaluons une solution, c'est-à-dire comment nous déterminons son énergie. Contrairement aux deux aspects mentionnés précédemment, la façon de définir ce qu'est une bonne solution, c'est-à-dire ce qu'est un bon placement, peut changer radicalement d'un type de placement à un autre. Pour commencer, reprenons l'exemple avec le placement Colisée que nous avons décrit au début de cette section. La conclusion que nous en avons tirée est qu'il faut éloigner autant que possible les paquetages du placement qui sont fortement liés. Ainsi, nous mesurons la valeur d'un placement Colisée en additionnant la distance, en ligne droite, existant entre chacune des paires de classes liées dans le logiciel : plus la distance totale est élevée, plus la valeur du placement l'est aussi. Nous préférons utiliser la distance en ligne droite plutôt que la longueur de la courbe liant deux classes pour deux raisons :

1. Avoir à déterminer la longueur de toutes les courbes présentes dans la visualisation, pour chaque solution évaluée par l'algorithme, ferait exploser le temps qu'il prend pour s'exécuter.

2. La distance en ligne droite entre deux éléments est plus proche de la distance en degré qui les sépare que la longueur de la courbe existant entre eux.

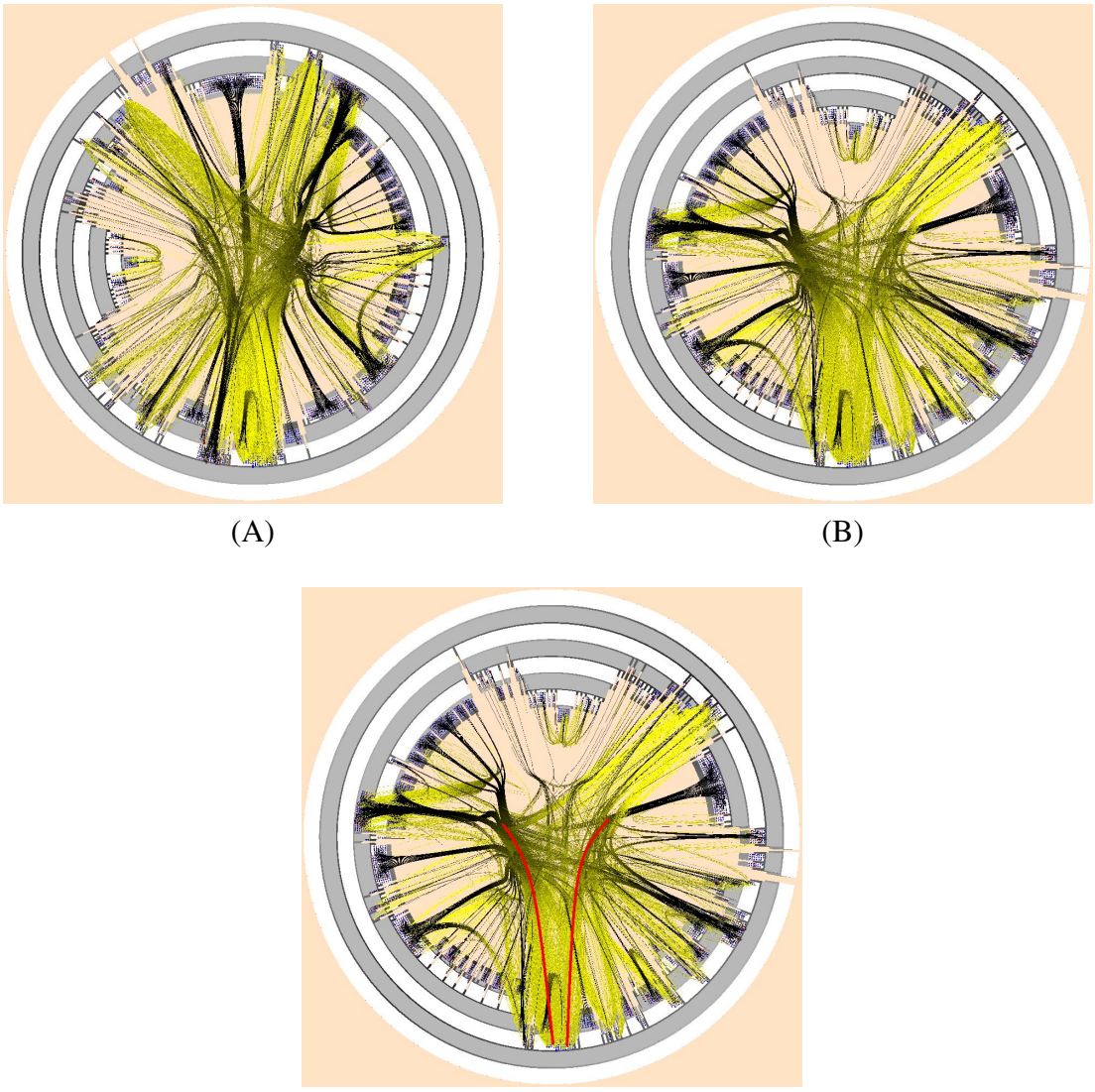
La distance en ligne droite est ainsi non seulement suffisante pour obtenir de bons résultats, mais est en fait même plus appropriée que la longueur des courbes pour atténuer le problème de courbures prononcées dans les liens d'adjacence du placement

Colisée. Il est à noter que, étant donné que nous considérons un placement ayant une valeur supérieure à celle d'un autre comme étant meilleur que ce dernier, nous devons inverser, dans l'algorithme du recuit simulé, la façon de comparer deux énergies afin de favoriser celle qui est la plus élevée des deux. Dans cette situation, nous ne cherchons donc pas à obtenir la solution ayant la plus petite énergie mais, au contraire, celle qui a la plus grande. La figure 4.24 montre le meilleur placement Colisée que nous avons obtenu après avoir exécuté cet algorithme.

Le placement radial étant très similaire au placement Colisée, le problème de courbure prononcée des liens est également présent dans celui-ci. La façon d'évaluer la valeur d'un placement radial est donc la même que celle que nous venons de décrire pour le placement Colisée. La figure 4.25 montre l'amélioration obtenue avec l'utilisation du recuit simulé dans le placement radial.

Le placement *Treemap* étant largement différent du placement Colisée et du placement radial, la façon d'évaluer la valeur d'un placement de ce type n'est donc pas forcément la même que dans les deux autres. Par contre, comme nous l'avons mentionné dans la section 3.2.1, il est important que le placement *Treemap* laisse un peu d'espace vide entre les paquetages afin d'éviter qu'il y ait trop d'encombrement dans la visualisation lorsque nous y ajoutons les liens d'adjacence. Éloigner autant que possible les paquetages fortement liés suit la même logique, car placer deux paquetages fortement liés côte à côte créerait un gros amas de liens concentrés dans une zone restreinte, les rendant difficiles à discerner. Ce problème est en fait ce que nous cherchions à atténuer autant que possible en ajoutant des espaces dans le placement *Treemap* : il serait donc contre-productif de ne pas utiliser le recuit simulé pour essayer de les éloigner davantage. C'est pourquoi nous évaluons la valeur d'un placement *Treemap* de la même façon que celle d'un placement Colisée et d'un placement radial, c'est-à-dire en favorisant les placements où la distance totale entre les éléments liés est élevée. La figure 4.26 montre la différence entre le résultat obtenu avec un recuit simulé évaluant la valeur d'un placement *Treemap* de cette façon et avec un autre qui, au contraire, cherche à rapprocher autant que possible les éléments liés.

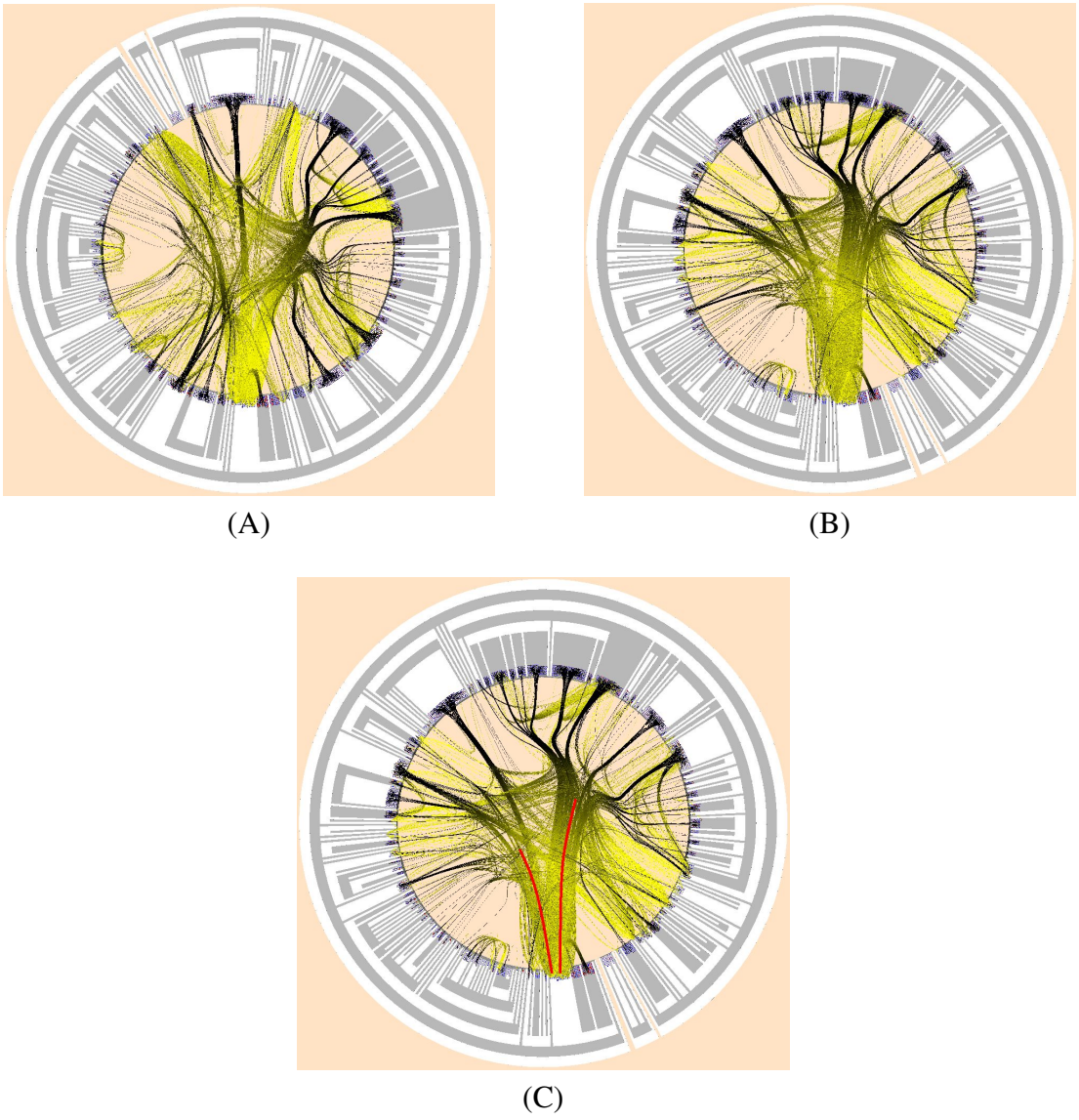
Tous les placements que nous avons obtenus avec le recuit simulé utilisaient les va-



(A) (B)

(C)

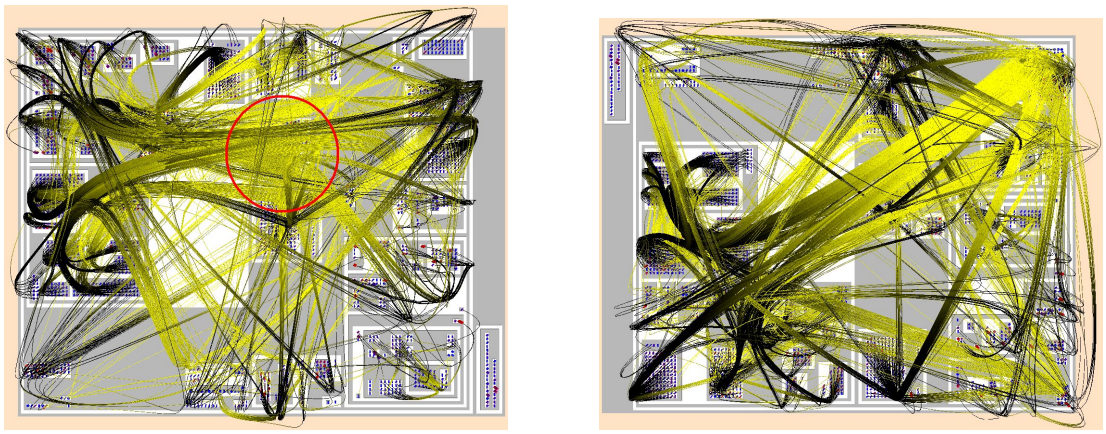
Figure 4.24 – Application du recuit simulé sur le placement Colisée. L'image A) montre le placement Colisée obtenu par défaut. L'image B) montre celui qu'on obtient après avoir appliqué l'algorithme du recuit simulé. L'image C) montre les deux paquets principaux de liens allant vers le groupe de classes situé dans le bas du placement. La direction de chacun de ces paquets de liens est plus facile à voir que dans l'image A).



(A) (B)

(C)

Figure 4.25 – Application du recuit simulé sur le placement radial. L'image A) montre la placement obtenu par défaut. L'image B) montre celui qu'on obtient après avoir appliqué le recuit simulé. Encore une fois, l'image C) montre que les deux paquets principaux de liens allant vers le groupe de classes situé dans le bas du placement sont plus facile à voir que dans l'image A). De plus, dans ce cas-ci les éléments appelants ont été regroupés en majeure partie dans le haut du placement, et ceux appelés dans le bas. Cette séparation rend plus facile la tâche de distinguer les groupes d'éléments appelants des groupes d'éléments appelés.



(A) (B)

Figure 4.26 – Application du recuit simulé sur le placement *Treemap*. L'image A) montre le résultat obtenu lorsqu'on applique sur le placement *Treemap* un recuit simulé qui essaie de rapprocher autant que possible les éléments liés, ce qui pourrait être une façon simple de réduire le nombre de croisements entre les liens. Le cercle en rouge montre un amas de liens qui s'est formé au centre du placement, dans lequel les paquets de liens, ainsi que leur direction, sont difficiles à distinguer. L'image B) montre le résultat obtenu lorsqu'on applique sur le placement *Treemap* un recuit simulé qui, à l'inverse de celui utilisé dans l'image A), essaie d'éloigner autant que possible les éléments liés. L'amas de liens présent dans l'image A) s'est grandement estompé : les paquets principaux de liens sont faciles à distinguer, de même que leur direction. Il est aussi plus facile de distinguer les groupes d'éléments appelants des groupes d'éléments appelés.

leurs suivantes pour les paramètres de l'algorithme :

- $T = 300\ 000$
- $t_{min} = 1$
- $\alpha = 0.99$
- $k = 250$

Avec ces valeurs, le recuit simulé, appliqué à *ArgoUML*, a pris environ 35 minutes pour terminer son exécution avec les placements *Treemap* et radial, et environ 1 heure et demie avec le placement Colisée. L'ordinateur que nous avons utilisé pour exécuter cet algorithme est muni d'un processeur Core 2 Duo cadencé à 2.67 GHz.

4.5 Analyse de l'information par méthodes d'interaction dynamiques

Tous les exemples de la version finale de la visualisation présentés jusqu'à maintenant, dans lesquels les liens d'adjacence sont affichés en même temps que la hiérarchie du logiciel, présentent le logiciel dans son ensemble en affichant le niveau de détails le plus précis offert par notre système, c'est-à-dire les classes. Regarder l'ensemble des liens d'adjacence en utilisant une visualisation aussi détaillée est efficace pour voir les tendances générales dans les liens ainsi que pour voir les groupes importants d'éléments appelants ou appelés, bref pour nous donner une vue d'ensemble du logiciel. Par contre, plusieurs tâches d'analyse, pour être menées à terme, requièrent de l'utilisateur qu'il se concentre sur des parties précises du logiciel, chose qui lui est difficile de faire lorsque tous les liens d'adjacence lui sont présentés simultanément. Cette difficulté est principalement due à l'encombrement visuel que cet affichage simultané des liens génère dans la visualisation. Afin de pouvoir cibler des parties précises du logiciel, en faisant ressortir clairement les détails les concernant, celui-ci doit pouvoir filtrer l'information qui lui est présentée, et ce autant au niveau des liens d'adjacence qu'au niveau des liens hiérarchiques. Pour ce faire, notre système offre à l'utilisateur diverses façons de modifier la visualisation qui lui est présentée et d'interagir avec elle, tout cela en temps réel.

