Université de Montréal

Direction des bibliothèques

AVIS

Ce document a été numérisé par la Division de la gestion des documents et des archives de l'Université de Montréal.

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.

Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

This document was digitized by the Records Management & Archives Division of Université de Montréal.

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.

Université de Montréal

Un système d'aide à la visualisation interactive de logiciels

par
Salima Hassaine

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

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

Décembre, 2007

© Salima Hassaine, 2007.

Université de Montréal
Faculté des études supérieures

Ce mémoire intitulé:

Un système d'aide à la visualisation interactive de logiciels

présenté par:

Salima Hassaine

a été évalué par un jury composé des personnes suivantes:

Victor Ostromoukhov,	président-rapporteur
Houari Sahraoui,	directeur de recherche
Pierre Poulin,	codirecteur
Sylvie Hamel,	membre du jury

Mémoire accepté le: 8 février 2008

RÉSUMÉ

La visualisation est un moyen efficace et flexible pour inspecter et analyser des données de logiciels à plusieurs niveaux de détail. Cependant, la construction d'une visualisation efficace nécessite de l'utilisateur de comprendre les règles de la perception visuelle humaine. Malheureusement, les spécialistes en génie logiciel n'ont pas souvent assez de connaissances dans les domaines de la visualisation et de la science cognitive, et ils doivent donc faire appel à des experts pour les aider à visualiser leurs données. Dans ce mémoire, nous adoptons une approche centrée sur les besoins d'utilisateur afin de l'assister durant tout le processus de la visualisation. Notre approche comporte deux parties. La première partie consiste à construire des visualisations efficaces pour des données multidimensionnelles. La deuxième partie consiste à aider les utilisateurs à construire des visualisations spécifiques pour réaliser leurs tâches d'analyse. Pour résoudre ces problèmes nous proposons d'utiliser des textures pour la visualisation de données multidimensionnelles, et une approche pour aider les utilisateurs à accomplir leurs tâches d'analyse. À cet effet, nous utilisons des taxonomies des tâches analytiques de bas niveau, des tâches interactives à haut niveau et des règles de la perception pour concevoir un générateur, qui transforme une tâche d'analyse décrite dans un langage de haut niveau en un scénario d'utilisation pour un environnement de visualisation. Finalement, nous illustrons notre approche avec un problème particulier de détection d'anomalies en utilisant la visualisation et les métriques de code.

Mots clés: visualisation de logiciels, qualité, métriques, perception visuelle, tâches d'analyse, interaction.

ABSTRACT

Software visualization is an efficient and flexible way to inspect and analyze software data at various levels of detail. However, software analysts typically do not have a sufficient background in visualization and cognitive science to select efficient representations and parameters without the help of visualization experts. To overcome this problem, our work focuses on how to construct visualizations for multi-dimentional data, and how to help users construct perceptually salient visualizations to display their data and achieve their analysis tasks. To solve these issues we suggest to use textures for multi-dimensional data visualization, and propose an approach to automatically generate software analysis tasks that use visualization. To this end, we use taxonomies of low-level analytic tasks, high-level interactive tasks, and perceptual rules to design an assistant that proposes analysis scenarios. We illustrate our approach on the particular problem of anomaly detection using visualization and code metrics.

Keywords: software visualization, quality, metrics, visual perception, analysis task, interaction.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

CBO	*Coupling Between Objects*
CSP	*Constraint Satisfaction Problem*
DIT	*Depth in Inheritance Tree*
LCOM5	*Lack of COhesion in Method Version 5*
VERSO	Visualisation pour l'Evaluation et la rétro-ingénierie du SOftware
WMC	*Weighted Methods per Class*
UML	*Unified Modeling Langage*

À mes parents.

À ma sœur Nacima.

À mes frères Amine et Aymène.

REMERCIEMENTS

J'adresse mes sincères remerciements au président du jury Victor Ostromoukhov et au membre du jury Sylvie Hamel, pour avoir accepté d'évaluer ce travail.

Je remercie également mes directeur et co-directeur Houari Sahraoui et Pierre Poulin pour m'avoir donné la chance de faire de la recherche sous leur direction. J'aimerais également les remercier pour leur aide précieuse tout au long du projet ainsi que pour le temps qu'ils ont consacré à la correction des versions préliminaires de ce mémoire.

Je remercie aussi tous les membres du laboratoire GEODES et tout particulièrement le professeur Yann-Gaël Guéhéneuc, pour ces nombreux conseils.

Ma reconnaissance va aussi à Guillaume Langelier, Karim Dhambri et Stephane Vaucher, chacune de nos discussions et leurs multiples conseils ont énormément contribué à l'enrichissement de ma réflexion. J'aimerais les remercier ainsi que Duc-Loc Huynh pour leur bonne humeur, et leurs encouragements ont été des facteurs indispensables à la réussite de cette maîtrise.

J'adresse une pensée toute particulière et je témoigne toute mon affection à mes chers parents qui ont su m'apporter tout leur soutien, leurs pensées, leurs prières et leur amour à distance mais à tout instant.

Finalement, je veux aussi remercier ma sœur Nacima et mes amies Alicia Heraz, Assia Bouguelmouna et Mariame Dhambri pour leur soutien pendent les moments difficiles. Sans vous ce travail n'aurait jamais pu aboutir, je ne l'oublierai jamais. Merci beaucoup.

CHAPITRE 1

INTRODUCTION

Nous décrivons brièvement dans ce chapitre, les problèmes qui ont motivé cette recherche, le but de notre travail ainsi que les solutions que nous proposons. Nous terminons en donnant un aperçu sur les différents chapitres du mémoire.

1.1 Contexte

Les systèmes industriels actuels nécessitent le développement de logiciels de plus en plus complexes et exigent de leur part qu'ils soient de très bonne qualité du point de vue de la maintenabilité et de la réutilisabilité. Dans l'industrie, le coût de maintenance d'une application informatique est estimé à 50% et 75% de son coût total [37]. On estime également que plus de la moitié de cette maintenance est consacrée à la compréhension de l'application elle-même [8]. La maîtrise de la compréhension des applications informatiques s'avère donc indispensable.

Par ailleurs, le développement orienté objet est devenu un standard du développement des logiciels. Les concepts introduits par cette approche ont apporté une réponse à de nombreux problèmes du génie logiciel. Ils permettent, en particulier, une modélisation plus fidèle de la réalité et une meilleure maîtrise de la complexité des logiciels, ainsi qu'une réutilisation plus facile des artefacts produits. Bien que cette approche ait rencontré un grand succès en production de logiciels complexes, l'expérience pratique avec de grands projets a montré que les programmeurs font toujours face à des difficultés dans la maintenance de leur code [33]. Par conséquent, de nouvelles méthodes et de nouveaux outils d'évaluation et de compréhension de la qualité des logiciels sont nécessaires pour aider aussi bien les gestionnaires que les développeurs dans leurs activités de maintenance. Les travaux effectués, entre autres, par Chidamber et Kemerer [7] ou encore Fenton [14], ont été une véritable révolution dans ce domaine en proposant des techniques et des métriques pour l'évaluation de la qualité des programmes orientés

objet. Cependant, ces techniques d'évaluation et de compréhension de la qualité des logiciels nécessitent la manipulation et l'analyse d'une grande masse de données, ce qui n'est pas une tâche facile.

D'autre part, la visualisation de données est un domaine assez prometteur. Son but est d'exploiter les capacités du système visuel humain pour fournir à l'utilisateur une compréhension qualitative du contenu de l'information. L'information peut se présenter sous forme de données, de processus, de relations ou de concepts. La représentation graphique peut nécessiter la manipulation des entités graphiques (points, lignes, formes, surfaces, etc.) et des attributs (couleur, intensité, taille, position, forme, etc.). La visualisation des logiciels est une application de la visualisation de données. Elle offre une réponse possible au problème de la communication de l'information au programmeur. Elle aide à mieux comprendre et percevoir les relations et les interactions à l'intérieur du logiciel. Dans ce domaine, l'espace d'informations visualisées est fortement structuré, mais la complexité naît du fait qu'il n'existe pas une structure unique pour cet espace, mais bel et bien plusieurs structures significatives ou pertinentes du point de vue de l'utilisateur. De ce fait, l'espace des données est extrêmement riche. Pour comprendre la complexité du phénomène observé, il convient de restituer la richesse de cet espace.

1.2 Caractérisation du problème

Les systèmes de visualisation ont grandement évolué au cours de ces dernières années du fait de la croissance considérable de la quantité de données à analyser dans différents domaines scientifiques. Cette évolution profonde des systèmes de visualisation doit s'accompagner d'une transformation des outils utilisés pour faciliter l'accès à l'information. En effet, les outils existants ne permettent toujours pas de pallier la complexité grandissante des données visualisées. Chacun de ces outils exige de l'utilisateur de spécifier explicitement les paramètres de la visualisation. La construction d'une visualisation efficace nécessite de l'utilisateur de comprendre les règles de constructions graphiques issues des sciences cognitives et de la perception visuelle humaine. Malheureusement, les spécialistes en génie logiciel n'ont pas assez de connaissances dans ces

domaines, et ils font donc souvent appel à des experts pour les aider à visualiser leurs données et à utiliser les outils disponibles de la manière la plus efficace possible.

En conclusion, la visualisation est un outil puissant qui peut être limité par deux problèmes majeurs : l'explosion de la quantité de données soumises à l'utilisateur et le mode de représentation qui peut être inadéquat aux tâches de l'utilisateur. Un problème important des chercheurs dans ce domaine est de trouver de nouvelles métaphores visuelles pour représenter les données et de comprendre quelles sont les tâches analytiques qu'elles peuvent supporter.