L'utilisateur peut donc filtrer les informations qui ne l'intéressent pas afin de se concentrer sur celles qui peuvent l'aider dans la tâche qu'il doit accomplir. La section 4.5.1 explique comment l'utilisateur peut modifier le niveau de détails offert par la représentation hiérarchique ainsi que l'effet que cela produit sur les liens d'adjacence. La section 4.5.2 explique comment celui-ci peut cibler les parties du logiciel qui l'intéressent en les sélectionnant et en y appliquant certains filtres. La section 4.5.3 explique comment l'utilisateur peut sélectionner les liens d'adjacence qui l'intéressent. Finalement, la section 4.5.4 explique le mécanisme de filtrage entre les liens internes et les liens externes.

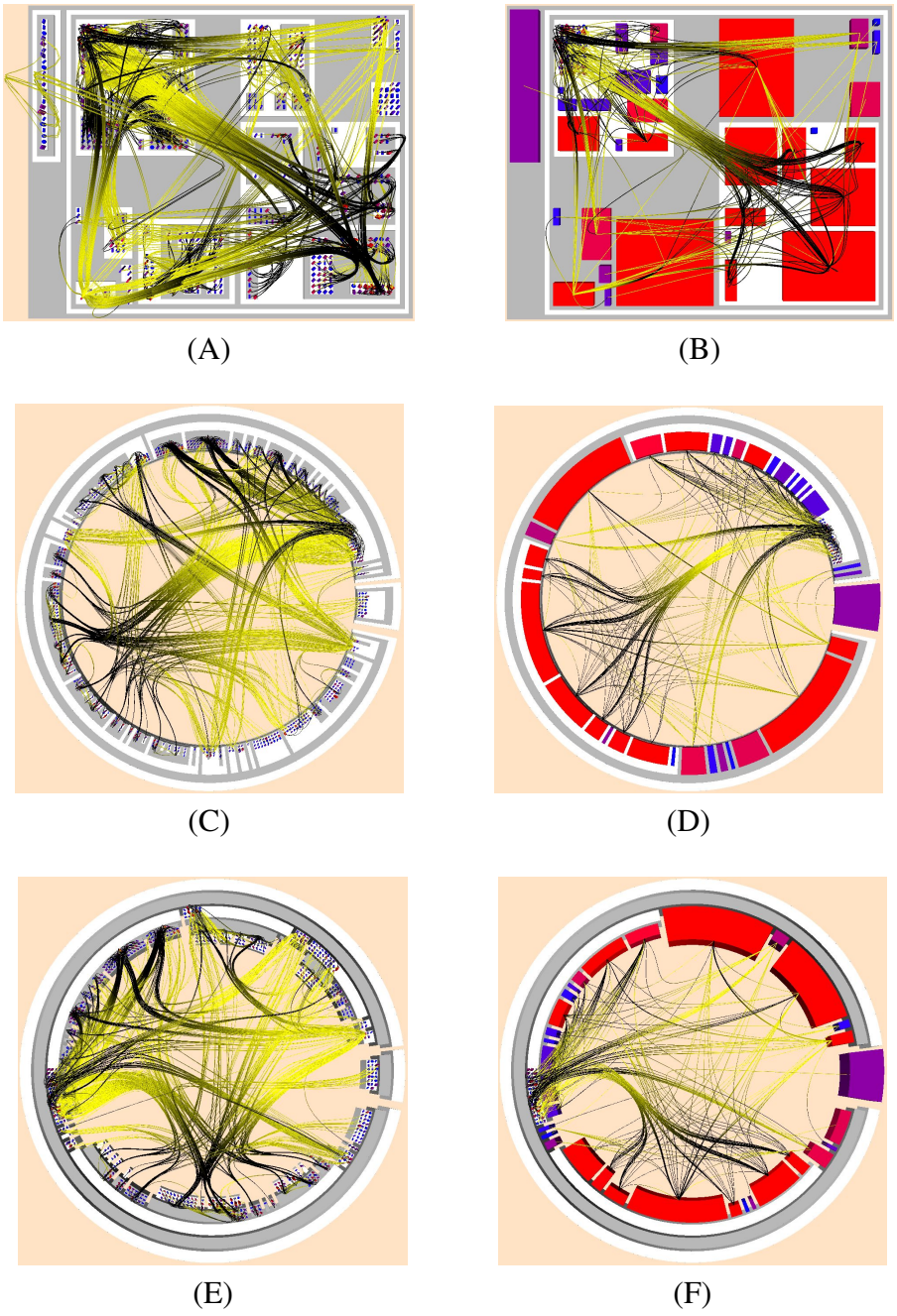
4.5.1 Modification du niveau de détails offert par la représentation hiérarchique

Comme nous l'avons expliqué à la section 2.5, la version de VERSO à partir de laquelle nous avons développé notre système offrait déjà un mécanisme permettant de fermer ou d'ouvrir les paquetages, et ainsi de modifier le niveau de détails offert par la représentation hiérarchique. Par contre, ce mécanisme ne permettait à l'utilisateur que de fermer ou d'ouvrir l'ensemble des paquetages situés à un niveau donné de la hiérarchie. Nous l'avons donc modifié de sorte que l'utilisateur puisse ouvrir ou fermer un paquetage en particulier, lui permettant ainsi de modifier le niveau de détails seulement pour les paquetages qui l'intéressent dans la représentation hiérarchique. Par exemple, l'utilisateur pourrait commencer par utiliser une représentation hiérarchique montrant seulement les paquetages principaux du logiciel, et ensuite d'investiguer ceux qui l'intéressent en n'ouvrant que ceux-ci, répétant le même procédé avec leurs sous-paquetages. Ainsi, l'utilisateur peut explorer seulement les branches de la hiérarchie qui l'intéressent, sans être importuné par les petits détails contenus dans les autres.

Nous avons déjà expliqué à la section 4.3.3.1.3 que, lorsqu'un paquetage est fermé, celui-ci est considéré comme un élément de base dans la représentation hiérarchique, et peut donc être lié aux autres éléments de base présents dans celle-ci, c'est-à-dire les classes et les autres paquetages fermés. Les liens impliquant au moins un paquetage n'existent pas réellement dans le logiciel, car seules les classes peuvent être liées entre elles : les paquetages, quant à eux, peuvent seulement être liés à d'autres éléments par

l'entremise de leurs classes descendantes, liens que nous qualifions d'implicites. Nous devons donc déterminer comment créer ces liens implicites à partir des liens explicites existant entre les classes du logiciel. La première étape est de déterminer l'ensemble des liens explicites qu'un lien implicite doit représenter. Ensuite, afin d'offrir une visualisation des liens d'adjacence qui soit cohérente, les liens implicites que l'on crée doivent être obtenus en fusionnant, d'une façon ou d'une autre, les liens explicites qu'ils représentent.

Considérons pour commencer les liens unissant deux paquetages. Le lien implicite qui existe entre deux paquetages représente l'ensemble des liens existant entre leurs descendants. Plus précisément, celui-ci représente tous les liens existant entre une classe descendant d'un des paquetages et une autre descendant de l'autre. Les liens implicites que nous créons entre deux paquetages ont donc de très fortes chances de représenter plusieurs liens. Nous avons déjà passé en revue à la section 4.3.4 la problématique d'afficher des liens qui en représentent plusieurs. Il suffit donc, lorsqu'on crée un de ces liens implicites, de comptabiliser le nombre de liens allant dans chacune des directions entre les deux paquetages. Une fois ces valeurs calculées, l'algorithme utilisé pour créer les courbes représentant les liens va pouvoir traiter ces liens implicites sans problème, en leur donnant une couleur et une taille adéquate selon l'ensemble de liens qu'ils représentent. Le principe pour créer les liens implicites existant entre un paquetage et une classe est exactement le même : cela revient essentiellement à la même chose que de créer un lien entre deux paquetages où l'un d'eux contient seulement une classe. La figure 4.27 montre, pour chaque type de placement, la comparaison entre la visualisation obtenue lorsqu'on utilise le niveau de détails le plus élevé possible dans la représentation hiérarchique et celle qu'on obtient avec un niveau de détails plus bas. Finalement, lorsqu'on ouvre et ferme des paquetages, il est important d'enlever ou d'ajouter des liens dans la visualisation selon les éléments qui sont devenus cachés ou qui sont devenus visibles, de sorte que la visualisation des liens d'adjacence soit en tout temps cohérente avec les éléments affichés par la représentation hiérarchique.



(A) (B)

(C) (D)

(E) (F)

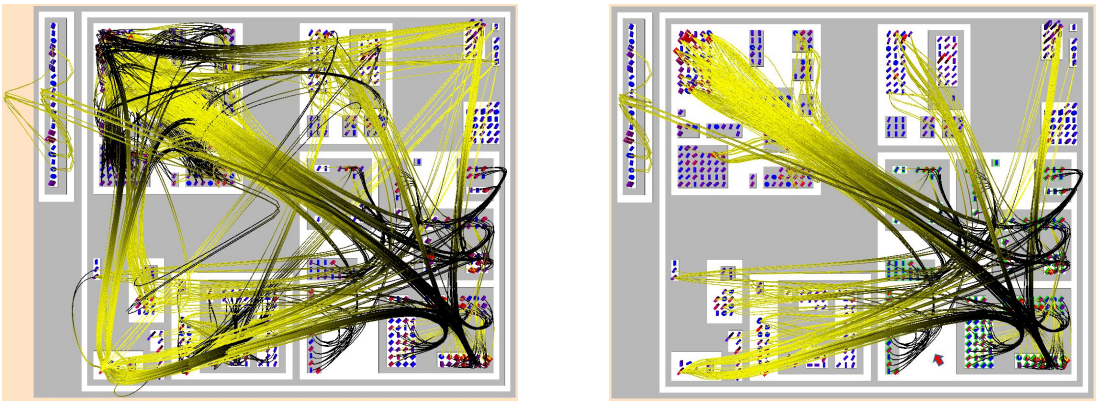
Figure 4.27 – Comparaison entre différents niveaux de détails dans la représentation hiérarchique. Chaque paire d'images montre le même logiciel en utilisant la même représentation hiérarchique, mais celles-ci utilisant des niveaux de détails différents. Dans chaque cas, l'image de gauche utilise le niveau de détails le plus élevé offert par la représentation hiérarchique, tandis que plusieurs paquetages ont été fermés dans celle de droite.

4.5.2 Sélection d'éléments dans la représentation hiérarchique

Afin de permettre à un utilisateur d'investiguer plus en détail certaines parties du logiciel visualisé, une première fonctionnalité de base que nous avons ajoutée à notre système est la sélection d'éléments dans la représentation hiérarchique. Plusieurs mécanismes de sélection sont offerts à l'utilisateur lorsque celui-ci veut sélectionner un ou plusieurs éléments. Premièrement, celui-ci peut tout simplement cliquer sur un élément de base, c'est-à-dire sur une classe ou un paquetage fermé, pour le sélectionner. L'utilisateur peut aussi cliquer sur un paquetage ouvert afin de sélectionner tous les éléments de base descendant de ce paquetage qui sont présentement affichés dans la représentation hiérarchique : cela équivaut à sélectionner le contenu du paquetage, et non le paquetage en lui-même. L'utilisateur peut aussi ajouter à sa sélection d'autres éléments, en cliquant dessus l'un après l'autre tout en maintenant une touche précise enfoncée, ce qui lui permet de sélectionner tous les éléments souhaités dans la visualisation. Finalement, afin de rendre la sélection d'éléments plus naturelle, nous avons aussi ajouté la possibilité de sélectionner des éléments en créant une zone de sélection rectangulaire avec la souris, de la même façon qu'on sélectionnerait des fichiers dans un répertoire.

Une fois que l'utilisateur a sélectionné les éléments qu'il souhaitait, nous filtrons les liens d'adjacence afin d'afficher seulement ceux qui sont reliés à au moins un des éléments sélectionnés. Ce mécanisme de sélection des éléments est donc surtout utile pour sélectionner un groupe d'éléments (par exemple le contenu d'un paquetage) afin de voir les communications existant à l'intérieur des éléments du groupe (du paquetage) et celles qui existent entre ces éléments et le reste du logiciel. La figure 4.28 montre un exemple de la sélection d'éléments et de l'effet que cela a sur la visualisation.

Lorsqu'un utilisateur sélectionne plusieurs paquetages, il peut arriver que celui-ci s'intéresse seulement aux liens existant entre ceux-ci, et non pas aux liens qu'ils ont avec tout le reste du système, c'est-à-dire avec les éléments qui n'ont pas été sélectionnés. Dans cette situation, les liens que ces paquetages ont avec des éléments n'intéressant pas l'utilisateur ne font que lui nuire, en interférant avec les liens qui intéressent vraiment celui-ci. Nous avons donc ajouté un filtre permettant à l'utilisateur d'afficher seulement



(A) (B)

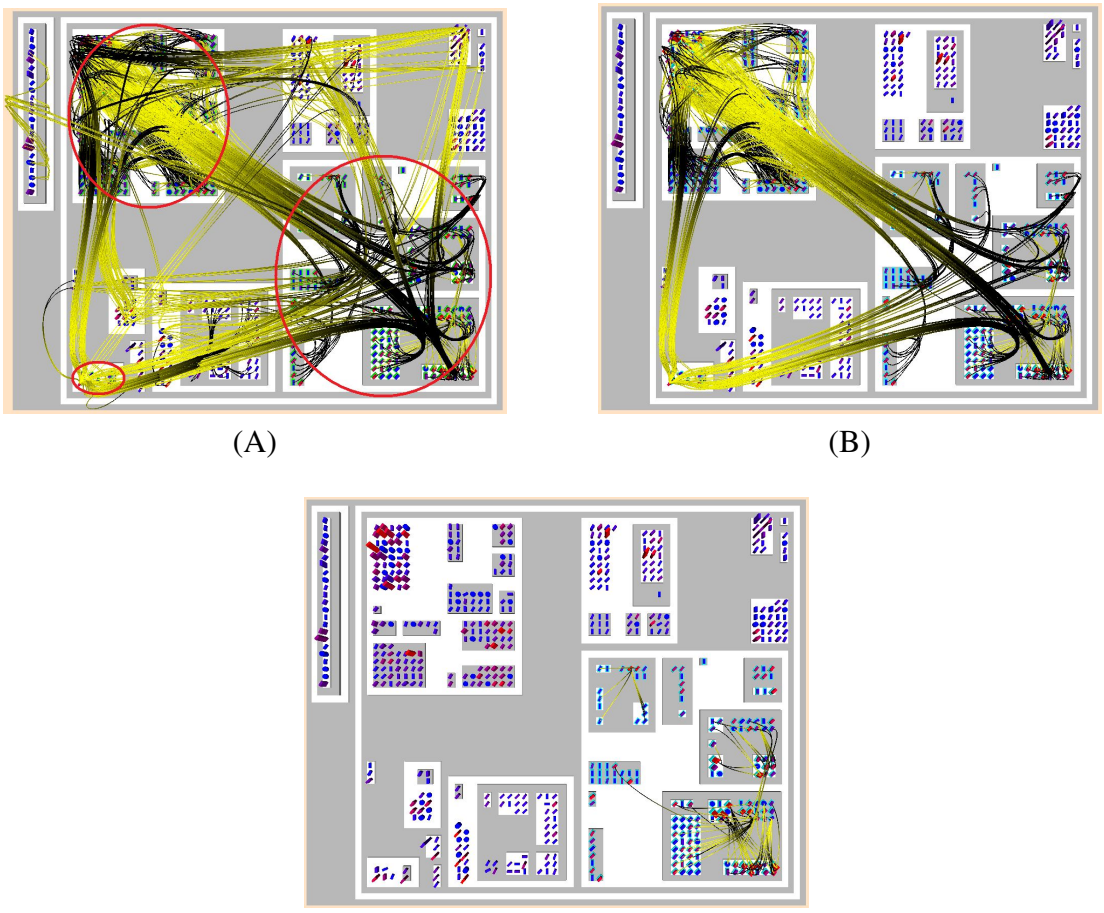
Figure 4.28 – Exemple de la sélection d'éléments. L'image A) montre la visualisation au niveau le plus détaillé, sans aucun filtre. L'image B) montre ce qui arrive lorsque l'utilisateur sélectionne le paquetage ouvert situé sous le curseur (la flèche rouge) en cliquant dessus. Tous les descendants de ce paquetage qui sont présentement affichés, dans ce cas-ci l'ensemble de ses classes descendantes, se retrouvent sélectionnés, ce qui est indiqué par la bordure verte entourant chacun d'eux. En filtrant les liens d'adjacence en fonction des éléments sélectionnés, nous pouvons nous concentrer uniquement sur les liens d'adjacence se rapportant aux éléments qui nous intéressent. Par exemple, dans ce cas-ci nous pouvons voir facilement qu'il n'y a aucun lien entrant dans le paquetage sélectionné.

les liens d'adjacence qui unissent deux éléments qui ont été sélectionnés, et de cacher tous les autres. En réduisant l'encombrement visuel causé par les liens qui n'ont rien à voir avec la sélection de l'utilisateur, cela lui permet de mieux voir les communications existant entre les paquetages qui l'intéressent ou entre les éléments internes d'un paquetage en particulier. La figure 4.29 montre un exemple de l'application de ce filtre, que nous appelons *filtre intra-sélections*.