1.2.1 L'explosion de la quantité de données soumises à l'utilisateur

La plupart des systèmes de visualisation existants [12, 27, 28, 30, 34] sont limités quant aux dimensions de données qu'ils permettent de visualiser simultanément, et aussi quant au changement du niveau de granularité de la visualisation de manière la plus efficace vis-à-vis de l'utilisateur. Généralement, lorsque l'utilisateur souhaite avoir plus d'informations sur le logiciel visualisé, il doit effectuer un changement de paramètres de la visualisation. Certains de ces changements ont pour effet de rendre visibles ou invisibles certaines données. Souvent, ces changements brusques de visibilité rendent l'exploration plus difficile. Le changement du contexte de visualisation exige de l'utilisateur de construire une nouvelle représentation mentale cohérente de la scène visualisée, ce qui augmente l'effort cognitif nécessaire pour accomplir les tâches de l'utilisateur. L'une des solutions proposées pour ce problème est d'enrichir les représentations graphiques par plus d'indices visuels, pour visualiser plusieurs informations simultanément. En revanche, la multiplication des indices visuels (couleurs, textures, etc.) associés à chaque représentation graphique est susceptible de nuire à la perception de l'utilisateur, et d'introduire des ambiguïtés. Les outils de visualisation visant à enrichir l'espace d'information doivent aujourd'hui prendre en compte les règles issues des études menées en psychologie cognitive sur la perception visuelle, pour créer des visualisations plus efficaces.

1.2.2 Un mode de représentation inadéquat aux tâches de l'utilisateur

Lorsque l'utilisateur participe au processus de *traitement* des données (recherche, analyse, compréhension, etc.), il se trouve confronté, d'une part à une masse importante de données, et d'autre part à une information dont la représentation est inadaptée. Elle est inadaptée parce qu'elle a souvent été conçue sans prendre en compte le contexte. Les outils de visualisation visant à supporter les activités des tâches d'analyse doivent aujourd'hui intégrer le contexte afin de mieux répondre aux besoins des utilisateurs. La prise en compte du contexte permet de définir quelles sont les informations et les fonctionnalités dont l'utilisateur aura besoin, pour accomplir sa tâche d'analyse.

1.3 Contributions

Face à cette croissance continue de la quantité de données manipulées et à l'évolution profonde des systèmes de visualisation, il convient donc d'assister les utilisateurs dans leur tâche d'analyse. Nous proposons dans ce cadre :

- Améliorer la représentation graphique des logiciels par le raffinement du niveau de détails pour enrichir l'espace d'information,
- Permettre à l'utilisateur l'expression fine de tâches d'analyse en langage de haut niveau, en délégant à un système d'aide, la génération de la visualisation et des interactions les plus adaptées à ses besoins.

1.4 Plan du mémoire

Le chapitre 2 introduit les différents travaux préalables dans les domaines reliés à notre recherche. Les règles sur la perception humaine, les taxonomies de tâches d'analyse et d'interactions, et enfin, les travaux précédents concernant la visualisation de logiciels. Le chapitre 3 présente une discussion sur l'outil de visualisation VERSO à la base de nos travaux. Le chapitre 4 présente notre première contribution, une description détaillée sur l'intégration des textures et des représentations d'un niveau de granularité plus bas (niveau de méthodes) en VERSO. Le chapitre 5 décrit en détails l'approche que

nous proposons afin de concevoir un système d'aide à la visualisation, et montre l'utilisation de ce système pour l'analyse de logiciels de grande taille. Ceci est fait par le biais d'un exemple détaillé d'une tâche d'analyse, qui consiste à détecter une anomalie de conception du logiciel. Enfin, le chapitre 6 est une conclusion portant à la fois sur le bilan de notre recherche et sur l'identification de perspectives de nos travaux qui auront lieu suite à cette recherche.

CHAPITRE 2

CONTEXTE ET TRAVAUX CONNEXES

Nous nous intéressons dans ce chapitre aux différents travaux préalables dans les domaines reliés à notre recherche. Il est articulé en trois parties. La première partie de ce document est dédiée à la caractérisation des représentations graphiques. La seconde partie présente les techniques d'interaction. Enfin, la troisième partie présente les travaux antérieurs en visualisation de logiciels.

Dans la première partie (Section 2.1), nous allons identifier les composantes du système d'expression (*les variables visuelles*). Cette partie présente les travaux fondamentaux de Jacques Bertin, qui constituent une référence incontournable à ce niveau. Pour visualiser une information, il est nécessaire de disposer d'un moyen d'exprimer graphiquement cette information, mais aussi de connaître les règles qui gouvernent la perception et la compréhension de ce mode d'expression. Le système visuel humain comporte des forces et des faiblesses, et il est important de les prendre en considération lors de la conception d'un système de visualisation. La section 2.1.1 présente les facteurs de la perception visuelle, qui peuvent induire l'utilisateur en erreur, ainsi que les caractéristiques du système visuel humain, dont il est possible de tirer profit pour construire des visualisations efficaces. La section 2.1.2 étudie ces variables visuelles en tant que moyen d'expression. Cette étude s'intéresse à la manière dont l'information visuelle est perçue et interprétée ; elle définit les propriétés visuelles qui font que certaines variables sont mieux adaptées que d'autres pour représenter certaines données. Dans la section 2.1.3, nous présentons et comparons plusieurs techniques de visualisation d'informations visant à automatiser les règles de constructions graphiques.

Après avoir défini les règles de constructions graphiques, nous devons comprendre comment l'utilisateur peut interagir avec un système de visualisation. La seconde partie de ce chapitre (Section 2.2) présente les différentes taxonomies de tâches d'analyse et techniques d'interaction, qu'un outil de visualisation doit supporter pour répondre aux besoins d'utilisateur.

La troisième partie de ce chapitre (Section 2.3) est dédiée à l'état de l'art des outils de visualisation. Dans cette section, nous présentons un survol sur les outils de visualisation de logiciels. Quelque soit le type d'information recherché lors de l'analyse d'un logiciel, deux problèmes récurrents à ce domaine ressortent fréquemment, la dimension et la granularité des données récoltées. Enfin, cette section finit par la mise en correspondance des différents travaux sur la visualisation des données multidimensionnelles et la visualisation du plus bas-niveau de granularité (méthodes), pour définir le cadre de nos réflexions et identifier une problématique précise.

2.1 Principes de la construction graphique

Bertin s'intéresse à la construction de visualisation par les symboles graphiques. Ses travaux s'attachent à définir les principes de la construction de cartes géographiques. Il a défini un langage de signes dont les règles sont basées sur les propriétés de notre perception des signes employés.

D'après Bertin [2], une représentation graphique n'est constituée que par adjonction et juxtaposition de taches dans le plan. Les taches se distinguent essentiellement par l'interprétation géométrique élémentaire qu'on leur associe par rapport aux deux dimensions du plan : une tache peut représenter un point, une ligne ou une zone. C'est ce que Bertin nomme l'implantation (ponctuelle, linéaire ou zonale). Le lecteur de la carte perçoit six variations sensibles attachées aux taches (symboles de la carte) : en taille, en valeur (ou niveau de gris), en grain, en teinte (ou couleur), en orientation, ou en forme (Figure 2.1). Il les appelle variables visuelles, 'composantes du système d'expression'. Une fois identifiées ces variables visuelles et leurs particularités perceptives, il devient possible d'exprimer graphiquement une information.

2.1.1 Les facteurs de la perception préattentive

La notion d'indices visuels, employée ici, n'est pas totalement disjointe de celle de variables visuelles. Cependant, nous réservons le terme de variables visuelles aux variables précédemment identifiées, et celui d'indices visuels (qui englobe un ensemble

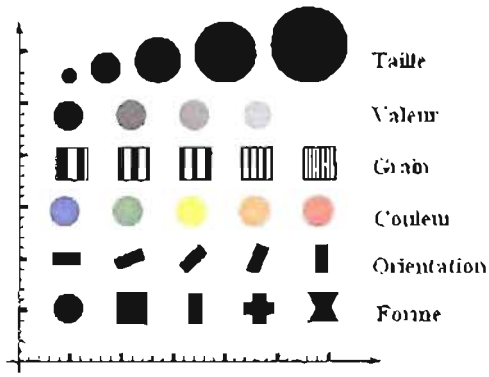

Figure 2.1 – Les variables visuelles - d'après Bertin. Chaque tache employée dans un graphique peut varier en taille, en valeur, en grain, en couleur, en orientation, en forme ou en position dans le plan dans lequel est construite la représentation.

plus large) aux informations visuelles contenues dans l'image et qui influent sur la manière dont celle-ci est perçue. A titre d'exemple, la couleur, la forme et la taille sont des variables visuelles qui sont également des indices visuels. À l'inverse, la régularité des formes ou la fréquence spatiale du tracé sont des facteurs qui influencent la perception de l'image (qui sont donc des indices visuels) mais qui ne sont pas des composantes du système d'expression (*les variables visuelles*).

La perception préattentive de certaines variables visuelles est un phénomène qui a donné lieu à plusieurs tentatives d'explications [19] : il existe des indices visuels dont la perception est extrêmement rapide. Il s'agit d'indices visuels dont la perception est possible en un temps très court sans qu'il soit nécessaire, contrairement aux autres indices visuels, d'examiner un par un tous les éléments de l'image. Les études expérimentales menées dans ce domaine ont permis d'identifier les indices visuels qui rendent possible ce phénomène, les interférences qui peuvent intervenir entre ces différents indices, et enfin les tâches supportées dans des conditions de perception préattentive. C'est à ces résultats que nous nous intéressons et que nous présentons brièvement ici.

2.1.1.1 Relations perçues en condition préattentive

Chaque indice visuel peut prendre plusieurs valeurs différentes. L'ensemble des valeurs d'un indice visuel constitue la variation de cet indice. Cette variation définit l'ensemble des catégories auxquelles peut appartenir l'indice visuel. Dans des conditions de perception préattentive, il est possible de détecter un élément (une cible), de regrouper des éléments similaires ou encore d'estimer la quantité d'éléments appartenant à une catégorie donnée. La détection d'une cible consiste à repérer un élément graphique qui n'appartient pas à la même catégorie de l'indice étudié que ceux parmi lesquels il est représenté.

La Figure 2.2 illustre un exemple de détection d'un disque noir, au milieu d'éléments appartenant à une autre catégorie du même indice, ici des disques blancs. Ce phénomène n'est pas limité à la variation de couleur, mais a également été observé et expérimenté dans le cas de l'orientation, de la taille, de la courbure, de la connexité, de l'intersection, de la valeur, de la direction de déplacement et du scintillement [18, 19].

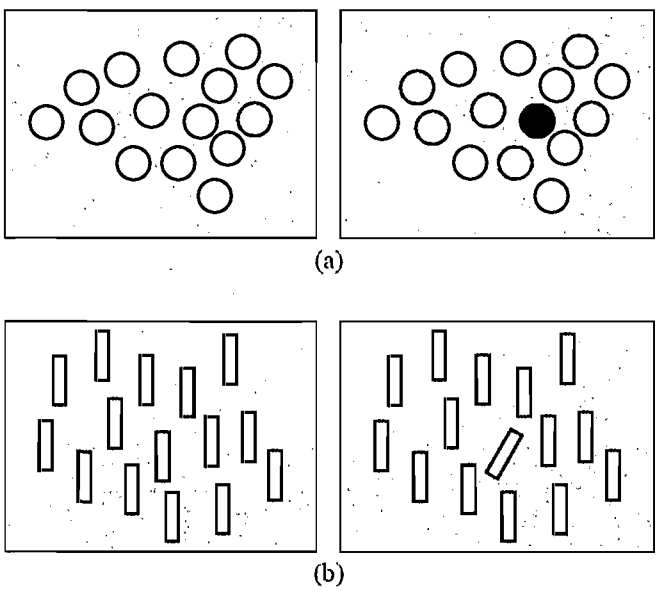

Figure 2.2 – Perception préattentive de la couleur en (a) et de l'orientation en (b) [5].

2.1.1.2 Interférence entre les indices visuels

Bertin met en évidence la notion d'efficacité (ou de prégnance) qu'il définit ainsi : *"Si pour obtenir une réponse correcte et complète à une question donnée, et toutes choses égales, une construction requiert un temps d'observation plus court qu'une autre construction, on dira qu'elle est plus efficace pour cette question"* [2].

L'utilisation combinée de plusieurs indices visuels dans un même graphique entraîne des effets d'interférences qui peuvent rendre inefficaces certains indices visuels [6]. Par exemple, l'effet de l'efficacité de la couleur interfère avec celui de la forme alors que l'inverse n'est pas vrai (Figure 2.3). Il en est de même pour la variation de brillance qui interfère avec la variation de teinte tandis que l'inverse n'est pas vrai.

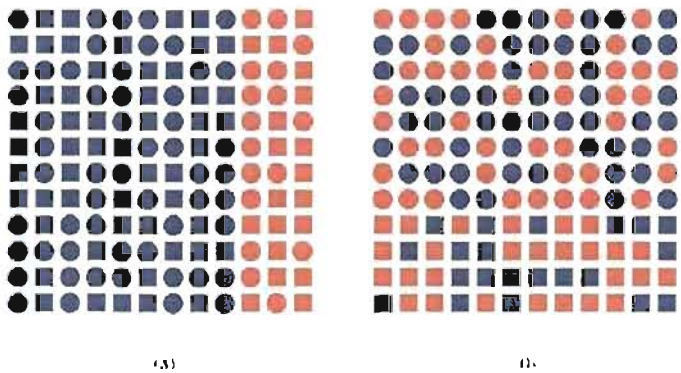

Figure 2.3 – Interférence de la couleur et de la forme [19]. (*a*) Détection d'une frontière grâce à l'effet de prégnance de la couleur sans interférence due à la variation de forme des objets. (*b*) Interférence de l'effet de prégnance de la forme due à la variation de la couleur.

Les interférences entre indices existent également en dehors de l'effet de prégnance. Garner [15] a mis en évidence le fait que certains indices visuels sont corrélés : ils ne peuvent être distingués qu'au prix d'un effort perceptif de la part de l'observateur. Un changement selon ces deux indices sera perçu comme un unique changement et non comme la variation de deux indices visuels. La largeur et la hauteur d'un signe sont des indices corrélés. À l'inverse, la forme et la couleur sont des indices orthogonaux.

2.1.1.3 Influence de la visibilité et l'asymétrie de la perception

L'effet de prégnance n'est pas non plus identique pour toutes les catégories d'un même indice visuel. Si l'on considère le problème de la détection d'une cible parmi un ensemble d'éléments graphiques appartenant à une catégorie différente, alors la catégorie à laquelle appartient la cible influence la facilité avec laquelle elle sera perçue. La Figure 2.4 illustre ce phénomène pour l'indice visuel 'taille' : les deux catégories de taille en présence ne présentent pas les mêmes facilités de détection selon que la cible appartient à la catégorie des objets de petite taille ou à celle de grande taille.

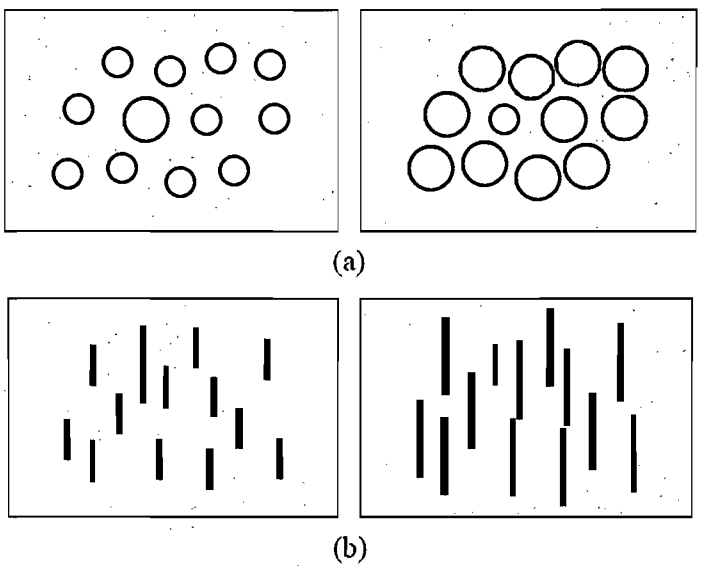

Figure 2.4 – Asymétrie de l'effet de prégnance de la taille et de la surface [5]. L'effet de prégnance de la taille est différent selon la catégorie de la taille en (a). Même effet dans le cas de la longueur en (b).

2.1.2 L'expression graphique d'une information

La visualisation d'informations consiste à donner une représentation graphique à des entités ne possédant pas nécessairement une transcription visuelle directe. La représentation graphique s'appuie sur un système d'expression visuel basé sur des variables visuelles que nous avons identifiées (Section 2.1). La visualisation consiste alors à passer

de l'espace des informations à une représentation graphique qui traduit, grâce aux variables visuelles utilisées, l'information originale. C'est cette étape de traduction ou de *transcription* de l'information vers un espace de représentation visuel qui nous intéresse.

Pour exploiter correctement ce mode d'expression, il est indispensable de définir un critère précis, mesurable, qui nous permet de classer les constructions graphiques, de définir la meilleure et d'expliquer, s'il y a lieu, pourquoi certains utilisateurs préfèrent une construction et certains une autre. Nous nous intéressons dans ce qui suit aux trois aspects fondamentaux de la construction graphique dont nous avons déjà parlé : les variables visuelles utilisées pour transcrire ces données, le type des données représentées et la perception de ces variables.

2.1.2.1 Caractérisation des composantes du système d'expression

Bertin [2] a proposé une caractérisation des variables visuelles qui tient compte de leurs propriétés perceptives. Cette caractérisation comporte quatre niveaux d'organisation :

- **Variable associative** ($\equiv$) : lorsqu'elle permet de regrouper spontanément toutes les correspondances différenciées par cette variable. L'associativité nécessite que le système perceptif visuel soit capable de faire abstraction des variations de l'indice visuel pour regrouper tous les objets affectés par celle-ci. Cette propriété doit être rapprochée de la notion de visibilité. Lorsque des éléments graphiques apparaissent avec la même puissance visuelle, ils ont la même visibilité. Une variable associative est une variable qui ne fait pas varier la visibilité. Selon Bertin, toutes les variables peuvent être associatives. La valeur et la taille en revanche ne le sont pas (Figure 2.5) ; on les dit dissociatives ($\not\equiv$).
- **Variable sélective** ($\not=$) : lorsqu'elle permet d'isoler spontanément toutes les correspondances appartenant à une même catégorie (de cette variable). Les éléments graphiques appartenant à une même catégorie forment une famille que l'on peut distinguer "spontanément" (autrement dit sans faire de recherche visuelle élément par élément). Seule la forme n'est pas sélective. La Figure 2.6 illustre cette propriété dans le cas des variables forme et valeur.

- **Variable ordonnée (O)** : lorsque ses différentes variations peuvent être classées selon un ordre qui s'impose spontanément. Grain, valeur et taille sont des variables ordonnées.
- **Variable quantitative (Q)** : Lorsque la différence de perception qui existe entre deux variations d'une variable visuelle peut être exprimée par un rapport numérique (distance visuelle). Par exemple, la taille est typiquement une variable quantitative.

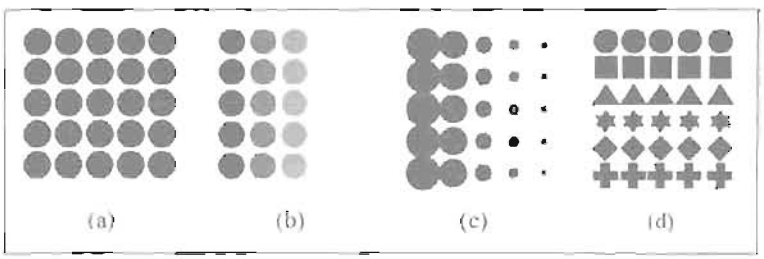

Figure 2.5 – Perception associative [5].(*a*) Une série de points semblables forme une zone homogène. (*b*) et (*c*) La valeur et la taille modifient la visibilité des points, leur perception est dissociative. (*d*) La perception de la forme est associative.

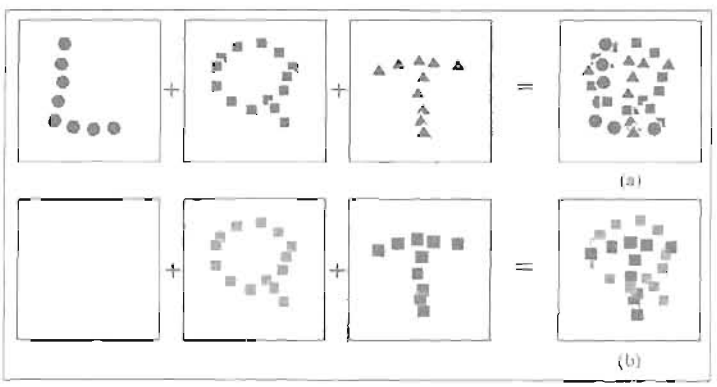

Figure 2.6 – Perception non sélective et sélective [5]. Les trois figures à gauche de l'image sont composées d'éléments graphiques appartenant à des catégories différentes d'un même indice visuel. (*a*) L'identification des trois figures d'origine est difficile car la perception de la forme est non sélective. (*b*) À l'inverse la valeur dont la perception est sélective permet de séparer les trois figures d'origine.