Il peut aussi arriver que l'utilisateur ne s'intéresse qu'aux communications qu'ont les éléments d'un paquetage donné avec le reste du système. Dans cette situation, l'affichage des liens internes du paquetage qu'a sélectionné l'utilisateur ne font que lui nuire. Nous avons donc ajouté un autre filtre, que nous appelons *filtre extra-sélections*, qui est en fait le dual du filtre intra-sélections, c'est-à-dire qu'il cache tous les liens existant entre deux éléments ayant été sélectionnés. La figure 4.30 montre un exemple de l'application de ce filtre.

4.5.3 Sélection de liens d'adjacence dans la visualisation

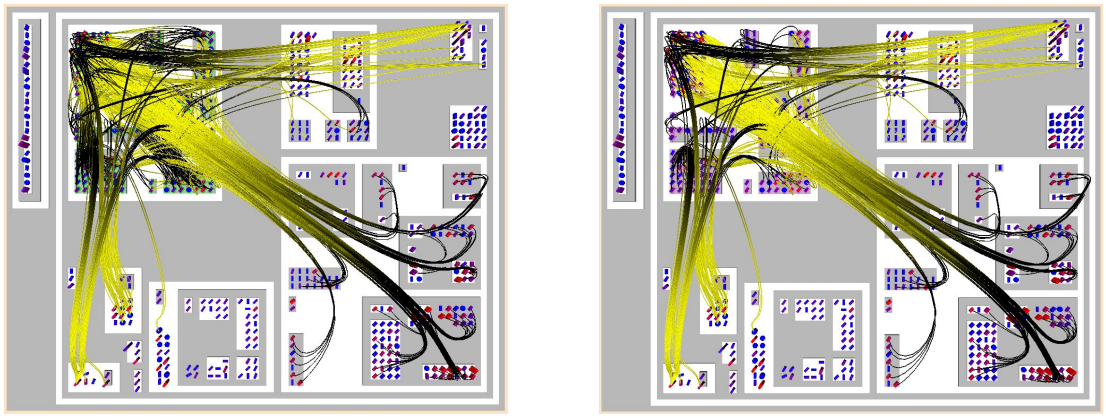
Il peut arriver souvent qu'un utilisateur ne s'intéresse pas à des éléments de la représentation hiérarchique en particulier, mais plutôt à certains paquets de liens d'adjacence. Il est donc nécessaire dans ce cas-là de lui donner la possibilité de n'afficher que les paquets de liens qui l'intéressent et de cacher tous les autres. Pour ce faire, l'utilisateur peut sélectionner les liens désirés en traçant avec la souris une zone de sélection rectangulaire, exactement comme il le ferait pour sélectionner un groupe d'éléments. Nous avons donc modifié l'interface afin que l'utilisateur puisse spécifier ce qu'il veut sélectionner avec la souris : des éléments de la représentation hiérarchique ou des liens d'adjacence. L'utilisateur peut aussi appliquer simultanément plusieurs rectangles de sélection, c'est-à-dire qu'il trace avec la souris plusieurs rectangles l'un après l'autre en maintenant une touche précise enfoncée puis, lorsqu'il la relâche, les liens d'adjacence sont filtrés en utilisant tous les rectangles qu'il a tracés comme zone de sélection. Cette fonctionnalité est aussi présente pour la sélection d'éléments de la représentation hiérarchique, mais celle-ci n'est vraiment utile que pour la sélection de liens d'adjacence. La raison à cela est que, lorsque l'utilisateur sélectionne des liens d'adjacence, tous ceux



(A) (B)

(C)

Figure 4.29 – Exemple de l'application du filtre intra-sélections. Les trois paquetages encerclés en rouge dans l'image A) sont ceux qui ont été sélectionnés par l'utilisateur. L'image B) montre les liens obtenus après avoir appliqué le filtre intra-sélections : celui-ci fait ressortir plus clairement les communications existant entre ces trois paquetages. Dans l'image B), la bordure autour des éléments qui ont été sélectionnés est de couleur cyan afin d'indiquer que le filtre intra-sélections est présentement appliqué. L'image C) montre le résultat obtenu en ne sélectionnant qu'un seul des trois paquetages montrés à l'image A). Le filtre intra-sélections permet de confirmer qu'il n'y a presqu'aucune communication entre ses sous-paquetages directs, et qu'il y a en général très peu de communications entre ses descendants.

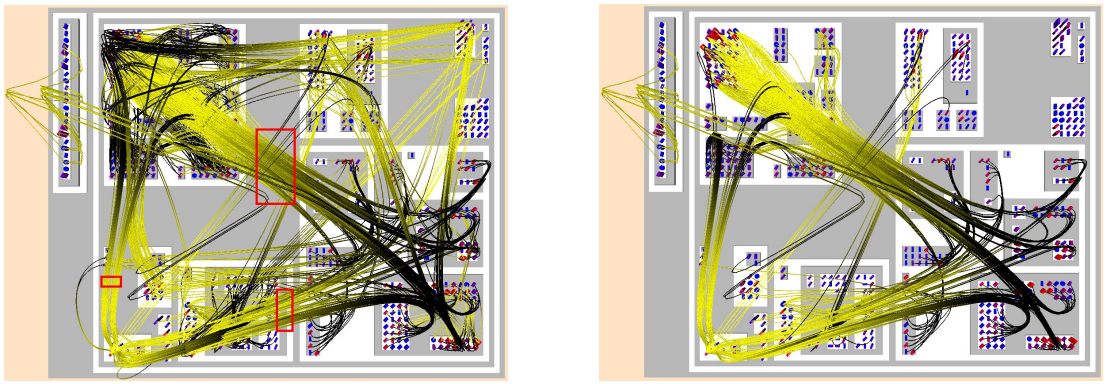


(A) (B)

Figure 4.30 – Exemple de l'application du filtre extra-sélections. Dans l'image A), l'utilisateur a sélectionné le paquetage situé en haut à gauche dans la visualisation. Les liens internes de ce paquetage créent un peu d'interférence avec ses liens externes : par exemple, la présence accrue de noir dans le paquetage, causée par l'affichage des liens internes, pourrait nous faire croire que celui-ci appelle davantage d'éléments dans le reste du système qu'il ne le fait en réalité. L'image B) montre les liens obtenus après avoir appliqué le filtre extra-sélections : il est plus facile dans cette image de voir exactement ce qui entre ou sort de ce paquetage. Dans l'image B), la bordure autour des éléments qui ont été sélectionnés est de couleur violet afin d'indiquer que le filtre extra-sélections est présentement appliqué.

qui ne font pas partie de la sélection sont enlevés de la visualisation, l'empêchant ainsi d'ajouter par la suite d'autres paquets de liens à sa sélection. Sans la sélection multiple, il serait donc impossible pour l'utilisateur de sélectionner deux paquets de liens qui l'intéressent lorsque ceux-ci sont trop éloignés dans la visualisation, car la plupart du temps dans cette situation ceux-ci ne peuvent pas être regroupés dans la même zone de sélection sans inclure du même coup plusieurs autres liens d'adjacence qui ne sont pas désirés. Ce problème ne se présente pas dans la sélection d'éléments de la représentation hiérarchique, car tous ceux qui ne sont pas sélectionnés restent tout de même affichés dans la visualisation, ce qui permet à l'utilisateur de compléter sa sélection après coup.

La sélection de liens d'adjacence est surtout utile lorsque l'utilisateur s'intéresse à un paquet de liens en particulier, et qu'il veut déterminer quels sont les groupes d'éléments appelants et d'éléments appelés dont les liens forment ce paquet, et ainsi de voir si celui-ci peut être divisé en plusieurs sous-paquets de liens. La figure 4.31 montre un exemple de sélection multiple de paquets de liens d'adjacence.



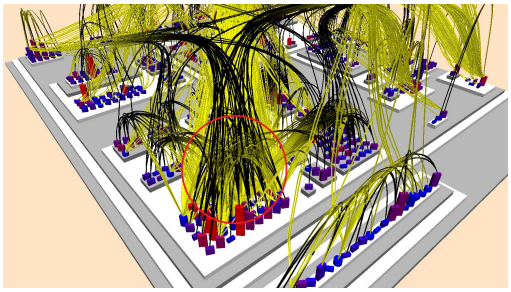
(A) (B)

Figure 4.31 – Exemple de sélection multiple des liens d'adjacence. La visualisation dans l'image A) présente tous les liens d'adjacence. Les rectangles rouges sont les zones de sélection qu'a tracées l'utilisateur, montrant quels sont les paquets de liens qui l'intéressent. L'image B) montre les liens qui restent affichés après avoir appliqué le filtre correspondant à la sélection. Même si certains liens non désirés ont été sélectionnés, dû à leur proximité avec les liens d'intérêts, l'utilisateur peut toujours raffiner sa sélection par la suite. Après avoir appliqué sa sélection, l'utilisateur peut voir plus facilement où commencent et où se terminent les paquets de liens d'adjacence sélectionnés, et entre quels groupes d'éléments ceux-ci existent.

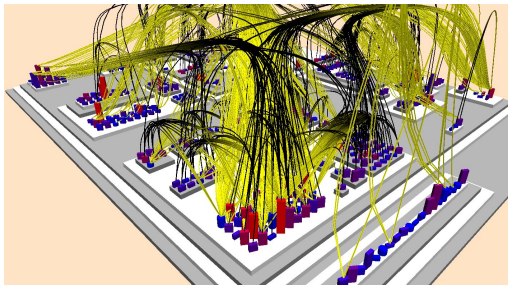
4.5.4 Filtrage des liens internes et externes

Nous définissons deux catégories de liens d'adjacence : les liens existant entre des classes sœurs, que nous appelons liens internes, et les autres, c'est-à-dire ceux existant entre deux classes qui ne sont pas des sœurs, que nous appelons liens externes. Par défaut, la visualisation que nous offrons à l'utilisateur n'affiche pas les liens internes, comme c'était le cas dans tous les exemples présentés jusqu'à maintenant. La raison à cela découle du fait que celles-ci sont toujours placées en groupes compacts, phénomène qui était aussi la source du problème d'enchevêtrement des liens entrants et sortants expliqué à la section 4.3.3.2, et qui nous a amenés à créer le concept des nœuds de sortie afin d'atténuer autant que possible celui-ci. Les liens internes, lorsqu'ils sont affichés, créent beaucoup d'interférence avec les liens venant de l'extérieur du groupe de classes sœurs, c'est-à-dire avec les liens externes. Ainsi, si nous affichions les liens internes en même temps que les liens externes, cela risquerait de générer beaucoup d'encombrement visuel dans la zone d'entrée des groupes de classes sœurs. Cet encombrement visuel rendrait autant les liens internes que les liens externes plus difficiles à distinguer, ce qui détruirait en partie le gain obtenu par l'ajout des nœuds de sortie. C'est pourquoi nous avons décidé de n'afficher qu'une seule de ces deux catégories de liens à la fois, et de permettre à l'utilisateur de basculer en temps réel entre chacune d'elle. De plus, la plupart du temps les liens internes et externes donnent des informations complémentaires à l'utilisateur, et dans de tels cas il n'est pas crucial que celui-ci puisse voir ces deux catégories de liens en même temps. Par exemple, avec des liens d'appels les liens externes montrent les dépendances qu'a un paquetage avec le reste du logiciel, tandis que les liens internes indiquent plutôt la cohésion de celui-ci. L'utilisateur pourrait donc, dans cette situation, commencer par utiliser les liens externes pour faire l'analyse des dépendances existant entre les paquetages du logiciel puis, en basculant dans la visualisation des liens internes, faire l'analyse de la cohésion de ceux-ci. La figure 4.32 montre l'encombrement visuel causé par l'affichage simultané des liens internes et externes. La figure 4.33 présente un exemple d'une vue d'ensemble des liens internes. Il est à noter que nous ne considérons pas les liens entre paquetages frères fermés comme étant

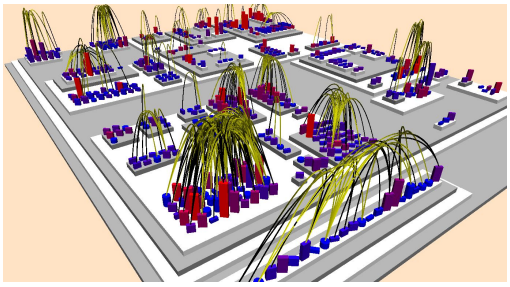
des liens internes, car les paquetages frères sont rarement placés en groupes compacts et, ainsi, les liens internes les unissant créent rarement de l'interférence avec les liens externes.



(A)



(B)



(C)

Figure 4.32 – Affichage simultané des liens internes et externes. L'image A) montre l'encombrement visuel présent à l'entrée des groupes de classes lorsque nous affichons simultanément les liens internes et externes. L'image B) montre la visualisation obtenue lorsque nous affichons seulement les liens externes, tandis que l'image C) montre celle obtenue lorsque nous affichons seulement les liens internes. Ces images montrent que chaque catégorie de liens est plus facile à visualiser lorsqu'elle affichée seule.

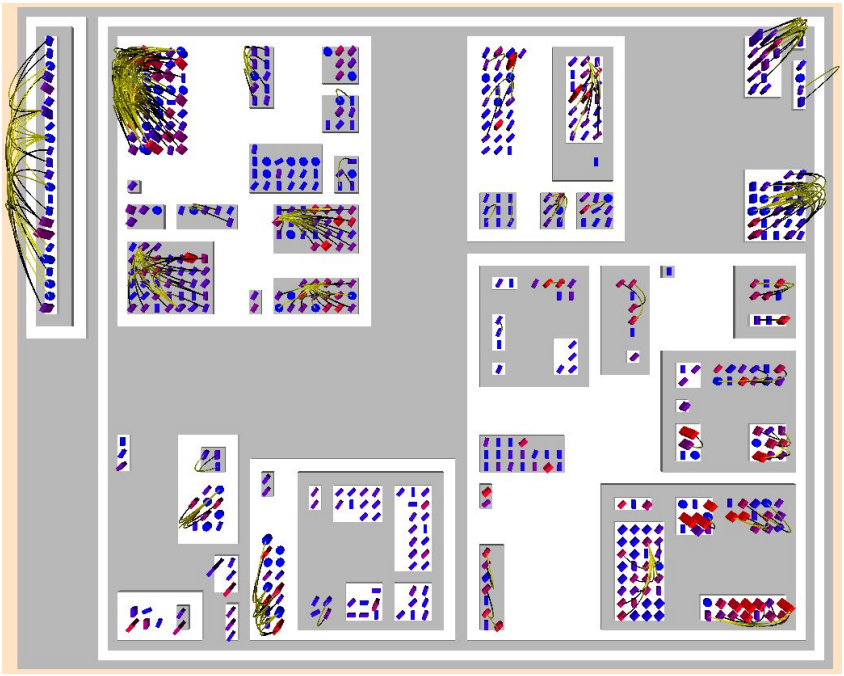


Figure 4.33 – Exemple d'une vue d'ensemble des liens internes. Nous pouvons remarquer qu'il y a seulement quelques paquetages, dans ce logiciel, dont les sous-classes directes communiquent beaucoup entre elles.