Bertin dresse ainsi le tableau des associations (*variables - niveaux d'organisation*) que le lecteur trouvera dans le Tableau 2.1 :

	Associative	Sélective	Ordonnée	Quantitative
Position	$\equiv$	$\neq$	**O**	**Q**
Taille	$\not\equiv$	$\neq$	**O**	**Q**
Valeur	$\not\equiv$	$\neq$	**O**	
Grain	$\equiv$	$\neq$	**O**	
Couleur	$\equiv$	$\neq$		
Orientation	$\equiv$	$\neq$		
Forme	$\equiv$			

Tableau 2.1 – Niveaux d'organisation des variables visuelles [2].

2.1.2.2 Les règles d'usage des variables visuelles

Mackinlay [29] divise l'espace de données en données qualitatives, ordonnées et quantitatives. Il développe un langage graphique pour codifier une représentation, et construit des échelles d'expressivité et d'efficacité pour évaluer des variables visuelles. Une variable est expressive si elle transcrit dans le domaine visuel toutes les informations nécessaires à l'accomplissement d'une tâche visuelle, et seulement ces informations. Une variable est efficace si elle permet de représenter une donnée de la manière la plus juste vis-à-vis les capacités du système visuel humain. Le Tableau 2.2 résume ces travaux. On y retrouve les variables visuelles définies par Bertin, avec parfois des notions légèrement différentes mais souvent proches. Ainsi, la variable couleur a été séparée en deux composantes différentes que sont la saturation et la teinte. Cette séparation rend plus explicite la perception de la variable visuelle valeur car la saturation possède les mêmes propriétés que la valeur sans être accompagnée d'une variation de teinte. De même la variable taille est affinée en trois catégories : longueur, surface et volume.

Mackinlay intègre également des éléments graphiques qui sont déjà des variables visuelles selon Bertin. C'est pour cette raison que nous avons préféré parler ici d'indice visuel (facteur graphique) plutôt que de variable visuelle au sens où nous l'avons décrit. Cela se traduit par l'apparition dans le tableau d'indices visuels tels que les angles, les

connexions entre éléments graphiques ou l'inclusion d'un élément dans un autre. Pour chaque type de données, les indices visuels sont classés par ordre décroissant d'expressivité. Ceux qui ne sont pas classés sont ceux qui ne peuvent être utilisés pour exprimer ce type de données.

	Quantitative	Ordonnée	Qualitative
Position	0	0	0
Longueur	1	7	8
Angle	2	8	9
Orientation	3	9	10
Surface	4	10	11
Volume	5	11	12
Valeur	6	1	5
Saturation	7	2	6
Teinte	8	3	1
Texture	N	4	2
Connexion	N	5	3
Inclusion	N	6	4
Forme	N	N	7

Tableau 2.2 – Classification des indices visuels selon l'échelle d'expressivité. Par exemple, la position est l'indice le plus expressif pour tous les types de données. Les indices visuels qui ne sont pas classifiés sont ceux qui ne peuvent pas exprimer ce type de données [29].

On retrouve enfin cette même préoccupation dans les travaux de Zhang [45]. Sa proposition rejoint la notion d'expressivité de Mackinlay. Si une donnée est représentée par une variable visuelle possédant un plus petit nombre de propriétés que la donnée, alors celle-ci est représentée de manière incomplète. Si au contraire, le nombre de propriétés de la variable visuelle est supérieur à celui des données, alors la mise en correspondance de cette variable avec la donnée introduit un facteur d'erreur qui risque d'entraîner une mauvaise perception de la donnée.

2.1.3 L'automatisation des règles de la construction graphique

L'objectif général des systèmes automatisés de visualisation est de concevoir automatiquement une représentation optimale, basée sur des caractéristiques de l'ensemble

des données et les propriétés perceptuelles des représentations graphiques. Pour accomplir cette tâche, ces systèmes doivent caractériser de façon formelle l'espace des données, l'espace de représentation, les relations entre les deux, et une échelle pour évaluer les conceptions qui résultent. Plusieurs techniques d'Intelligence Artificielle permettent d'aider à la conception en exploitant la connaissance. On distingue traditionnellement pour cela deux types d'approches : les raisonnements à base de règles et à base de contraintes. Ces deux raisonnements s'appuient sur des connaissances explicites et formalisées. Dans les sections suivantes, nous exposons les travaux qui utilisent ces approches pour la conception de systèmes automatisés de visualisation.

2.1.3.1 Les systèmes à base de règles

Mackinlay a développé un système automatisé de visualisation (*Application independent Presentation Tool - APT*) [29]. Il a développé un langage graphique pour codifier une représentation et a construit des échelles d'expressivité et d'efficacité pour évaluer des conceptions alternatives, selon le type de données à visualiser (qualitatif, ordonné, ou quantitatif). Le critère d'expressivité est relié à la capacité de transcrire une donnée dans le domaine visuel. Le critère d'efficacité est relié à la capacité du système visuel de percevoir un indice visuel donné.

'APT' utilise un système à base de règles qui s'appuie sur la nature des données, les propriétés perceptives (ou l'expressivité) des composantes de la graphique et la tâche de l'utilisateur. Il sélectionne une règle qui est capable de proposer une représentation satisfaisant les critères d'efficacité et d'expressivité. 'APT' utilise la stratégie diviser-pour-régner (*devide-and conquer*). L'algorithme se déroule en trois étapes : division, sélection et composition. Les critères d'expressivité et d'efficacité sont appliqués à chaque étape. Cette stratégie est devenue un modèle de base d'exécution générale de visualisation. La classification des données est effectuée par la division, comme réordonner les objets graphiques dans la visualisation globale par leur importance. À l'étape de sélection, le système sélectionne les attributs des données à visualiser et associe à chacun de ces attributs un indice visuel. La dernière étape consiste à construire une visualisation par composition des attributs sélectionnés.

2.1.3.2 Les systèmes à base de contraintes

Senay et Ignatinus ont développé 'VISTA', un système automatisé de génération de la visualisation. Ce système utilise des règles de constructions graphiques (appelées règles de compositions) [35] pour la génération automatique du *mapping* entre les données et les représentations graphiques. Pour exprimer graphiquement les données, ces dernières sont classifiées en deux catégories : qualitative et quantitative. Les données qualitatives se divisent en deux types : données qualitatives (nominales) et ordonnées (ordinales). Suivant leur classification, ces données seront exprimées graphiquement par les indices visuels les plus appropriés. L'ensemble de ces primitives forme le vocabulaire de la visualisation. Senay et Ignatinus ont étendu la classification de Mackinlay (*Application-independent Presentation Tool*) [29] en ajoutant cinq techniques de composition : la composition par le marquage, par la superposition, par l'union, par la transparence, et par l'intersection. La Figure 2.7 illustre des exemples de ces compositions dans l'outil 'VISTA'.

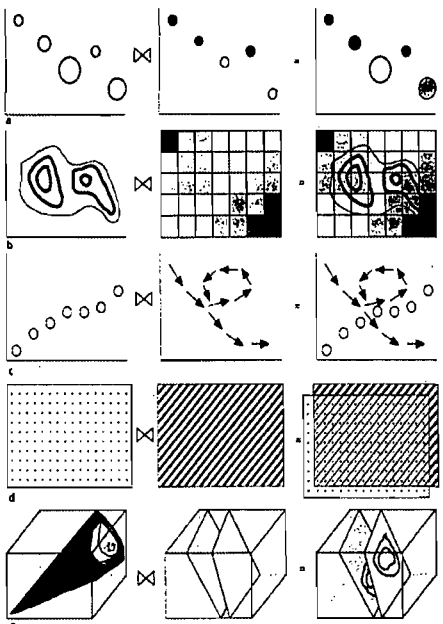

Figure 2.7 – Quelques exemples de composition de 'VISTA' [35].

Healey *et al.* [20] ont développé un assistant d'aide à la visualisation *(VIsualization Assistant- VIA)* conçu pour aider les utilisateurs à construire des visualisations perceptuellement efficaces. Ils ont proposé des directives basées sur des études de la perception humaine. Ces directives seront utilisées par des moteurs d'évaluation du *mapping* entre les attributs de données et les indices visuels, pour une visualisation donnée. Les résultats (les poids d'évaluation) seront employés par un algorithme de recherche (*mixed-initiative search*) pour identifier de nouvelles visualisations plus efficaces. 'VIA' s'intéresse principalement à chercher :

- comment construire des visualisations d'une façon indépendante de l'application ;
- comment produire les visualisations qui supportent les tâches d'exploration, la recherche, et l'analyse ;
- comment permettre aux utilisateurs de participer à ce processus pour garantir des visualisations qui respectent les préférences du contexte et de l'utilisateur.

'VIA' permet à l'utilisateur d'interagir avec le processus de recherche, en lui permettant de spécifier des conditions et des contraintes pour les visualisations. 'VIA' a été intégré à un outil de visualisation, utilisé pour deux applications différentes : la météo et le commerce électronique.

2.2 Interaction avec la visualisation

Pour être réellement efficace, la visualisation de données doit être interactive et permettre d'affiner dynamiquement l'affichage et de répondre aux requêtes graphiques de l'utilisateur. Diverses techniques ont été développées afin de faciliter la compréhension globale et la recherche d'information dans un environnement visuel. Afin de concevoir des outils de visualisation les mieux adaptés aux besoins d'utilisateurs, nous distinguons deux catégories de recherche d'information :

- la recherche d'une réponse à une requête très précise, comme par exemple la recherche d'une entité ayant une valeur très élevée. Dans ce cas, les taxonomies de tâches d'analyse sont les plus efficaces.
- la recherche dont le sujet d'intérêt n'est pas précis ; l'utilisateur a besoin de com-

prendre et d'explorer le système, afin de découvrir des informations susceptibles de l'intéresser. Ce type de recherche d'information est plus complexe, car la requête n'est pas exprimée de manière précise. Cette tâche peut être facilitée par l'utilisation d'outils qui aident l'utilisateur à sélectionner et atteindre les endroits à visiter. Les taxonomies d'interaction et de navigation dans des systèmes complexes sont plus adaptées à ce type de besoin.

L'utilisation de taxonomies peut s'avérer efficace pour positionner précisément un travail de recherche. Dans le domaine de la visualisation d'information, plusieurs taxonomies ont été proposées. Il est important de remarquer qu'elles peuvent porter sur différents objets ou combinaisons d'objets, tels que, par exemple, les types de données, les tâches à accomplir ou encore les techniques d'interaction [1, 36, 42, 46]. Dans les sections suivantes, nous allons présenter les différentes taxonomies de tâches d'analyse et techniques d'interaction.

2.2.1 Taxonomies de tâches d'analyse

Wherend et Lewis [42] ont élaboré une taxonomie d'opérateurs (appelés aussi : *tâches cognitives*) qui permettent à un utilisateur d'accomplir sa tâche d'analyse (voir Tableau 2.3). Ces opérateurs décrivent l'ensemble de requêtes ou de questions qu'un utilisateur pourrait se poser lors de l'analyse de données (quelles sont les données qui répondent à certains critères, qu'est-ce-qu'elles représentent, etc.). Puis, ils créent une matrice de représentation de sous-problèmes qui correspondent à une combinaison de certains types d'objets, tels que : scalaire, vecteur ou tâche d'utilisateur (*identify, locate, distinguish, categorize, cluster, distribute, rank, compare within entities, compare between relations, associate, correlate*). Ensuite, pour trouver la représentation graphique la plus adéquate à un problème donné, ils enrichissent cette matrice par des techniques de visualisation. Cette matrice permet de générer un *mapping* entre les techniques et les problèmes.

Zhou et Feiner [46] ont identifié un ensemble de tâches visuelles qui servent d'interface entre les buts d'utilisateurs (*intents*) et les techniques de visualisation. Ces tâches

Opérateurs	Description
Locate	l'utilisateur connaît l'entrée de l'ensemble de données et l'indique en pointant sur elle ou en la décrivant.
Identify	cette tâche est similaire à *Locate*, mais l'utilisateur décrit l'entrée de l'ensemble de données sans qu'il le connaisse au préalable.
Distinguish	différents objets doivent être présentés comme étant des entités visuelles distinctes.
Categorize	les objets peuvent être différents car il appartiennent à différentes catégories (qui doivent être décrites par l'utilisateur).
Cluster	le système peut trouver les catégories et les objets qui leur appartiennent, ils sont présentés comme étant groupés ou liés les uns aux autres.
Distribution	l'utilisateur spécifie les catégories et les objets qui leurs appartiennent, ils seront distribués selon ces catégories.
Compare	on demande à l'utilisateur à comparer entre les entités à la base de leurs attributs.
Compare within/between relations	on demande à l'utilisateur de comparer entre les entités similaires ou les différents ensembles d'objets.
Associate	on demande à l'utilisateur d'établir les relations entre les objets visualisés.
Correlate	l'utilisateur peut observer les attributs partagés entre les objets.

Tableau 2.3 – La taxonomie des opérateurs - selon Wherend et Lewis [42].

permettent aux utilisateurs d'accomplir leurs buts dans un environnement visuel (voir Tableau 2.4). Zhou et Feiner ont examiné les techniques pour construire automatiquement des présentations multimédia dans leur outil IMPROVISE. Ces constructions sont basées sur les objectifs de l'utilisateur. Les auteurs regroupent les objectifs haut niveau de la présentation en deux catégories : *inform* et *enable*. La première (*inform*) traite l'élaboration et le résumé des données, tandis que la seconde (*enable*) traite l'exploration des données et la dérivation des relations. Ensuite, ils raffinent les opérations de Weherend et Lewis [42], en des tâches visuelles organisées par leur accomplissement visuel (utilisateur bas niveau) et leurs implications visuelles (les capacités visuelles nécessaires dans la réalisation des accomplissements visuels). Chaque objectif de présentation correspond aux tâches visuelles qui l'achèvent. Par exemple, l'objectif *enable-compute-sum* correspond aux tâches : *correlate, locate* et *rank.*

Implication	Type	Sous type	Tâches élémentaires
Organization	*Visual Grouping*	*Proximity*	*associate, cluster, locate*
		Similarity	*categorize, cluster, distinguish*
		Continuity	*associate, locate, reveal*
		Closure	*cluster, locate, outline*
	Visual attention		*cluster, distinguish, emphasize, locate*
	Visual sequence		*emphasize, identify, rank*
	Visual composition		*associate, correlate, identify, reveal*
Signaling	*Structuring*		*tabulate, plot, structure, trace, map*
	Encoding		*label, symbolize, portray, quantify*
Transformation	*Modification*		*emphasize, generalize, reveal*
	Transition		*switch*

Tableau 2.4 – La taxonomie des opérateurs - selon Zhou et Feiner [46].

Amar *et al.* [1] ont défini un ensemble de tâches d'analyse servant à formuler les questions qu'un utilisateur pourrait se poser, lors de la manipulation de données (voir le Tableau 2.5).

Opérateurs	Description
Retrieve Value	Étant donné un ensemble d'entités, trouver les attributs de ces entités.
Filter	Étant données certaines conditions sur les valeurs des attributs, trouver les entités qui satisfassent ces conditions.
Compute Derived Value	Étant donné un ensemble d'entités, calculer une représentation numérique de ces données (par exemple : moyenne , médiane, etc).
Find Extremum	Trouver les entités ayant une valeur extrême d'un attribut donné.
Sort	Étant donné un ensemble d'entités, ordonner selon une métrique ordinale.
Determine Range	Étant donné un ensemble de données et un attribut quantitatif d'intérêt, caractériser la distribution de ses valeurs dans l'ensemble.
Characterize Distribution	L'utilisateur spécifie les catégories et les objets qui leurs appartiennent, ils seront distribués selon ces catégories.
Find Anomalies	Identifier n'importe quelle anomalie dans l'ensemble de cas de données en respectant la condition donnée.
Cluster	Regrouper les entités selon les valeurs de leurs attributs.
Correlate	Étant donné un ensemble d'entités et deux attributs, déterminez la relation utile entre les valeurs et ses attributs.

Tableau 2.5 – La taxonomie des opérateurs - selon Amar *et al.* [1].

2.2.2 Taxonomies de techniques d'interaction

C'est l'interaction qui rend possible l'exploitation réelle des vues d'ensembles une fois produites. En effet, l'être humain est particulièrement habile à extraire des informations d'un environnement qu'il contrôle directement et activement par rapport à un environnement qu'il ne peut qu'observer de manière passive. Selon l'approche psychologique de la perception, la perception est indissociable de l'action : il faut agir pour percevoir et il faut percevoir pour agir (c'est un couplage action-perception).

Shneiderman [36] propose une taxonomie pour caractériser comment l'utilisateur interagit avec la visualisation. Il propose sept types de tâches associées à la visualisation d'informations :

- **Vue d'ensemble et zoom**. Le zoom est une façon de concilier la vue globale et de permettre aussi aux utilisateurs d'accéder aux détails. Un zoom avant révèle ces détails tandis qu'un zoom arrière révèle le contexte. On peut distinguer deux types de zoom : le zoom infini et le zoom sémantique. Le premier est un changement d'échelle sur la représentation qui est alors considéré comme une image, ce mode est peu intéressant dans le cas de la visualisation de l'information. Le second en revanche l'est beaucoup plus car chaque niveau de zoom correspond à un niveau différent de l'hiérarchie.
- **Détails sur demande**. L'utilisateur peut sélectionner un élément ou un groupe d'éléments et interroger ces éléments sur leurs caractéristiques. La méthode la plus employée est de cliquer sur l'élément en question, les détails apparaissant dans une fenêtre.
- **Filtrer et relier des informations**. Cette tâche possède comme principale caractéristique de permettre à l'utilisateur de filtrer dynamiquement les données qui lui sont présentées sur un espace à deux dimensions. La représentation graphique réagit instantanément aux différentes actions de l'utilisateur qui filtre à l'aide de moyens de contrôles qui agissent sur les couleurs appliquées aux données, sur leur apparition ou disparition, etc. Ce système permet donc de sélectionner les données

suivant leurs différents attributs.

- **Historique**. La visualisation permet une démarche d'exploration des données. Cette démarche s'effectue nécessairement en plusieurs étapes et il est important de conserver un historique des étapes franchies.

2.3 État de l'art en visualisation de qualité de logiciel

Quelque soit le type d'information recherché lors de l'analyse d'un logiciel, deux problèmes récurrents à ce domaine ressortent fréquemment, la dimension et la granularité des données récoltées. La visualisation de programmes est une réponse possible à ces deux problèmes, aidant à l'appréhension du comportement de systèmes informatiques. Mais la visualisation seule n'a pas la prétention de résoudre le problème dans son intégralité. Dans ce domaine, l'espace des informations visualisé est fortement structuré, mais la complexité naît du fait qu'il n'existe pas une unique structure pour cet espace mais bel et bien plusieurs structures qui sont toutes significatives ou pertinentes du point de vue de l'utilisateur.

Nous avons évoqué dans les premières pages de ce document, l'importance de la représentation graphique de l'information à visualiser. Cette visualisation n'existe toujours pas de manière évidente, elle met en jeu à la fois l'aspect humain ou cognitif et l'aspect informatique de la visualisation. Cette section présente la mise en correspondance des différents travaux.

2.3.1 Représentation de données multidimensionnelles

Les données multidimensionnelles sont des données qui possèdent n attributs ($n > 3$) [36] et l'importance relative de ces attributs est sensiblement la même. Dans cette section nous allons présenter quelques travaux en visualisation des données multidimensionnelles.