CHAPITRE 5

ÉVALUATION DES RÉSULTATS

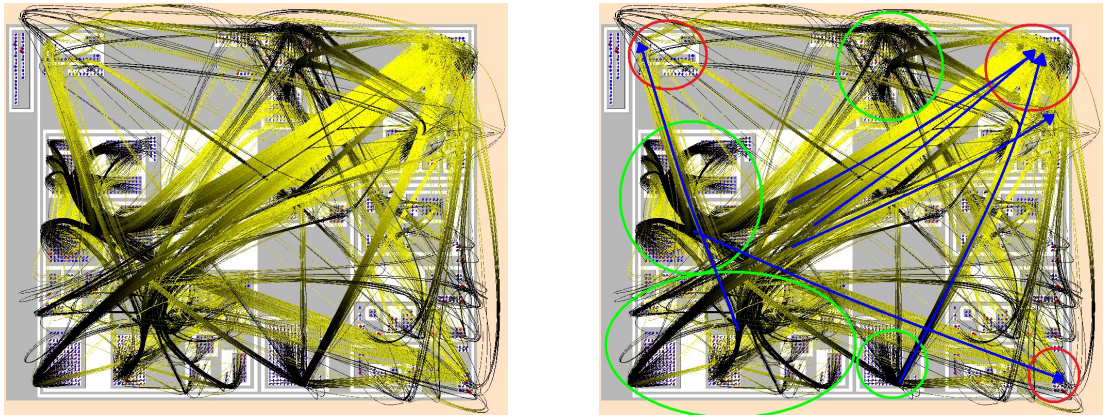
Dans ce chapitre, nous allons montrer certaines découvertes que nous avons faites en analysant de façon exploratoire quelques logiciels à l'aide de notre système de visualisation. Cette analyse sert à illustrer la façon dont nous pouvons utiliser notre système de visualisation, ainsi que les méthodes interactives qu'il offre, sur de vrais logiciels afin de faire ressortir de l'information potentiellement intéressante pour un développeur. Nous avons analysé deux sortes de liens avec chacun des logiciels : les liens d'invocation et les liens d'héritage. Les liens d'invocation représentent l'utilisation des classes du logiciel entre elles : il existe un lien d'invocation entre deux classes sitôt que l'une des deux utilise l'autre, c'est-à-dire sitôt qu'elle utilise une méthode de l'autre classe ou déclare un objet du type de celle-ci. Il existe aussi un lien d'invocation entre une classe et les interfaces qu'elle utilise. Ces liens sont affichés avec noir comme couleur de départ (classe appelante) et jaune comme couleur d'arrivée (classe appelée). Les liens d'héritage représentent l'héritage qui existe entre les classes du logiciel : il existe un lien d'héritage entre deux classes lorsqu'une hérite directement de l'autre. Évidemment, nous n'affichons pas les liens d'héritage de celles qui héritent de classes provenant de l'extérieur du logiciel, comme par exemple du paquetage *JDK*. Ces liens sont affichés avec noir comme couleur de départ (classe fille) et vert comme couleur d'arrivée (classe mère). Deux logiciels sont présentés dans ce chapitre : *ArgoUML* à la section 5.1, et *Azureus* à la section 5.2. Il est à noter que, pour raison d'espace, les exemples présentés dans chaque section utilisent tous le placement *Treemap*. Afin de pouvoir comparer rapidement la visualisation qu'offrent les autres types de placements présents dans notre système, une vue d'ensemble de chaque type de liens sera montrée avec le placement radial et le placement Colisée à la fin de chaque section.

5.1 Analyse des liens dans *ArgoUML*

Le logiciel visualisé dans cette section est *ArgoUML*, un logiciel utilisé pour créer des diagrammes *UML*. La version de *ArgoUML* utilisée est composée de 2291 classes réparties dans 142 paquetages, qui eux sont étalés sur un maximum de 10 niveaux de profondeur (incluant la racine). Au niveau le plus détaillé, la visualisation contient 4627 liens d'invocation et 665 liens d'héritage, si nous n'incluons pas les liens internes.

5.1.1 Liens d'invocation

Pour commencer, la figure 5.1 montre une vue d'ensemble des liens d'invocation. Les principales zones de classes appelantes et appelées sont encerclées, et la direction des principaux paquets de liens est montrée à l'aide d'une flèche bleue.

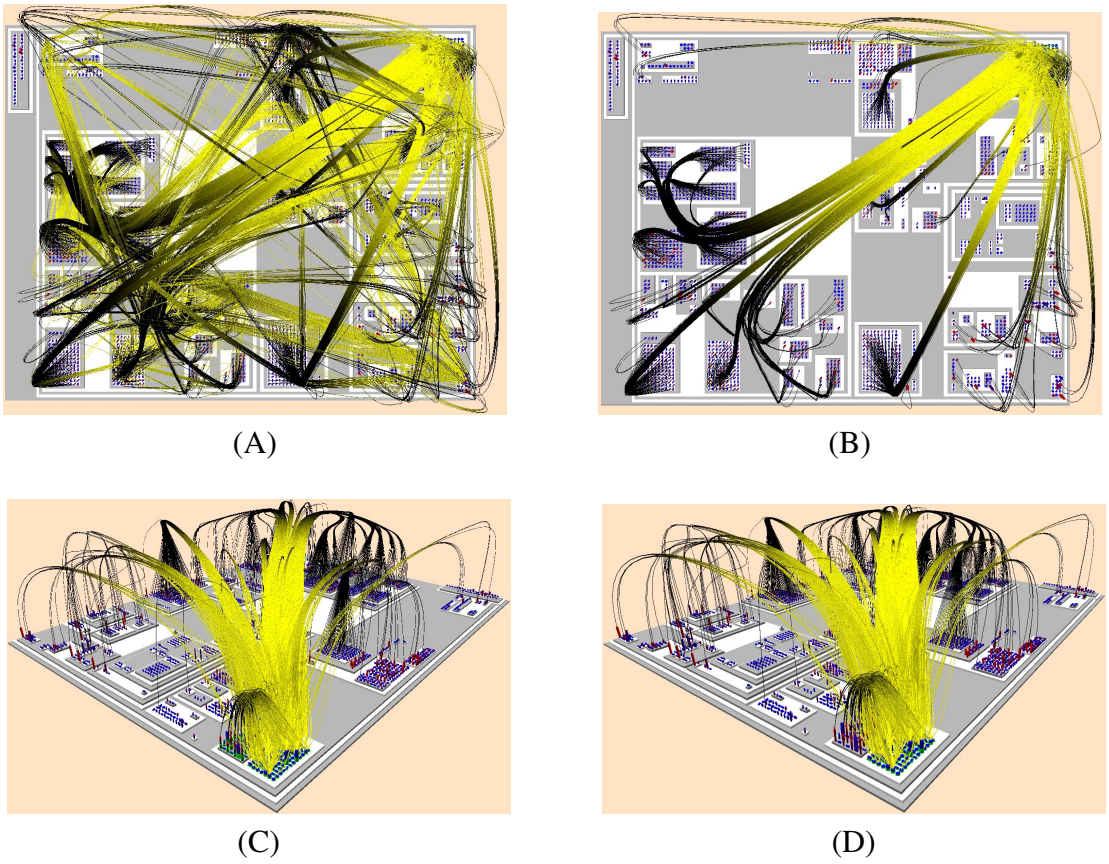


(A) (B)

Figure 5.1 – Vue d'ensemble des liens d'invocation dans *ArgoUML*. L'image A) montre la vue d'ensemble originale. L'image B) explicite les informations qui sont apparentes au premier coup d'œil : les principales zones de classes appelantes sont encerclées en vert, celles de classes appelées en rouge et des flèches bleues sont tracées pour indiquer la direction des principaux paquets de liens.

Dans cette figure, nous remarquons tout de suite que la majeure partie des liens se dirigent vers le paquetage situé en haut à droite du placement, ce qui nous porte à croire que celui-ci constitue le cœur du système. Le nom de ce paquetage étant *model*, cela nous laisse penser qu'il contient surtout des interfaces décrivant les modèles d'éléments

qu'on peut dessiner dans un graphe *UML*. Ceci expliquerait pourquoi autant de classes utilisent au moins un des éléments de ce paquetage, la plupart du temps une interface. Les figures 5.2 et 5.3 montrent plus en détail ce paquetage.



(A) (B)

(C) (D)

Figure 5.2 – Paquetage cœur d'*ArgoUML*. L'image A) montre la vue d'ensemble du logiciel. L'image B) montre les liens qu'on obtient en sélectionnant le paquetage en haut à droite : cela nous permet de mieux voir d'où proviennent les liens qui se dirigent vers celui-ci. L'image C) montre ce paquetage sous un autre angle de vue. L'image D) montre que l'ensemble de liens affichés n'a pas changé après avoir sélectionné les classes et interfaces situées dans le paquetage principal : cela signifie que tous les liens se dirigent vers elles seulement.

Nous pouvons voir assez facilement que la majorité des appels se font entre les paquetages principaux du logiciel, ce qui implique qu'il y a peu de communication à l'intérieur de ceux-ci. Les figures 5.4 à 5.5 montrent quelques exemples de cette observation. Ces figures montrent aussi que les paquetages clients sont presqu'uniquement des

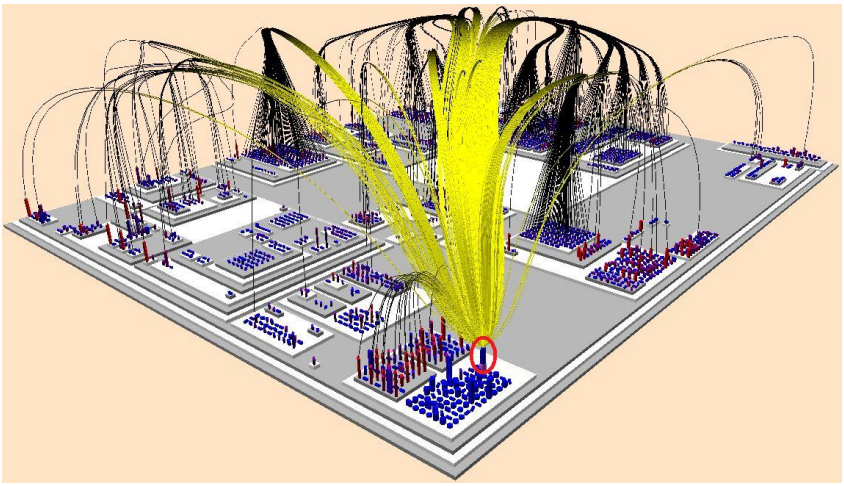
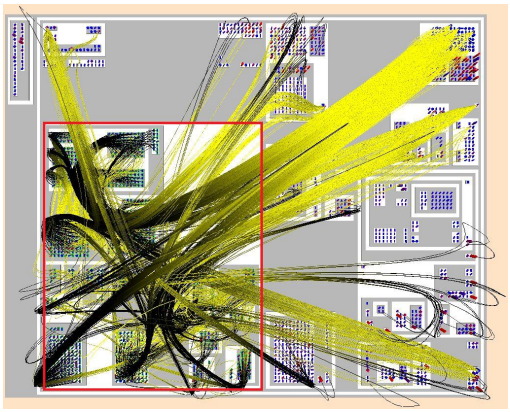


Figure 5.3 – Interface cœur du paquetage. Après avoir inspecté les éléments du paquetage cœur, nous avons remarqué qu'une très grande partie des liens se dirigent vers la même interface. Cette interface s'appelant *Facade*, cela nous laisse penser que le patron de conception façade est utilisé, ce qui serait cohérent avec l'immense quantité d'éléments qui utilisent cette interface.

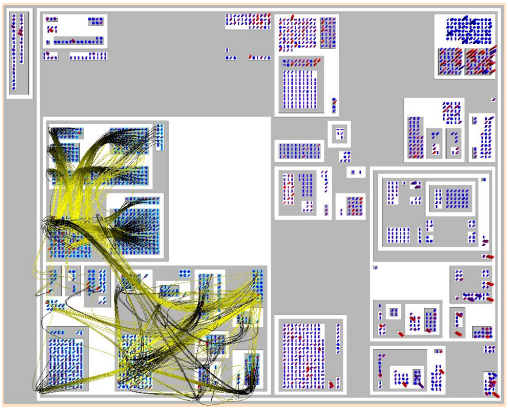
clients : pratiquement aucun lien venant de l'extérieur de ces paquetages rejoint un de leurs éléments. Les paquetages de ce logiciel sont donc pour la plupart soit un client, soit un fournisseur.

Une autre façon de confirmer que la majorité des liens se font entre les paquetages principaux du logiciel est de regarder la visualisation en biais (caméra à 45^o d'élévation). Faisant cela, nous voyons que les liens forment une couche assez élevée dans la visualisation. La formation d'une telle couche signifie que la plupart des liens passe par le même paquetage avant d'aller rejoindre leur élément cible. Dans ce cas-ci, la hauteur de la couche indique que c'est un paquetage élevé dans la hiérarchie. Après vérification, nous avons trouvé qu'il s'agit du paquetage *argouml*. Ce paquetage englobant pratiquement tous les éléments du logiciel, cela confirme que les liens se font majoritairement entre les principaux paquetages frères du logiciel, c'est-à-dire les sous-paquetages d'*argouml*. La figure 5.6 montre un exemple de cette couche.

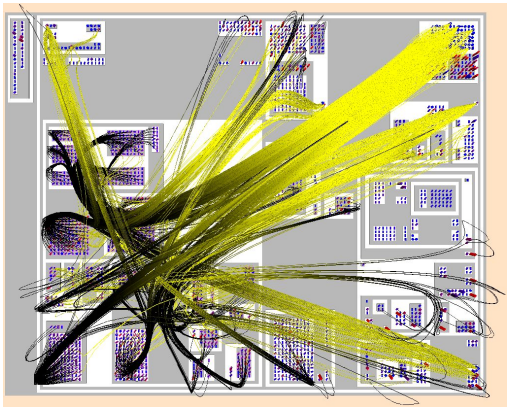
Une autre observation que nous avons faite est la quantité incroyable de liens se dirigeant vers une petite classe isolée dans le logiciel. La figure 5.7 montre cette situation. Après vérification, nous avons découvert que cette classe, qui se nomme *Translator*,



(A)

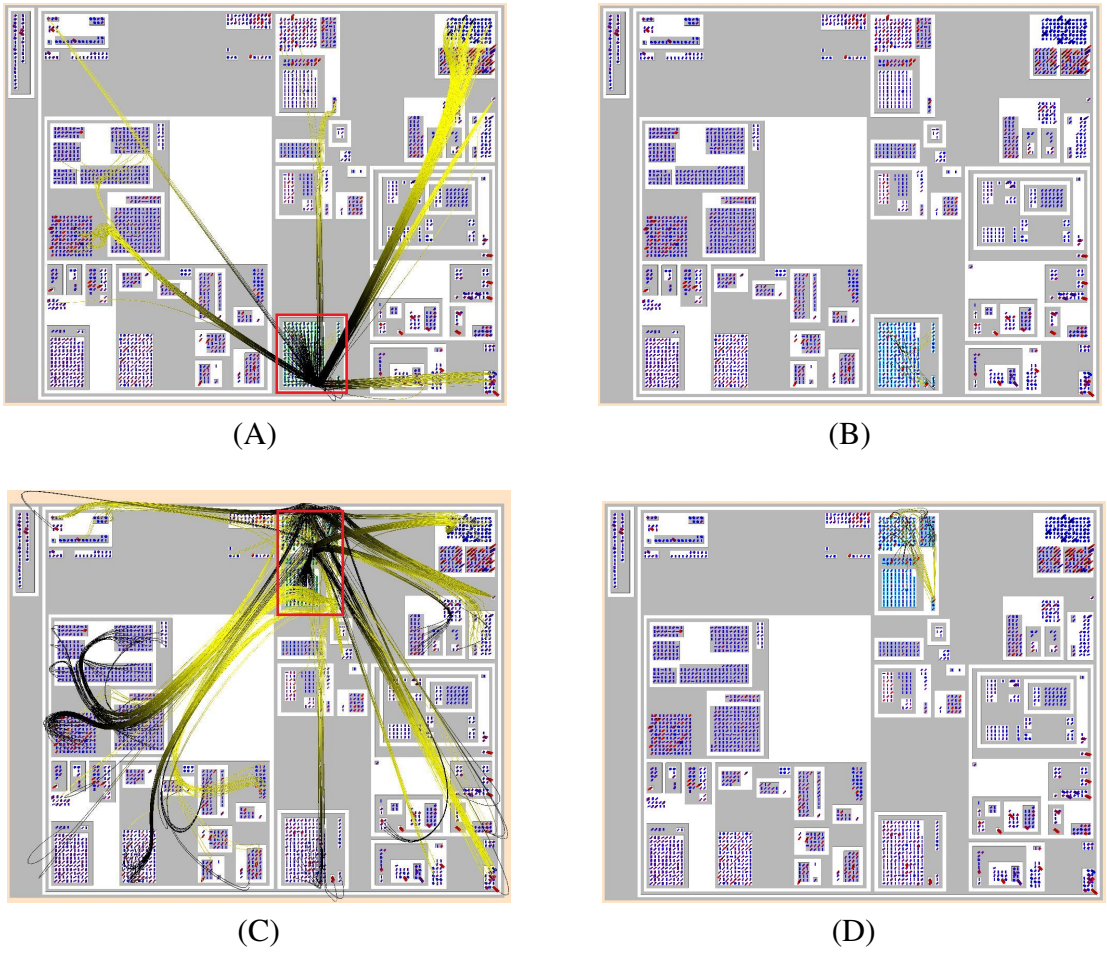


(B)



(C)

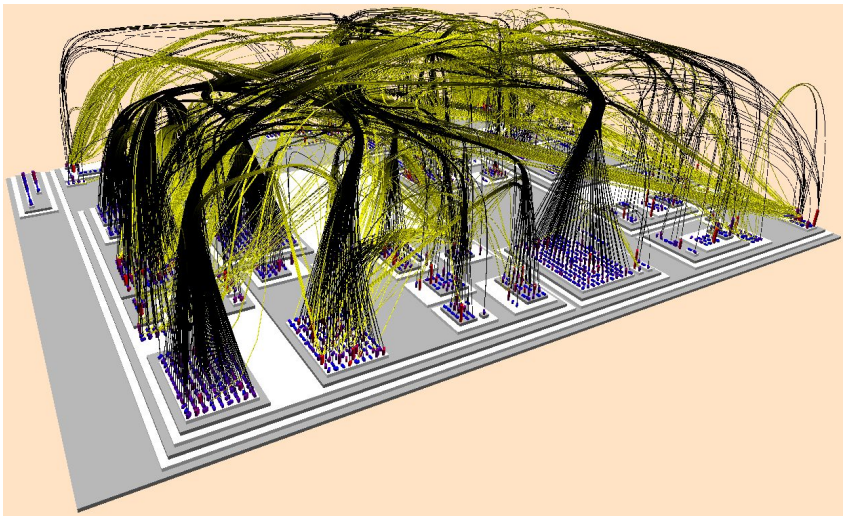
Figure 5.4 – Exemple de liens entre les paquetages principaux. L'image A) montre les liens qu'on obtient après avoir sélectionné le gros paquetage dans le coin inférieur gauche. L'image B) montre les liens entre les éléments internes de celui-ci, liens qui sont difficiles à voir lorsque tous les liens d'adjacence sont affichés. L'image C) montre les liens allant rejoindre des éléments situés à l'extérieur du paquetage. L'image C) nous confirme aussi que ce paquetage est majoritairement le client d'autres paquetages, par la direction des appels.