2.3.1.1 *MetricView*

Termeer *et al.* ont développé un outil de visualisation de logiciels appelé '*MetricView*' [39], qui transforme les diagrammes traditionnels d'UML en vues polymétriques. Le point de départ est un diagramme de classes UML ordinaire. L'utilisateur peut alors définir une grille 'invisible' sur la surface d'une classe UML et visualiser une métrique sur chaque cellule. La représentation visuelle de la métrique est choisie par l'utilisateur. Il est possible de visualiser une métrique par une icône ou un rectangle dont la largeur et la taille peuvent varier selon la valeur de la métrique. '*MetricView*' permet de préserver l'information structurelle via les diagrammes UML et visualiser des informations supplémentaires sur les métriques. La Figure 2.8 montre un exemple de leur approche.

Figure 2.8 – '*MetricView*' [39]. (À gauche) La représentation d'une vue d'ensemble du diagramme de classes UML. (À droite) la visualisation des métriques sur une classe UML.

2.3.1.2 *CodeCrawler*

Lanza *et al.* ont développé un outil de visualisation appelé '*CodeCrawler*' [9, 27], dans le but de comprendre l'architecture d'applications informatiques. Cet outil utilise des graphes polymétriques enrichis de métriques extraites du code source du système à analyser. Ces graphes d'informations comportent des nœuds représentant des classes du système étudié, pouvant être reliés par des arêtes en fonction de relations entre ces nœuds (héritage, communication, etc.). La Figure 2.9 illustre les six indices visuels d'un nœud utilisé pour représenter six mesures numériques : la hauteur, la largeur, la couleur (niveau de gris), deux positions (x, y), et l'épaisseur de ses arrêtes qui le relient avec

d'autres nœuds. Cet outil offre la possibilité de visualiser grand nombre d'informations

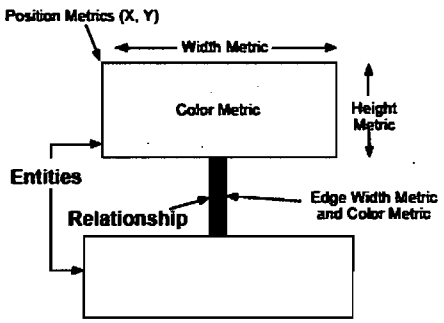

Figure 2.9 – Vue polymétrique [27].

(six dimensions) simultanément, mais en présence d'un grand nombre de classes dans le système, il devient très vite inutilisable :

- les informations pertinentes sont souvent cachées derrière cette grande masse de données, il faudra donc les filtrer pour pouvoir les visualiser. Cependant, ce système n'offre pas à l'utilisateur la possibilité d'interagir avec la visualisation.
- les règles issues des études sur la perception visuelle se sont pas prises en compte. La Figure 2.10 illustre deux exemples de ce problème. À gauche, l'utilisation simultanée de la variation de la largeur et la longueur introduit le phénomène d'interférence qui est susceptible de nuire à la perception de l'utilisateur et d'introduire des ambiguïtés. À droite, l'utilisation de la variation de l'épaisseur des arêtes n'est pas très efficace pour les gros systèmes.

2.3.1.3 *CodeCity*

Wettel et Lanza [43] utilisent une métaphore de la ville pour représenter le logiciel. Les édifices sont caractérisés par cinq indices visuels : dimensions, position, couleur, saturation, et transparence. Chaque indice visuel correspond à une métrique de l'entité logiciel, représentée par l'édifice. Une capture d'écran de leur outil est présentée à la Figure 2.11.

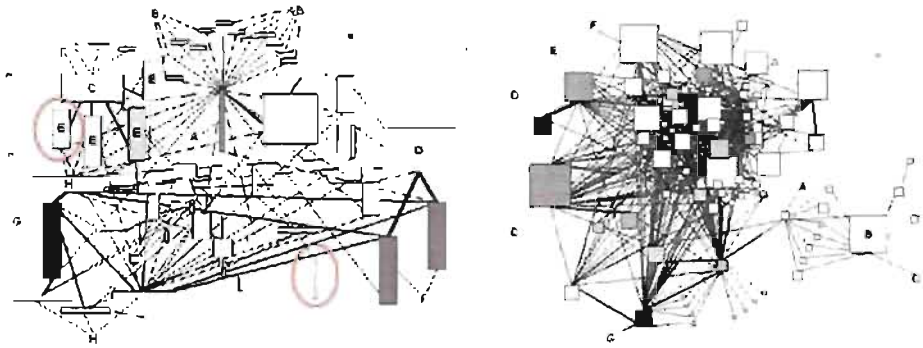

Figure 2.10 – Vue polymétrique [3]. (À gauche) Le phénomène d'interférence entre la variation de la largeur et la longueur. Les deux nœuds encerclés ont la même longueur, mais leur perception est biaisée, le nœud le moins large apparaît plus long que le second. (À droite) La variation de l'épaisseur des arêtes est non efficace pour les gros systèmes.

2.3.2 Représentation des informations de bas niveau

Dans la catégorie des visualisations affichant les informations de bas niveau sur le code ou les lignes de code elles-mêmes, on peut citer les travaux des sections suivantes.

2.3.2.1 *SeeSoft*

L'outil '*SeeSoft*' est développé par Eick *et al.* [13]. Il est utilisé pour visualiser des statistiques sur les lignes du code source d'un logiciel. Cet outil a par la suite été l'inspiration de toute une famille d'outils permettant de représenter les lignes de code à l'aide de segments de couleur. Ce système donne une vue d'ensemble du code source d'un logiciel en représentant chaque ligne de code par une barre horizontale. Ces barres sont organisées verticalement comme le sont les instructions d'un programme. Pour réduire au maximum la place nécessaire à la représentation, la hauteur de chaque barre horizontale est réduite à un seul pixel. La couleur de chaque ligne traduit une valeur statistique, comme par exemple, la date à laquelle la ligne a été ajoutée au code (sur la Figure 2.12, en rouge les lignes les plus récentes et en bleu les plus anciennes).

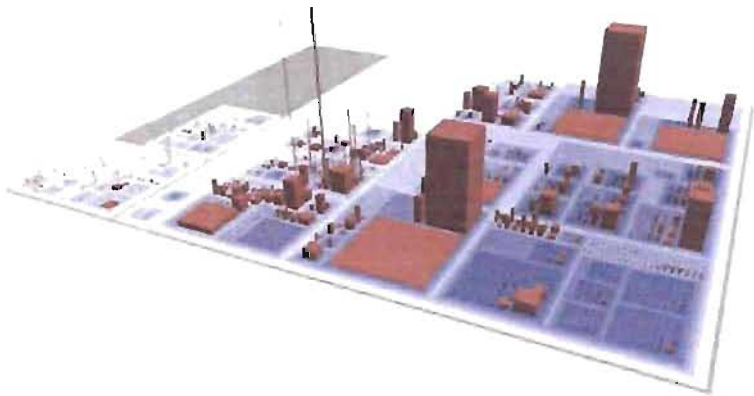

Figure 2.11 – Un exemple d'utilisation de *CodeCity* [43], pour la visualisation du système *ArgoUml*.

2.3.2.2 *Sv3D*

Dans leur outil '*Sv3D*' [30], Marcus *et al.* visualisent les lignes du code source d'un programme les unes à la suite des autres à la manière de '*SeeSoft*' [13]. Par contre, les auteurs utilisent la représentation 3D plutôt que la représentation par pixels, pour visualiser plus d'informations. Chaque ligne de code est représentée par une barre en trois dimensions sortant du plan. La longueur et la couleur des barres sont associées aux différentes informations des lignes du code, comme par exemple, des statistiques. Les auteurs utilisent le dessus et le dessous du plan pour représenter les éléments. Ceci leur permet de multiplier les caractéristiques et utiliser différents angles de vue en exploitent la transparence pour accentuer certaines parties de la représentation (voir la Figure 2.13).

2.3.2.3 *SequoiaView*

La représentation par '*treemap*' fut conçue par Shneiderman [22] pour résoudre un problème de visualisation du contenu d'un espace de stockage d'un système informatique. L'idée générale de cet algorithme est de représenter graphiquement une structure arborescente comme une succession de rectangles orientés alternativement dans la direction verticale et horizontale à l'intérieur d'une surface elle aussi rectangulaire. Il existe

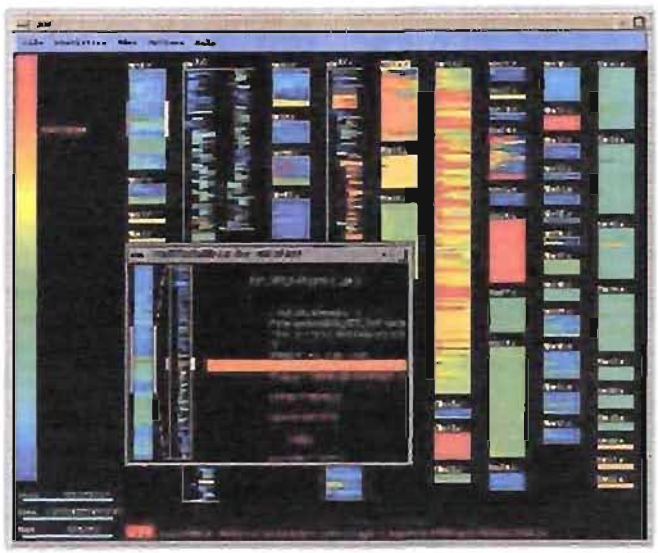

Figure 2.12 – Un exemple d'utilisation de 'SeeSoft' [13]. Les lignes du code source sont visualisées par des barres horizontales dont la couleur correspond à une valeur statistique du code.

plusieurs variantes de cet algorithme, comme par exemple, l'algorithme *Cushion Treemaps* [44] qui consiste à appliquer un éclairage particulier, pour accentuer la perception des rectangles.

Holten *et al.* ont développé une extension de l'outil 'SequoiaView' [21], pour visualiser des logiciels. Cette extension est basée sur l'algorithme *Cushion Treemaps* pour la représentation de la hiérarchie du logiciel dont les feuilles représentent les méthodes. À chaque feuille, ils associent une couleur et une texture pour visualiser deux métriques extraites du code source. Cet outil permet de visualiser les caractéristiques globales du logiciel (c'est-à-dire sa composition hiérarchique) tout en conservant certains détails (c'est-à-dire les métriques au niveau des méthodes). La Figure 2.14 illustre la visualisation d'un logiciel, dont la couleur et la texture sont utilisées pour visualiser les métriques *Fan-In* et *Fan-Out*, respectivement. La métrique *Fan-In* de la méthode M se rapporte au nombre de méthodes qui appellent M, tandis que la métrique *Fan-Out* de M se rapporte au nombre de méthodes qui sont appelées par M. Il est généralement souhaitable d'avoir une petite valeur de *Fan-Out* pour les méthodes qui ont une valeur élevée de *Fan-In*.