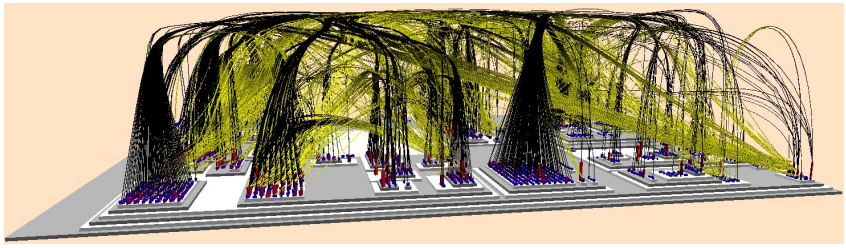
(A) (B)

(C) (D)

Figure 5.5 – Autres exemples de liens entre les paquetages principaux. Ces images nous confirment que les paquetages montrés en exemple présentent les mêmes caractéristiques que celui montré à la figure 5.4. Par contre, le paquetage montré aux images C) et D) contient une classe qui est utilisée par plusieurs éléments venant de l'extérieur, et donc n'est pas seulement un paquetage client.



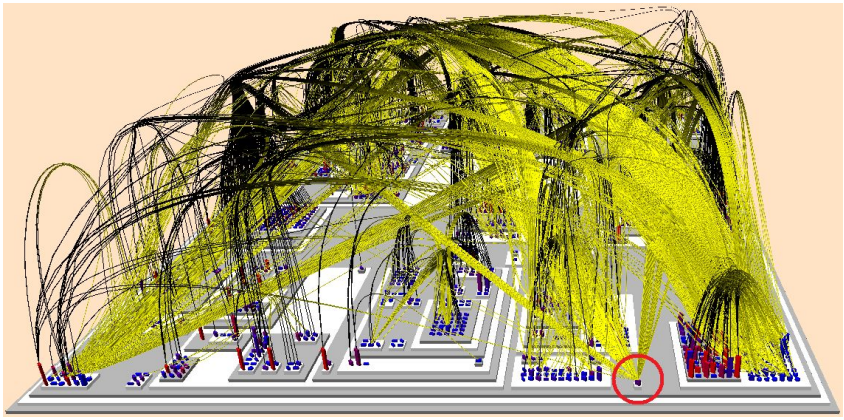
(A)



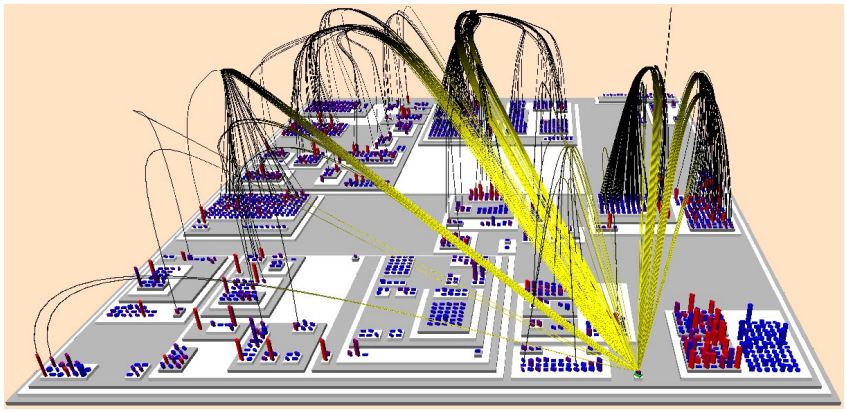
(B)

Figure 5.6 – Exemple d'une couche de liens. Les images A) et B) montrent, sous différents angles, la couche de liens située assez haut dans la visualisation. La hauteur de cette couche de liens, ainsi que le fait qu'elle couvre le paquetage *argouml*, nous permettent de déduire que la plupart des liens du logiciel se font entre les sous-paquetages de celui-ci.

est une classe utilitaire par laquelle doivent passer tous les appels provenant de classes voulant faire de la localisation, ce qui explique pourquoi autant de classes l'utilisent.



(A)

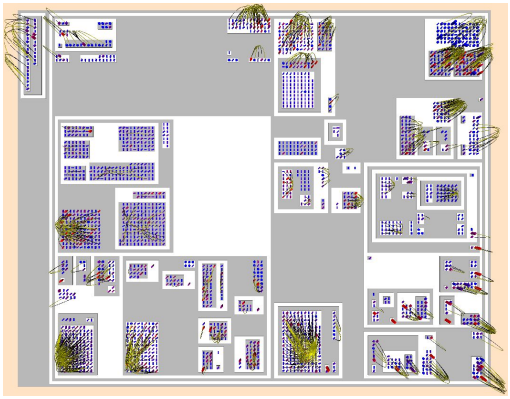


(B)

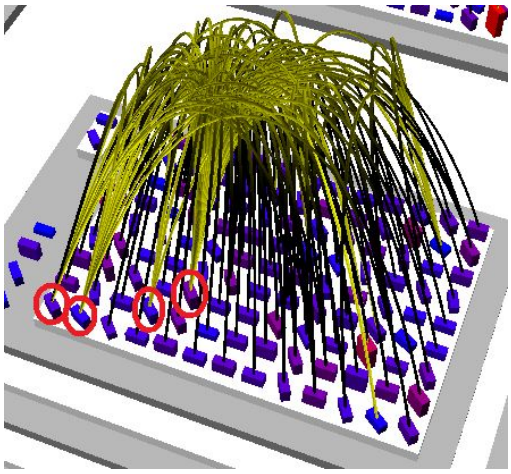
Figure 5.7 – Liens allant vers une classe isolée dans *ArgoUML*. L'image A) donne une vue d'ensemble alors que l'image B) affiche seulement les liens rejoignant cette classe.

En affichant les liens internes aux groupes de classes, cela nous permet de constater que plusieurs de ces groupes n'ont aucune, ou du moins presque aucune communication entre leurs éléments. Cela pourrait indiquer que ces paquetages ne sont pas très cohésifs. Pour les groupes présentant une forte communication entre leurs classes, nous pouvons facilement repérer, lorsque c'est le cas, les classes qui sont utilisées par une grande partie du reste du groupe. La figure 5.8 en montre un exemple.

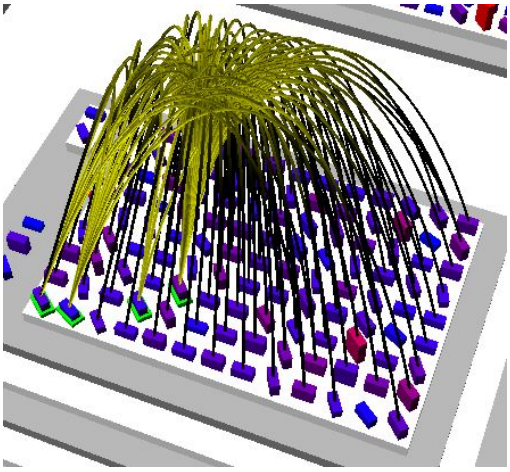
Une dernière observation intéressante que nous avons faite sur les liens d'invocation



(A)



(B)



(C)

Figure 5.8 – Liens d'invocation internes d'*ArgoUML*. L'image A) montre une vue d'ensemble des liens internes, nous permettant de constater que plusieurs paquetages en ont peu, voir aucun. L'image B) montre un paquetage où existe une forte communication entre ses classes. Les classes encerclées en rouge sont celles vers lesquelles semblent se diriger beaucoup de liens. L'image C) confirme cela une fois qu'on les sélectionne : les classes fortement liées à l'interne sont donc faciles à repérer.

est qu'il y a un paquetage complètement isolé dans le placement, celui-ci ne communiquant nullement avec le reste du logiciel. La figure 5.9 montre ce paquetage. Nous n'avons pas trouvé ce paquetage dans le code source d'*ArgoUML* : il se pourrait donc que nous l'ayons inclus par erreur dans le répertoire d'*ArgoUML* lorsque nous l'avons analysé pour créer le fichier utilisé par la visualisation. Cela nous a tout de même permis de constater qu'il est facile de détecter ce genre de paquetage dans notre visualisation.

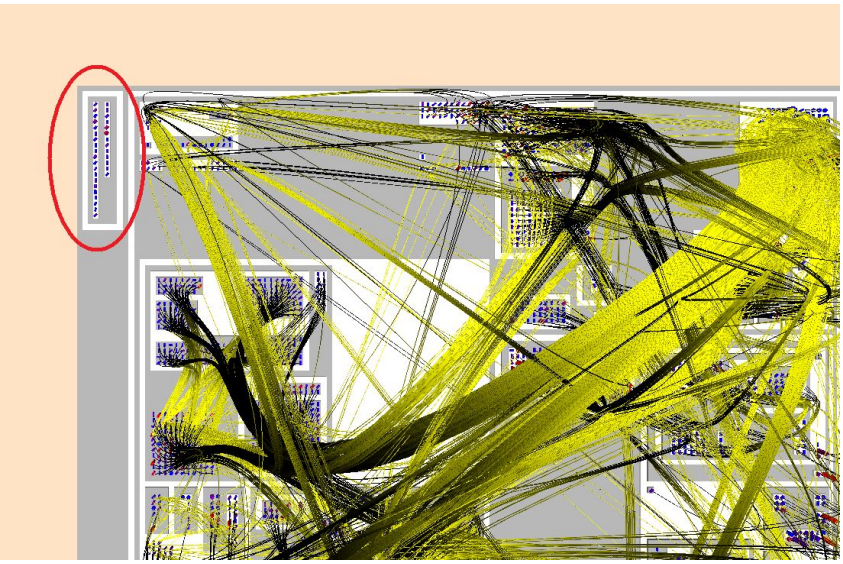
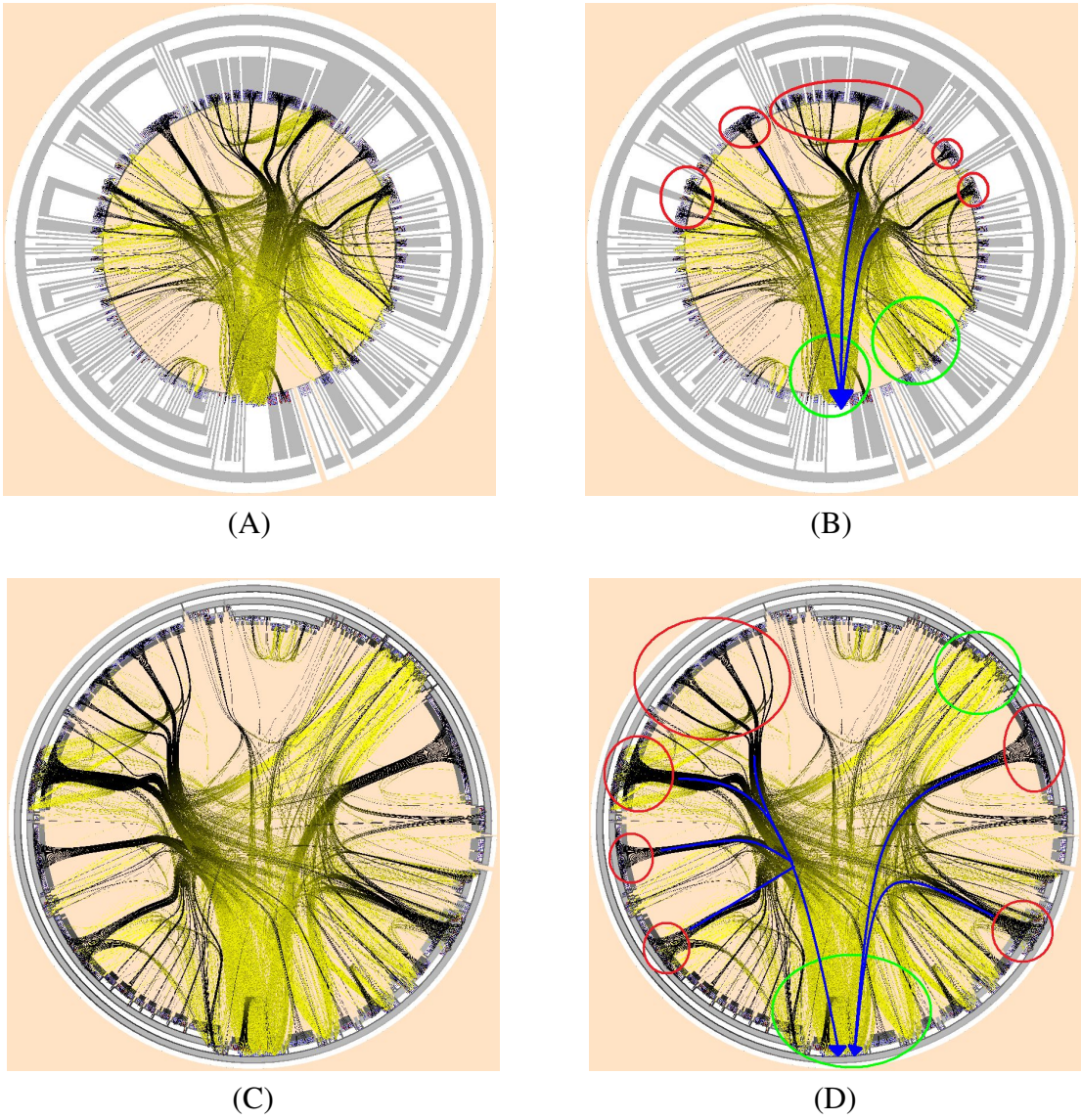


Figure 5.9 – Exemple de paquetage isolé dans *ArgoUML*.

Finalement, à des fins de comparaison, la figure 5.10 montre une vue d'ensemble des liens d'invocation en utilisant le placement radial et le placement Colisée. Chacune des observations faites utilisant le placement *Treemap* auraient pu être réalisées similairement avec ces deux placements.

5.1.2 Liens d'héritage

Pour commencer, la figure 5.11 présente une vue d'ensemble des liens d'héritage. Nous pouvons observer qu'il y a beaucoup de classes, situées dans le même paquetage principal, qui héritent d'une seule et même classe qui est assez éloignée dans la hiérarchie. Cette classe se nomme *UndoableAction* et, selon les commentaires du développeur dans le code source de cette classe, celle-ci est une classe de base pour toutes



(A) (B)

(C) (D)

Figure 5.10 – Vue d'ensemble des liens d'invocation d'*ArgoUML* avec les placements radial et Colisée. Les images A) et B) utilisent le placement radial, et les images C) et D) le placement Colisée.

les actions infaisables. Le commentaire expliquait aussi que toutes les classes représentant une action correspondant au début d'une interaction de l'usager devraient hériter de cette classe, ce qui explique pourquoi autant de classes en héritent. Les liens d'héritage avec cette classe représentent la majorité des liens d'héritage qui unissent deux classes éloignées dans la hiérarchie. La figure 5.12 offre une meilleure vue de cette classe.