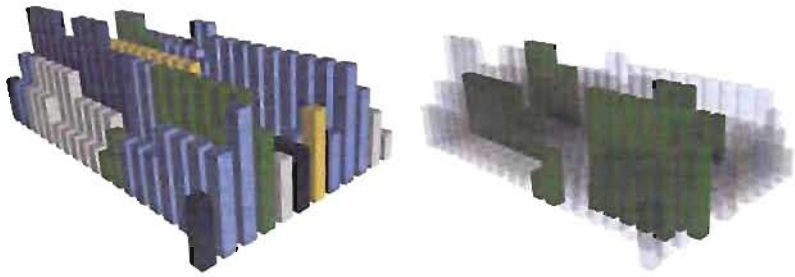

Figure 2.13 – Un exemple d'utilisation de '*Sv3D*' [30]. (À gauche) La représentation d'une vue d'ensemble du code source d'un programme. (À droite) L'utilisation de la transparence pour accentuer certaines parties de la représentation.

2.3.2.4 *Visual Code Navigator*

Lommerse *et al.* [28] pour leur part, ont développé l'outil '*Visual Code Navigator*' pour représenter l'information d'un très bas niveau, c'est-à-dire la présentation du texte lui-même. Les auteurs utilisent des rectangles pour englober les expressions et les différents niveaux de portée dans le code. Pour distinguer facilement la région occupée par un rectangle, ils appliquent sur ce dernier un éclairage particulier pour donner une illusion de trois dimensions (le rectangle semble bombé). Une amélioration de cette technique appelée *plateau cushions*, consiste à aplatir une bonne partie du rectangle pour obtenir un éclairage uniforme sur cette région, ce qui améliore la lisibilité du texte qui se trouve dans la portée sous le rectangle. Cet outil offre aussi la possibilité d'avoir une vue d'ensemble du code source. Dans ce cas les rectangles sont réduits jusqu'au point où on perçoit différents niveaux d'imbrication à l'aide de la superposition des rectangles où le texte devient illisible et sans importance.

Les auteurs utilisent le même principe de rectangles à l'intérieur du *Treemap* pour représenter tous les symboles d'un programme. Ces symboles incluent les différents *packages* ou espaces de nom jusqu'aux noms des différentes variables. Par contre, l'information contenue dans le corps des fonctions n'est pas considérée. La Figure 2.15 montre un exemple de leurs travaux.

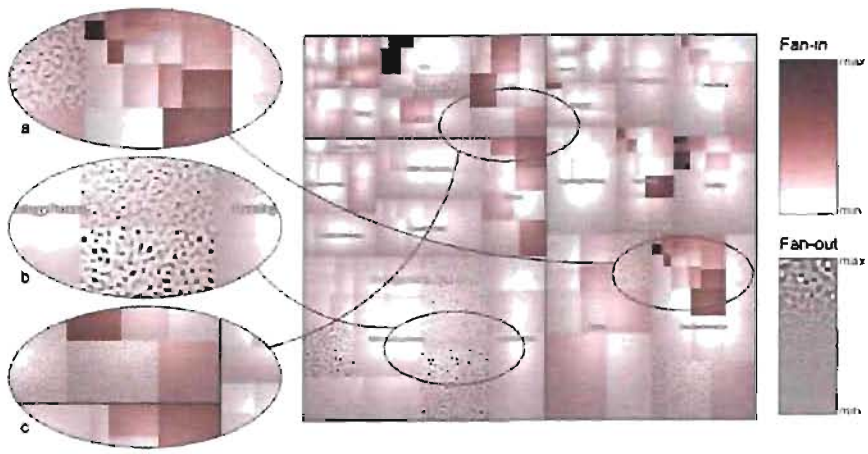

Figure 2.14 – Visualisation multivariable des métriques *Fan-In* et *Fan-Out*. (a) Méthodes ayant une valeur très élevée pour *Fan-In* et petite valeur de *Fan-Out*. (b) Méthodes ayant une petite valeur pour *Fan-In* et une valeur très élevée pour *Fan-Out*. (c) Méthodes ayant une valeur peu élevée pour *Fan-In* et une valeur moyenne pour *Fan-Out* (une anomalie) [21].

2.3.2.5 Extension de *CodeCrawler*

Ducasse *et al.* ont développé une extension de '*CodeCrawler*' [9, 27], visualisant la structure interne d'une classe par des représentations '*Class Blueprint*' [12]. Dans une représentation '*Class Blueprint*', les classes du système sont visualisées par des rectangles, chaque rectangle est subdivisé en cinq compartiments qui groupent les méthodes et les attributs : les méthodes d'initialisation, les méthodes publiques d'interface, les méthodes internes d'implémentation, les méthodes d'accès, et les attributs. La structure d'un compartiment est représentée par un graphe polymétrique enrichi de métriques extraites du code source. Dans ce cas, les graphes polymétriques comportent des nœuds représentant des méthodes et des attributs de la classe étudiée, pouvant être reliés par des arêtes en fonction de relations entre ces nœuds (invocation, utilisation, etc.). La Figure 2.16 illustre un exemple de représentation d'une classe : la hauteur et la largeur représentent deux métriques du code source, tandis que la couleur (teinte) encode une information sémantique (le type du nœud), par exemple, méthode abstraite, méthode qui retourne des valeurs constantes, etc.

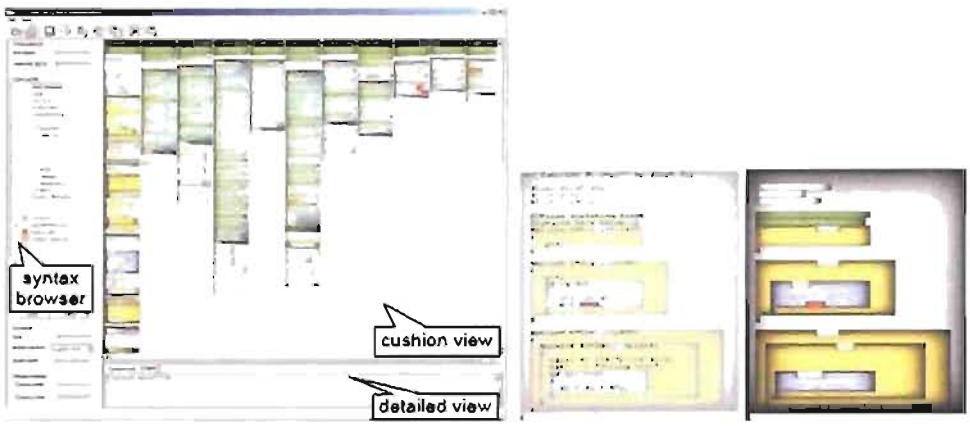

Figure 2.15 – Un exemple d'utilisation de '*Visual Code Navigator*' [28]. (À gauche) La représentation d'une vue d'ensemble du code source d'un programme. (À droite) L'application d'un zoom sur une portion du code source (le plus bas niveau de visualisation où le texte est visible). Les coussins peuvent être présentés avec différents niveaux de transparence.

2.4 Discussion

Le but de la visualisation d'informations est d'exploiter les caractéristiques du système visuel humain, pour faciliter la manipulation et l'interprétation de données informatiques variées. Nous avons pu montrer tout au long de ce chapitre l'importance de la prise en compte des aspects cognitifs lors de l'étude des aspects informatiques de la visualisation. Nous avons présenté plusieurs travaux [2, 6, 15, 18, 19, 29, 45] qui ont montré comment exploiter, de façon intuitive ou ad hoc, les caractéristiques de perception visuelle, afin de construire des visualisations efficaces. Notre étude, qui s'appuie sur ces travaux, portera spécifiquement sur l'exploitation des caractéristiques de la perception visuelle de façon plus systématique.

Dans la première partie, nous nous sommes intéressés à la fois au système perceptif visuel, aux composantes du système d'expression et à leur interprétation par l'utilisateur. L'objectif de cette démarche était d'identifier le système d'expression (les composantes des constructions graphiques) et d'en cerner les bornes. Cette première partie s'est articulé autour des points suivants :

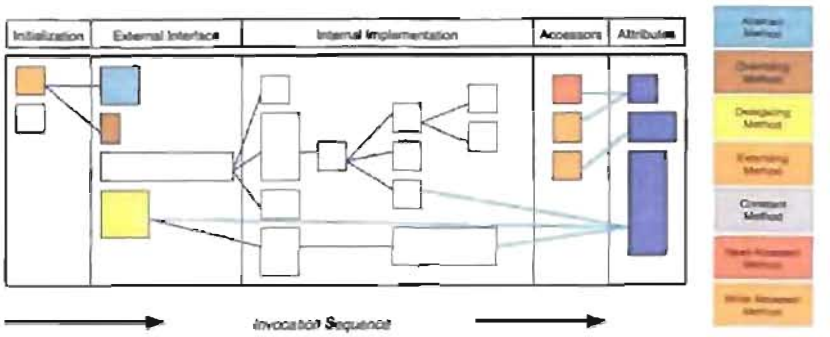

Figure 2.16 – Un exemple de visualisation de la structure d'une classe par une représentation '*Class Blueprint*' [12].

- la visualisation est un moyen d'expression qui s'appuie sur des composantes que nous nous sommes efforcés d'identifier au travers des travaux de Bertin et de Mackinlay, mais également ceux plus récents sur la perception préattentive.
- ces composantes sont utilisées pour transcrire les relations qui s'établissent entre les valeurs des données. Nous avons donc dans un premier temps caractérisé ces relations, avant de nous intéresser aux composantes (variables visuelles) proprement dites.
- enfin, nous avons caractérisé les variables visuelles selon des critères similaires à ceux utilisés pour décrire les relations qui existent entre les données. Cette caractérisation permet de définir des règles d'usage des variables visuelles exploitables pour la transcription de l'information.