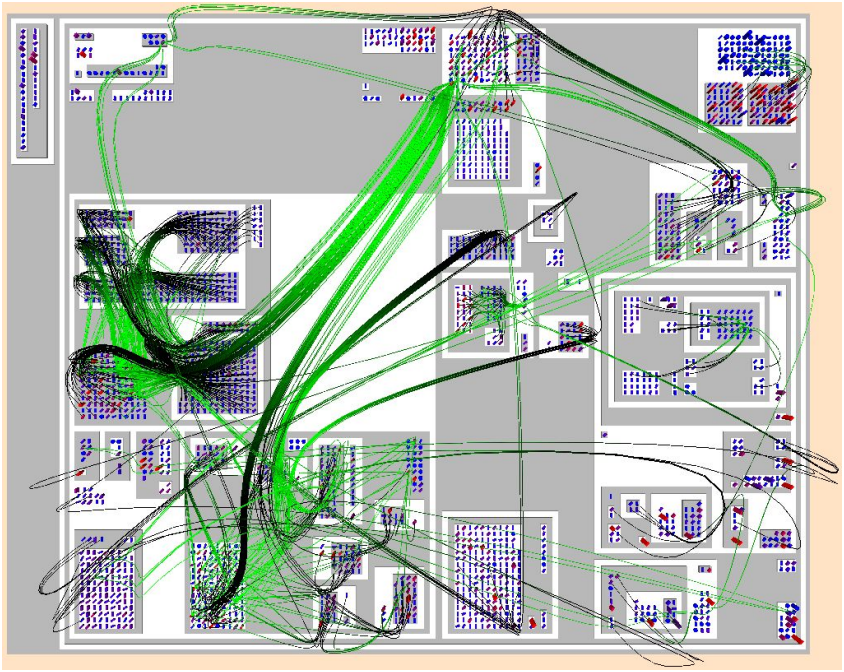


Figure 5.11 – Vue d'ensemble des liens d'héritage dans *ArgoUML*.

La figure 5.13 montre une vue d'ensemble des liens internes aux groupes de classes. Si nous y regardons de plus près, nous pouvons observer que les liens internes de certains groupes forment une « fontaine » de liens. Ces « fontaines » se produisent lorsqu'une grande partie des classes d'un paquetage héritent toutes de la même classe sœur. Il faudrait vérifier s'il ne serait pas préférable dans ces cas-là de créer des sous-classes à la classe mère principale qui pourraient servir d'intermédiaires au reste du groupe, de façon à diviser l'ensemble des classes du paquetage en différents sous-groupes plus restreints. La figure 5.14 montre quelques exemples de « fontaines » de liens. Cette image assez frappante permet de repérer facilement, dans notre visualisation, cette situation particulière dans l'héritage entre classes sœurs.

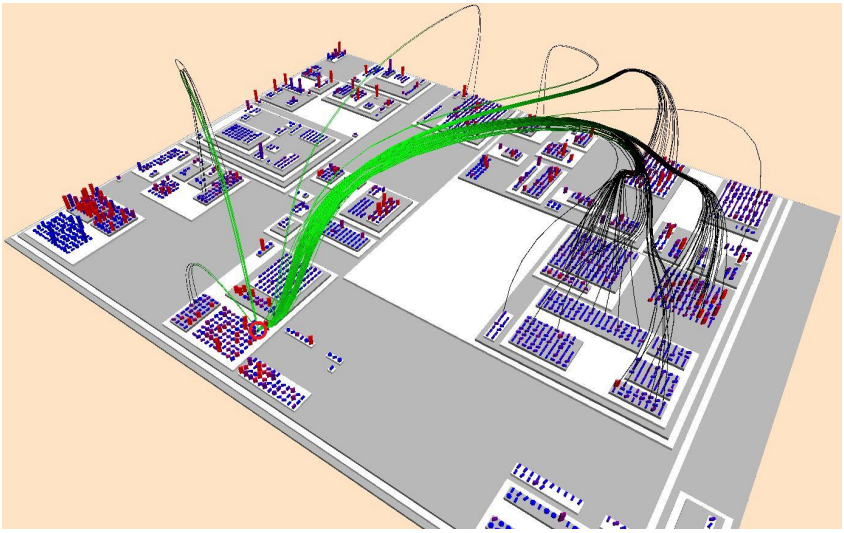


Figure 5.12 – Classe *UndoableAction* dans *ArgoUML*.

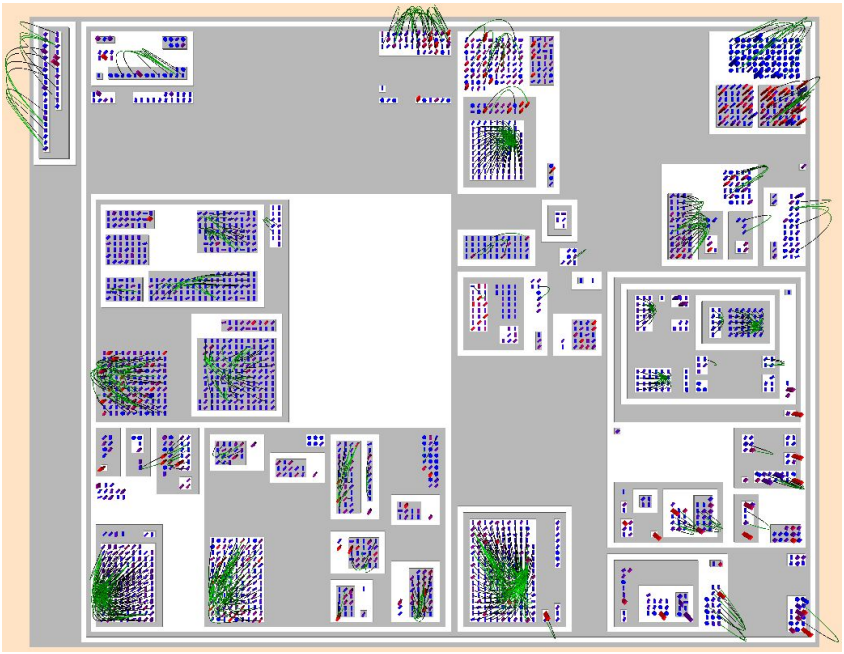
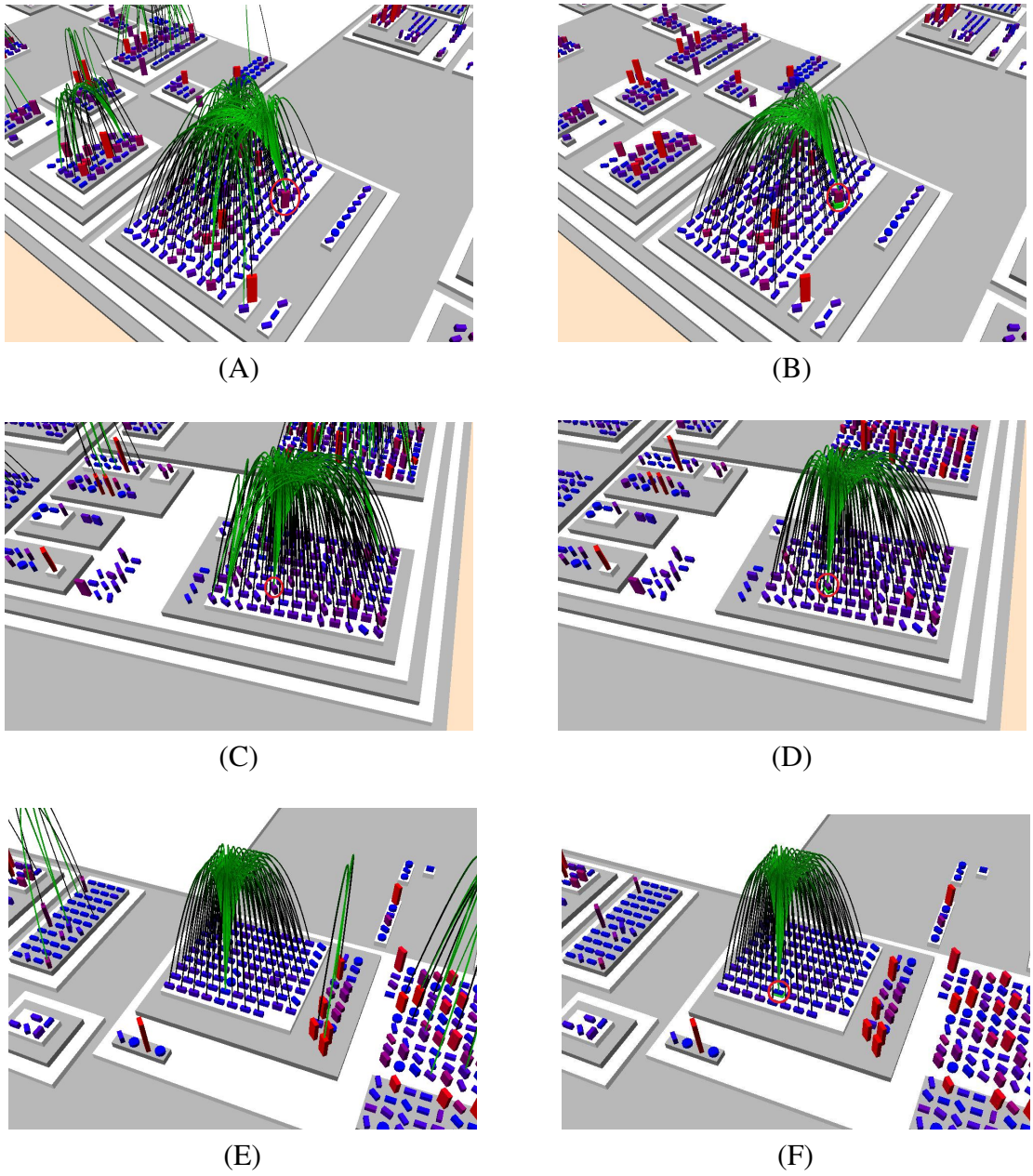


Figure 5.13 – Liens d'héritage internes d'*ArgoUML*.



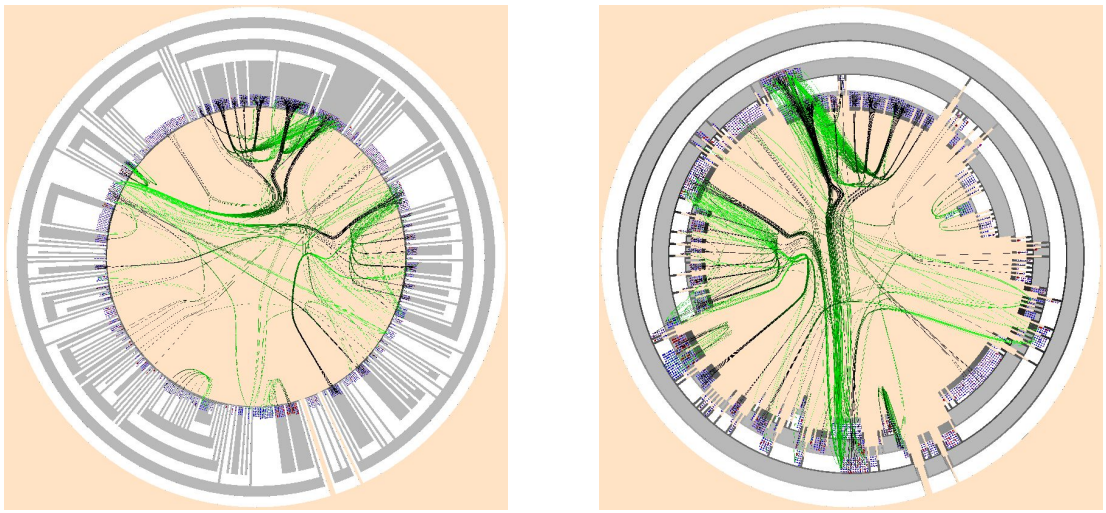
(A) (B)

(C) (D)

(E) (F)

Figure 5.14 – Exemple de « fontaines » de liens dans *ArgoUML*. Les images de la colonne de droite montre les liens obtenus après avoir sélectionné la classe suspectée d'être la source de la « fontaine ». Dans chaque cas, cela permet de confirmer nos soupçons. Il est à noter que la classe présentée dans les images E) et F) est la classe mère de presque toutes les classes de son paquetage.

Finalement, la figure 5.15 offre, à titre de comparaison, une vue d'ensemble des liens d'héritage qui utilise le placement radial et le placement Colisée.



(A) (B)

Figure 5.15 – Vue d'ensemble des liens d'héritage d'*ArgoUML* avec les placements radial et Colisée. L'image A) utilise le placement radial et l'image B) le placement Colisée.

5.2 Analyse des liens dans *Azureus*

Le logiciel visualisé dans cette section est *Azureus*, célèbre logiciel de partage de fichiers poste-à-poste. La version de *Azureus* utilisée est composée de 3286 classes réparties dans 520 paquetages, qui eux sont étalés sur un maximum de 11 niveaux de profondeur (incluant la racine). Au niveau le plus détaillé, la visualisation contient 10354 liens d'invocation et 439 liens d'héritage, si nous n'incluons pas les liens internes.

5.2.1 Liens d'invocation

Pour commencer, la figure 5.16 montre une vue d'ensemble des liens d'invocation. Les principales zones de classes appelantes et appelées sont encerclées, et la direction des principaux paquets de liens est montrée à l'aide d'une flèche bleue.

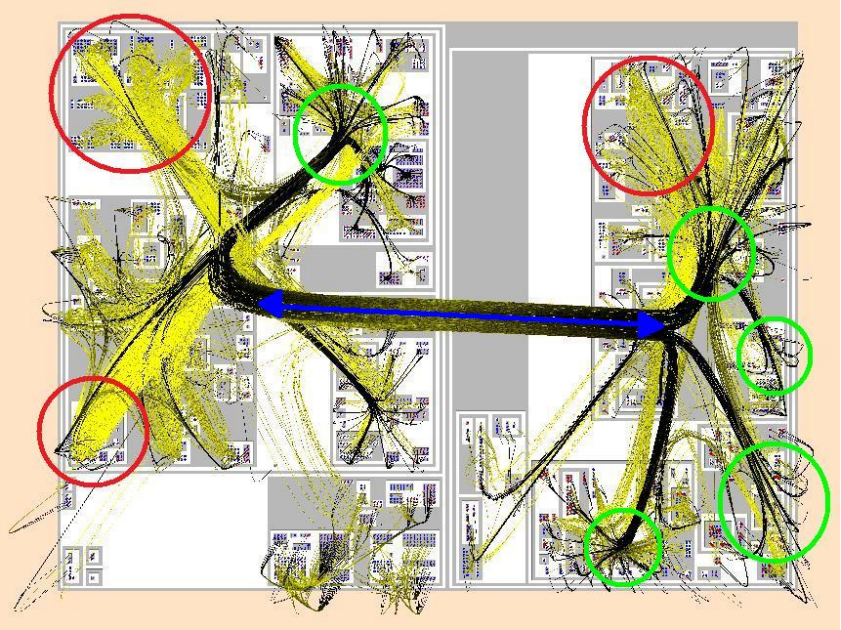


Figure 5.16 – Vue d'ensemble des liens d'invocation dans *Azureus*. Les principales zones de classes appelantes sont encerclées en vert, celles de classes appelées en rouge et une flèche bleue est tracée pour indiquer la direction des principaux paquets de liens.

Nous remarquons tout de suite un phénomène intéressant dans ce logiciel : il y a énormément de communication entre les deux paquetages principaux de celui-ci. Cela est clairement indiqué par l'énorme lien qui passe par le centre du placement. La figure 5.17 montre la visualisation obtenue après avoir sélectionné ce lien.

La vue de dessus nous permet de remarquer que ce gros lien est noir aux deux extrémités, ce qui indique qu'il est composé d'environ autant de liens allant dans un sens que dans l'autre. La figure 5.18 présente ce lien sous un autre angle afin de confirmer cette supposition.

La figure 5.18 nous confirme aussi que le gros lien est assez haut dans la hiérarchie, signifiant que les liens qui le composent montent beaucoup dans celle-ci avant d'atteindre leur cible.

La figure 5.6 à la section précédente montrait une couche de liens très haute dans la visualisation qui couvrait presque entièrement le placement. Cela signifiait que la plupart des liens se faisaient entre des éléments distants dans la hiérarchie du logiciel.

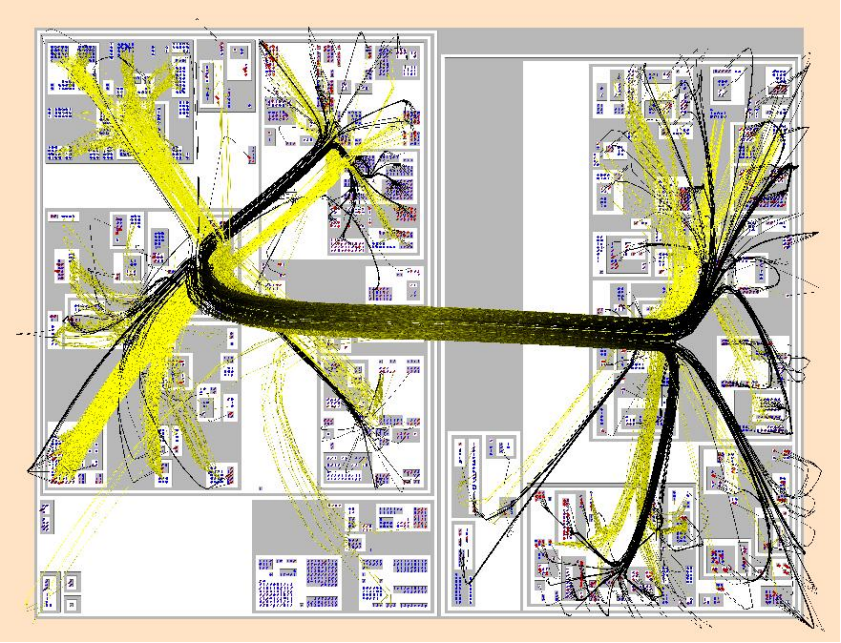


Figure 5.17 – Sélection du gros lien dans *Azureus*. Après avoir sélectionné ce lien, nous pouvons voir facilement les éléments appelants et appelés qui y participent.

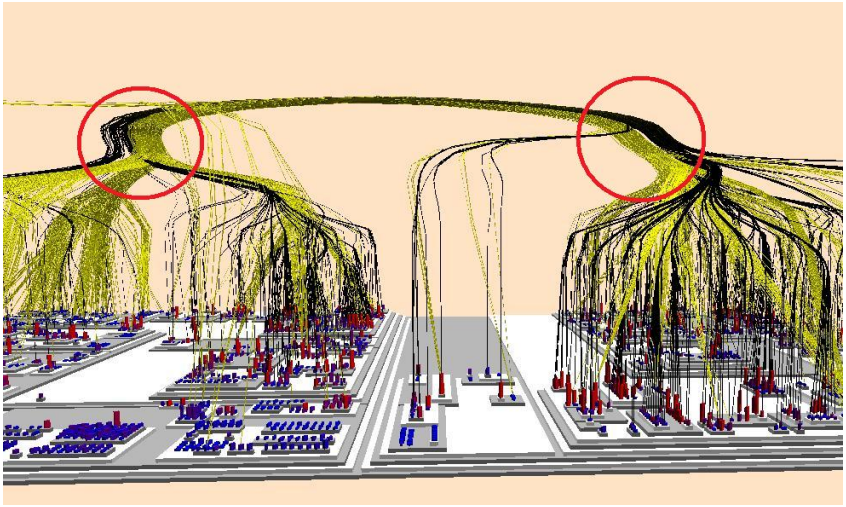


Figure 5.18 – Gros lien dans *Azureus* vu de côté. La vue de côté confirme que le gros lien est composé d'environ autant de liens allant dans un sens que l'autre, car on voit clairement qu'il y a un gros paquet de liens sortants (noirs) et de liens entrants (jaunes) à chaque bout.

Idéalement, nous devrions retrouver des couches de liens locales à des paquetages plus profonds dans la hiérarchie, ce qui serait un signe d'une meilleure cohésion des paquetages. La figure 5.19 montre une telle couche de liens, dont l'étendue est limitée à un groupe d'éléments relativement restreint.

Nous repérons assez facilement dans le placement un paquetage isolé qui ne communique pratiquement pas avec le reste du logiciel, comme le montre la figure 5.20. Avec un exemple similaire présenté à la section précédente, cela montre qu'il est assez facile avec notre visualisation de repérer de tels paquetages isolés dans le placement.

La figure 5.21 montre les liens internes aux groupes de classes sœurs. Nous pouvons remarquer qu'à part quelques exceptions, il y a peu de paquetages dont les classes communiquent beaucoup entre elles.

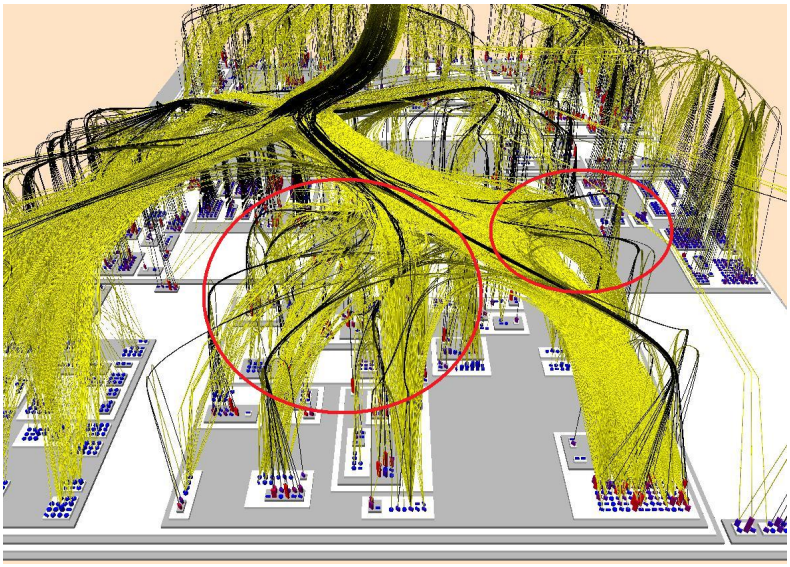
Finalement, à des fins de comparaison, la figure 5.22 montre une vue d'ensemble des liens d'invocation en utilisant le placement radial et le placement Colisée.

5.2.2 Liens d'héritage

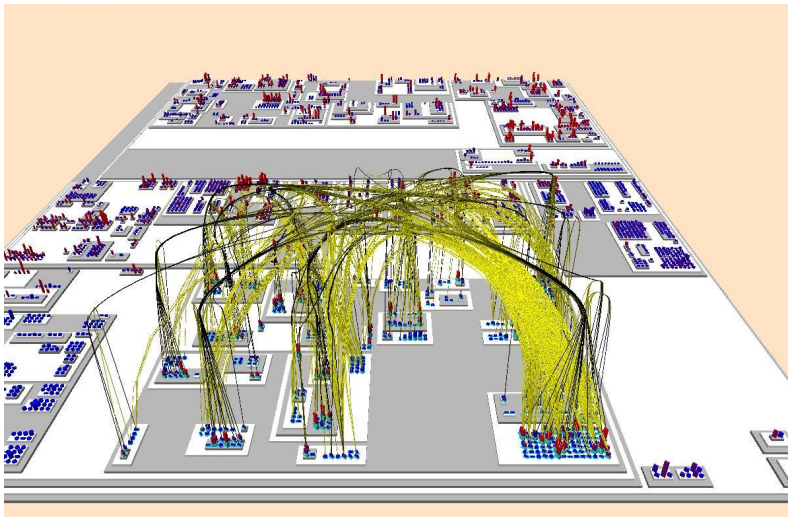
La figure 5.23 montre une vue d'ensemble des liens d'héritage. Nous pouvons remarquer qu'il y a très peu d'héritage entre des éléments éloignés de la hiérarchie.

Malgré qu'il y ait aussi très peu d'héritage entre des classes sœurs, nous avons quand même pu détecter quelques « fontaines » de liens, comme celles montrées à la section précédente dans *ArgoUML*. La « fontaine » de liens pourrait donc être considérée comme un *pattern* visuel permettant de repérer rapidement des groupes de classes héritant de façon massive de la même classe mère. La figure 5.24 montre quelques exemples de ce *pattern* dans *Azureus*.

Finalement, la figure 5.25 offre, à titre de comparaison, une vue d'ensemble des liens d'héritage qui utilise le placement radial et le placement Colisée.

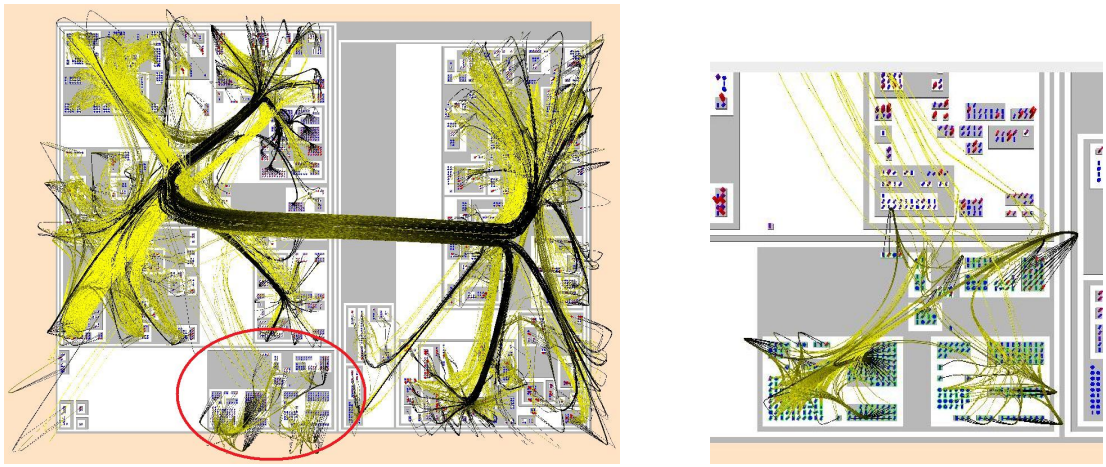


(A)



(B)

Figure 5.19 – Exemple de couche de liens dans *Azureus*. L'image A) montre que l'on peut repérer la couche de liens même lorsque tous les liens sont affichés. L'image B) affiche les liens internes à ce paquetage, faisant ressortir ainsi la couche de liens.



(A) (B)

Figure 5.20 – Exemple d'un paquetage isolé dans *Azureus*. L'image A) montre le paquetage par rapport au reste du logiciel. L'image B) sélectionne le paquetage afin d'afficher seulement ses liens : nous pouvons voir qu'il ne communique presque pas avec le reste du logiciel.

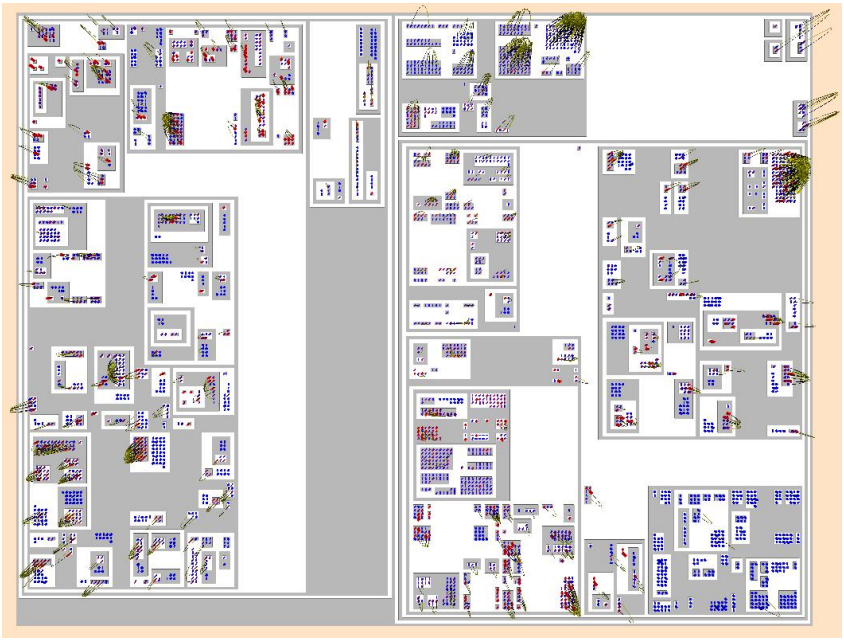
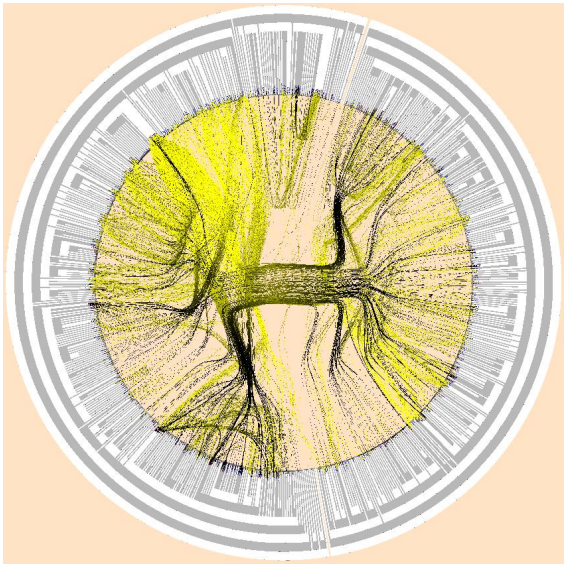
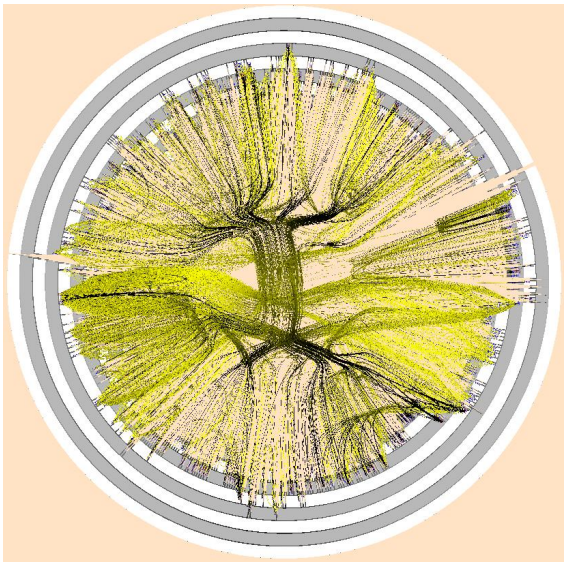


Figure 5.21 – Liens d'invocation internes de *Azureus*.



(A)



(B)

Figure 5.22 – Vue d'ensemble des liens d'invocation dans *Azureus* avec les placements radial et Colisée. L'image A) utilise le placement radial et l'image B) le placement Colisée.

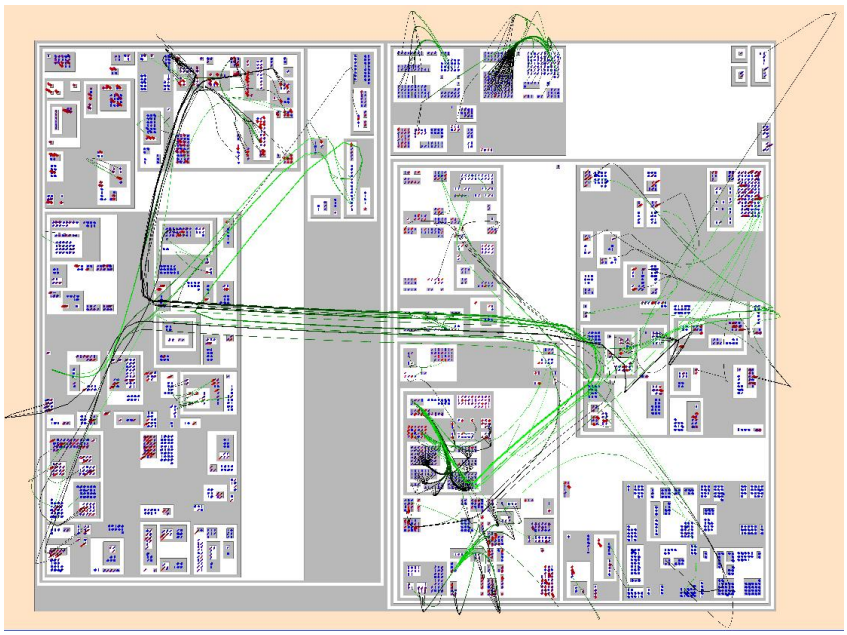
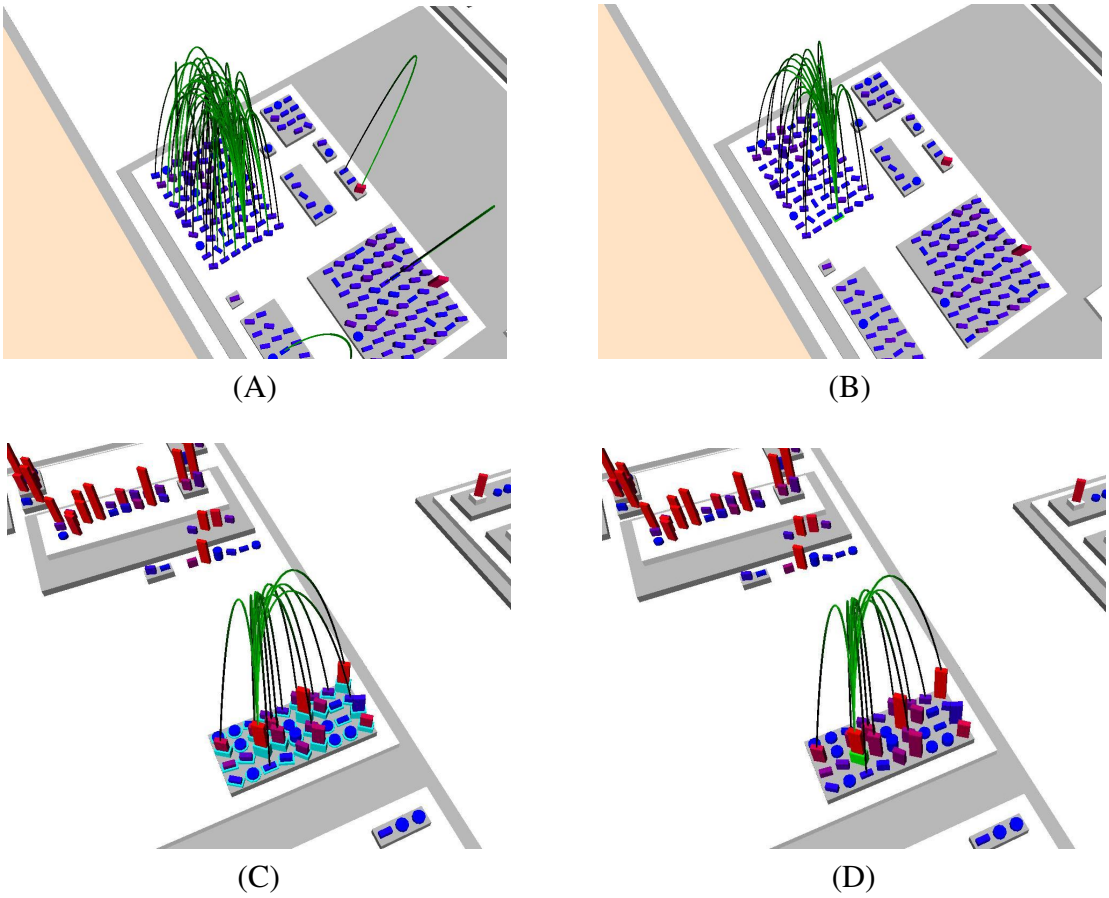


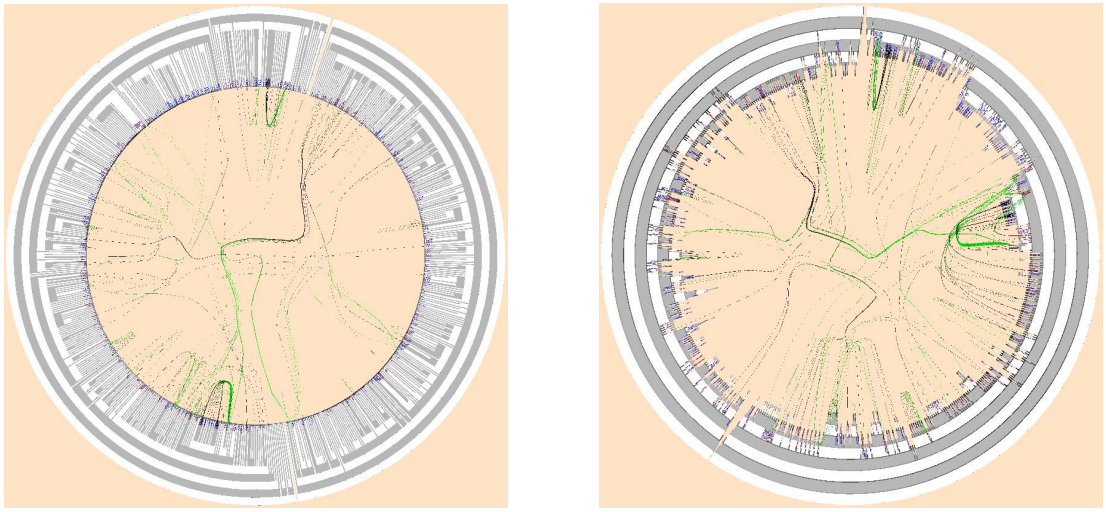
Figure 5.23 – Vue d'ensemble des liens d'héritage dans *Azureus*.



(A) (B)

(C) (D)

Figure 5.24 – Exemple de « fontaines » de liens dans *Azureus*. Les images de la colonne de droite montrent les liens obtenus après avoir sélectionné la classe suspectée d'être la source de la « fontaine ». Dans chaque cas, cela permet de confirmer nos soupçons.



(A) (B)

Figure 5.25 – Vue d'ensemble des liens d'héritage d'*Azureus* avec les placements radial et Colisée. L'image A) utilise le placement radial et l'image B) le placement Colisée.

CHAPITRE 6

CONCLUSION

Dans de ce mémoire, nous avons présenté un système de visualisation pouvant afficher simultanément des liens hiérarchiques et des liens d'adjacence entre les éléments d'un logiciel. Nos critères dans l'élaboration de cette visualisation étaient que l'affichage simultané de ces deux types de liens ne génère pas trop d'encombrement visuel, que la visualisation soit extensible et qu'elle offre à l'utilisateur diverses façons d'interagir avec elle afin de filtrer l'information visualisée.

Nous avons présenté les trois types de placements utilisés par notre système de visualisation, chacun d'eux ayant été conçu de façon à pouvoir être affiché en simultané avec des liens d'adjacence sans trop générer d'encombrement visuel et sans causer trop d'interférence avec la représentation utilisée pour afficher ces liens. Le premier type de placement présenté était une version améliorée de la visualisation *Treemap* par imbrication standard. Les améliorations apportées à celle-ci, tels placer les paquetages sur une grille et éloigner davantage les éléments, servaient avant tout à la rendre mieux adaptée à l'ajout de liens d'adjacence entre ses éléments. Nous avons aussi modifié certains attributs visuels du placement afin d'expliciter davantage la structure de la hiérarchie dans celui-ci. Nous avons ensuite présenté le placement radial, qui était essentiellement le même type de placement que celui présenté dans [10]. Par contre, nous avons ajouté à ce placement les mêmes modifications d'attributs visuels que pour le placement basé sur le *Treemap*, encore une fois dans le but d'expliciter davantage la structure hiérarchique présentée. Finalement, nous avons présenté le placement Colisée, qui est un placement circulaire dérivé du placement radial. Les principales différences que le placement Colisée a avec ce dernier est qu'il place les classes sur le niveau de leur parent, et les paquetages sont de tailles décroissantes de l'extérieur vers l'intérieur.

Pour afficher les liens d'adjacence, nous avons intégré l'algorithme *HEB* à notre visualisation tridimensionnelle. Nous avons défini, pour chaque type de placement, la façon de positionner les points de contrôle des liens dans l'espace 3D de sorte que leur

regroupement soit cohérent avec la hiérarchie et fasse ressortir les tendances dans l'ensemble des liens du logiciel. Nous avons modifié légèrement l'algorithme *HEB* en incluant un nœud de sortie supplémentaire aux groupes de classes de façon à pouvoir distinguer plus facilement les liens entrant des liens sortant de celui-ci. Nous avons défini comment utiliser la couleur et la taille de chaque lien afin d'en indiquer la direction et de quantifier ces liens lorsqu'ils forment un paquet. Nous avons intégré une métaheuristique à nos algorithmes de placement, qui détermine un ordre de placement des éléments afin de réduire davantage l'encombrement visuel dans les liens d'adjacence. Notre visualisation offre aussi toute une gamme de méthodes interactives permettant à l'utilisateur de filtrer l'information qui lui est présentée, et ce tant au niveau de la hiérarchie que des liens d'adjacence. Ces méthodes interactives permettent à l'utilisateur de se concentrer sur l'information pouvant l'aider dans sa tâche.

Finalement, nous avons fait une évaluation de notre système de visualisation en le testant avec trois logiciels composés de plus de 2000 classes chacun. Les résultats obtenus par l'analyse exploratoire de ces logiciels avec notre système de visualisation nous ont permis de faire certaines découvertes intéressantes, de vérifier que celui-ci s'adapte bien à des logiciels de cette taille et de confirmer l'utilité des différentes méthodes interactives proposées pour filtrer les informations offertes par la visualisation.

Malgré les résultats prometteurs obtenus avec notre validation, plusieurs améliorations pourraient encore être apportées à notre système de visualisation dans l'avenir. Premièrement, la réalisation d'une expérience contrôlée en proposant une série de tâches pour chaque placement nous permettrait non seulement d'évaluer l'utilité de notre système de visualisation en général pour résoudre ces tâches, mais aussi de comparer l'efficacité des placements entre eux.

Nous pourrions utiliser la couleur des liens entre les classes pour les quantifier, comme expliqué à la section 4.3.4.2. Cela nous permettrait de visualiser certains types de liens qui ne sont pas nécessairement unitaires entre les classes. Pour les liens entre les paquetages, nous pourrions aussi développer une façon de calculer automatiquement une taille d'affichage maximale des liens qui soit adaptée à la taille du logiciel. Cela éviterait à l'utilisateur d'avoir à la déterminer manuellement pour chaque logiciel visualisé.

Finalement, la façon proposée dans ce mémoire pour déterminer la valeur d'une solution dans l'algorithme du recuit simulé, bien qu'ayant apporté une certaine amélioration à la visualisation des liens d'adjacence, reste assez simpliste. Il pourrait être intéressant d'investiguer d'autres façons d'évaluer une solution qui pourraient améliorer davantage la visualisation des liens d'adjacence.

BIBLIOGRAPHIE

[1] Hani Abdeen, Ilham Alloui, Stephane Ducasse, Damien Pollet et Mathieu Suen. Package reference fingerprint : a rich and compact visualization to understand package relationships. Dans *Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering*, pages 213–222, Washington, DC, USA, 2008. IEEE Computer Society.

[2] Mark Bruls, Kees Huizing et Jarke van Wijk. Squarified treemaps. Dans *In Proceedings of the Joint Eurographics and IEEE TCVG Symposium on Visualization*, pages 33–42. Press, 1999.

[3] Pierre Caserta, Olivier Zendra et Damien Bodénès. 3D Hierarchical Edge Bundles to Visualize Relations in a Software City Metaphor. Dans *6th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2011)*, Williamsburg, États-Unis, septembre 2011.

[4] Bas Cornelissen, Danny Holten, Andy Zaidman, Leon Moonen, Jarke J. van Wijk et Arie van Deursen. Understanding execution traces using massive sequence and circular bundle views. Dans *IN PROC. 15TH INT. CONF. ON PROGRAM COMPREHENSION (ICPC*, pages 49–58. IEEE, 2007.

[5] Weiwei Cui, Hong Zhou, Huamin Qu, Pak Chung Wong et Xiaoming Li. Geometry-based edge clustering for graph visualization. *IEEE Transactions on Visualization and Computer Graphics*, 14:1277–1284, 2008.

[6] Orla Greevy, Michele Lanza et Christoph Wysseier. Visualizing live software systems in 3d. Dans *Proceedings of the 2006 ACM symposium on Software visualization*, SoftVis '06, pages 47–56, New York, NY, USA, 2006. ACM.

[7] Christopher G. Healey et James T. Enns. Attention and visual memory in visualization and computer graphics. *IEEE Transactions on Visualization and Computer Graphics*, 99, 2011.

[8] I. Herman, G. Melancon et M.S. Marshall. Graph visualization and navigation in information visualization : A survey. *Visualization and Computer Graphics, IEEE Transactions on*, 6(1):24 –43, jan-mar 2000.

[9] D. Holten, B. Cornelissen et J.J. van Wijk. Trace visualization using hierarchical edge bundles and massive sequence views. Dans *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 47 –54, june 2007.

[10] Danny Holten. Hierarchical edge bundles : Visualization of adjacency relations in hierarchical data. *IEEE Transactions on Visualization and Computer Graphics*, 12: 741–748, 2006.

[11] Danny Holten et Jarke J. Van Wijk. Force-directed edge bundling for graph visualization. *Computer Graphics Forum*, 28(3):983–990, 2009.

[12] Yuntao Jia, Michael Garland et John C. Hart. Hierarchical edge bundles for general graphs, 2009.

[13] Brian Johnson et Ben Shneiderman. Tree-maps : a space-filling approach to the visualization of hierarchical information structures. Dans *Proceedings of the 2nd conference on Visualization '91*, VIS '91, pages 284–291, 1991.

[14] S. Kirkpatrick, C. D. Gelatt et M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[15] Ernst Kleiberg, Huub van de Wetering et Jarke J. Van Wijk. Botanical visualization of huge hierarchies. *Information Visualization, IEEE Symposium on*, 0:87, 2001.

[16] Guillaume Langelier. Visualisation de la qualité des logiciels de grandes tailles. M.sc. thesis, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, décembre 2006.

[17] Guillaume Langelier, Houari A. Sahraoui et Pierre Poulin. Visualisation and analysis of software quantitative data. Dans *Proceedings 9th ECOOP Workshop on*

Quantitative Approaches in Object-Oriented Software Engineering, QAOOSE '05, 2005.

[18] Doantam Phan, Ling Xiao, Ron Yeh, Pat Hanrahan et Terry Winograd. Flow map layout. Dans *Proceedings of the IEEE Symposium on Information Visualization*, pages 219–224, 2005.

[19] Jun Rekimoto et Mark Green. The information cube : Using transparency in 3d information visualization. Dans *In Proceedings of the Third Annual Workshop on Information Technologies & Systems*, 1993.

[20] George G. Robertson, Jock D. Mackinlay et Stuart K. Card. Cone trees : animated 3d visualizations of hierarchical information. Dans *Proceedings of the SIGCHI conference on Human factors in computing systems : Reaching through technology*, CHI '91, pages 189–194, New York, NY, USA, 1991. ACM.

[21] Ben Shneiderman. Tree visualization with tree-maps : A 2-d space-filling approach. *ACM Transactions on Graphics*, 11:92–99, 1991.

[22] JOHN STASKO, RICHARD CATRAMBONE, MARK GUZDIAL et KEVIN MCDONALD. An evaluation of space-filling information visualizations for depicting hierarchical structures. *International Journal of Human-Computer Studies*, 53(5): 663 – 694, 2000.

[23] John Stasko et Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. *Information Visualization, IEEE Symposium on*, 0:57, 2000.

[24] A. Telea, H. Hoogendorp, O. Ersoy et D. Reniers. Extraction and visualization of call dependencies for large c/c++ code bases : A comparative study. Dans *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, pages 81 –88, sept. 2009.

[25] A. Telea, H. Hoogendorp, O. Ersoy et D. Reniers. Extraction and visualization of call dependencies for large c/c++ code bases : A comparative study. Dans *Visualizing Software for Understanding and Analysis, 2009. VISSOFT 2009. 5th IEEE International Workshop on*, pages 81 –88, 2009.

[26] Frank van Ham et Jarke J. van Wijk. Beamtrees : Compact visualization of large hierarchies. *Information Visualization*, 2(1):31–39, 2003.

[27] J.J. Van Wijk et H. Van de Wetering. Cushion treemaps : visualization of hierarchical information. Dans *Information Visualization, 1999. (Info Vis '99) Proceedings. 1999 IEEE Symposium on*, pages 73 –78, 147, 1999.

[28] V. Černý. Thermodynamical approach to the traveling salesman problem : An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[29] Weixin Wang, Hui Wang, Guozhong Dai et Hongan Wang. Visualization of large hierarchical data by circle packing. Dans *Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 517–520, New York, NY, USA, 2006. ACM.

[30] Richard Wettel et Michele Lanza. Program comprehension through software habitability. Dans *Proceedings of the 15th IEEE International Conference on Program Comprehension*, pages 231–240, 2007.