Nous avons présenté par la suite (Section 2.1.3) quelques systèmes automatisés de visualisation ('APT' [29], 'VISTA' [35] et 'VIA' [20]), qui utilisent les règles sur la perception visuelle comme un guide pour construire des visualisations efficaces. Ces systèmes utilisent différentes techniques d'Intelligence Artificielle (raisonnement à base de règles, raisonnement à base de contraintes), qui permettent de faire de l'assistance à la conception en exploitant de la connaissance d'un expert du domaine concerné (perception visuelle et sciences cognitives). Cependant, ces outils n'offrent aucun guide du point de vue des tâches d'analyse et interactions.

Comme nous l'avons déjà évoqué au début de ce chapitre, l'utilisation de taxonomies peut s'avérer interessante pour positionner précisément un travail de recherche. En seconde partie, nous avons présenté plusieurs travaux [1, 36, 42, 46] sur les taxonomies de tâches d'analyse et de techniques d'interaction. La prise en compte de ces taxonomies lors de la conception d'un outil de visualisation s'avère souvent efficace pour la construction d'un outil de visualisation qui répond aux besoins d'utilisateur en termes de tâches d'analyse et techniques d'interaction. La taxonomie des techniques d'interaction proposée par Shneiderman [36] met de l'ordre dans la nébuleuse des solutions graphiques produites à ce jour. Mais, si elle est une bonne manière de présenter l'existant, il est néanmoins difficile de l'utiliser comme un guide dans la conception de représentations graphiques. Cette difficulté naît d'une part du fait que la taxonomie ne propose qu'une présentation de l'existant (les différentes techniques d'interaction) et d'autre part du fait que le type de la tâche d'analyse n'est pas prise en compte. La taxonomie de tâches d'analyse proposée par Amar *et al.* [1] constitue une base intéressante pour la modélisation de tâches d'utilisateurs. À l'évidence, cet examen bibliographique approfondi nous a permis de constater que les travaux effectués par les différentes équipes sont très intéressants :

- nous suggérons d'étendre la taxonomie de tâches d'analyse qui a été proposée par Amar *et al.* [1], pour construire un langage de description d'une tâche d'analyse (instance du modèle d'Analyse).
- nous proposons étendre la taxonomie d'interactions qui a été proposée par Shneiderman [36], pour définir les interactions (instance du modèle d'interaction) disponibles dans environnement visuel.
- nous allons utiliser le raisonnement à base de contraintes, pour automatiser les règles de constructions graphiques, et transformer une tâche d'analyse en une suite d'interactions, pour aider l'utilisateur à accomplir sa tâche.

En troisième partie, nous avons présenté un survol sur les outils de visualisation de logiciels existants. Cette synthèse traduit les difficultés rencontrées par ces systèmes : la dimension et la granularité des données visualisées. La première catégorie des ou-

tils de visualisation [9, 27, 39] utilise plusieurs indices visuels (couleurs, textures, etc.) pour représenter plusieurs dimensions à la fois. La multiplication de ces indices visuels est susceptible de nuire à la perception de l'utilisateur et d'introduire des ambiguïtés. En revanche, les outils de visualisation visant à enrichir l'espace d'information doivent aujourd'hui prendre en compte les règles sur la perception visuelle, pour créer des visualisations plus efficaces. La deuxième catégorie des outils de visualisation [13, 21, 28, 30] n'opère qu'à un niveau d'abstraction beaucoup trop faible (code source du logiciel). Ces outils indiquent des informations ou des statistiques sur les lignes de code sans pour autant donner d'indication sur les caractéristiques des entités du logiciel, ni sur les relations entre ces entités. Au niveau du code source la quantité d'informations peut être très importante. Des milliers de lignes peuvent être visualisées différemment, la vision proposée par ces outils est bien trop détaillée pour être exploitable. En réalité, ils ne permettent de visualiser de manière plus globale que le niveau de répertoires pouvant contenir une hiérarchie de plusieurs milliers de fichiers. Ces outils sont utiles pour analyser un fichier et non pas plusieurs fichiers (logiciel).

Dans notre étude, nous proposons tout d'abord de représenter une vue d'ensemble du logiciel qui reflète les caractéristiques principales du logiciel, c'est-à-dire, son architecture (hiérarchie des entités logicielles, et les relations entre les différentes entités logicielles) pour faciliter sa compréhension globale. La vue d'ensemble permet à l'utilisateur d'identifier les principaux centres d'intérêt du système, afin qu'il décide par où commencer sa navigation. Au cours de celle-ci, il doit pouvoir explorer certaines parties du système et obtenir des informations détaillées. Ces différentes échelles nous permettent d'éviter les changements brusques de contexte et de construire des représentations graphiques plus riches en indices visuels, sans nuire à la perception de l'utilisateur.

CHAPITRE 3

OUTIL DE VISUALISATION VERSO

Dans ce chapitre, nous présentons l'outil de visualisation VERSO qui constitue la base de notre travail. La Section 3.1 explique la représentation graphique d'une classe. La Section 3.2 discute des algorithmes de placement des classes, à la fois pour représenter le plus grand nombre possible d'entités et aussi obtenir de l'information plus globale sur l'architecture du logiciel. La Section 3.3 présente les interactions offertes par VERSO. La Section 3.4 présente l'utilisation de VERSO pour l'analyse qualitative des logiciels orientés objet. Finalement, la Section 3.5 discute des limites de VERSO, qui ont motivé le travail présenté dans ce mémoire.

3.1 Visualisation de classes

Le niveau de granularité visualisé pour l'évaluation de la qualité des logiciels est la classe. La classe est une entité logicielle qui ne possède pas une transcription visuelle directe [24]. VERSO utilise une abstraction des classes qui consiste en une description basée sur une liste de ses métriques.

Les métriques du logiciel (*software metrics*) sont des mesures prises sur le code permettant de quantifier certains aspects [7, 14] de la qualité d'une entité du logiciel, tels que sa taille/complexité (*WMC - Weighted Method per Class*), sa cohésion (*LCOM5 - Lack of COhesion in Method*), son couplage (*CBO - Coupling Between Object*), sa profondeur dans l'arbre d'héritage (*DIT - Depth in Inheritance Tree*), etc. Ces mesures peuvent servir d'indicateur de la qualité des programmes.

Dans le cadre de la visualisation de logiciels orientés objet, il est possible de distinguer deux groupes d'éléments : les classes et les interfaces. Il est intéressant de pouvoir les différencier durant la visualisation d'un logiciel pour mieux comprendre les interactions des différentes parties du logiciel. Les classes sont représentées graphiquement par des boîtes en trois dimensions. Ces boîtes possèdent au moins trois caractéristiques

qui n'interfèrent pas entre elles, soit la couleur, la hauteur et l'orientation. L'association des classes aux représentations graphiques (indices visuels) consiste à lier les métriques aux caractéristiques graphiques que nous venons de décrire. Les interfaces pour leur part sont représentées par des cylindres, ce qui permet de les différencier des classes. La Figure 3.1 présente l'apparence graphique des classes.

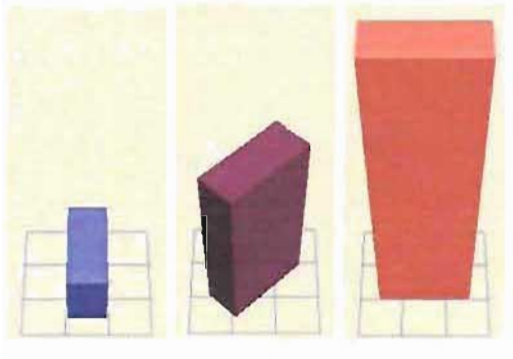

Figure 3.1 – Trois représentations de classes : les trois métriques (CBO, LCOM5, et WMC) associées respectivement aux attributs graphiques (couleur, orientation et hauteur) augmentent de gauche à droite [26].

3.2 Visualisation de systèmes

VERSO utilise deux techniques de placement : le *Treemap* [22] et le *Sunburst* [38]. Les deux algorithmes servent à représenter des hiérarchies de fichiers présents sur des disques. C'est pourquoi ces deux techniques sont toute indiquées pour représenter une hiérarchisation architecturale où des paquetages (*packages*) peuvent incorporer des classes ou d'autres paquetages.

L'idée générale de l'algorithme de *Treemap* est de représenter un arbre en utilisant le mieux possible l'espace disponible (le rectangle de l'écran). Le grand rectangle sert à identifier la racine de l'arbre. Ce rectangle est subdivisé verticalement en plusieurs rectangles plus petits où chaque portion représente un paquetage enfant. La taille accordée à chaque portion correspond à la taille de l'enfant. Ensuite le processus reprend de façon récursive avec les paquetages enfants, mais d'un niveau à l'autre, on alterne la direction

des subdivisions entre verticale et horizontale. Un exemple de notre représentation de système est donné à la Figure 3.2.

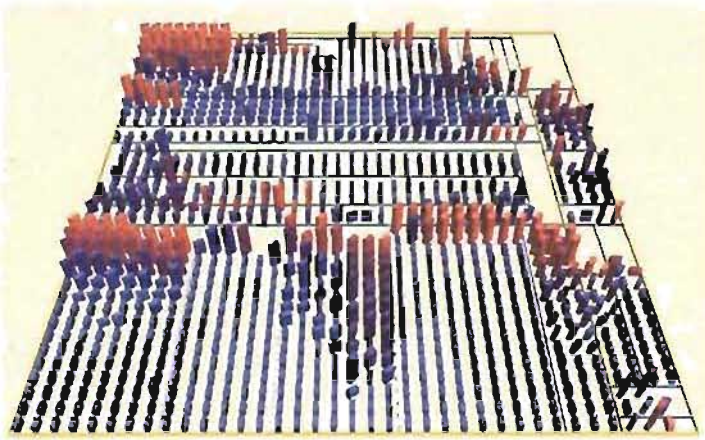

Figure 3.2 – Un exemple du placement à l'aide de l'algorithme du *Treemap*. L'image représente *PCGEN* une application servant pour les jeux de rôle (1129 classes). La couleur est ici associée au couplage, la hauteur est associée à la complexité de la classe et la cohésion est associée à l'orientation par rapport à l'axe des Y [26].

3.3 Interactions dans VERSO

L'objet de la visualisation n'est pas simplement limité à la production de représentations visuelles *statiques*. Ces représentations doivent également pouvoir être manipulées par l'utilisateur afin de comprendre l'espace des informations ou d'interagir avec lui. VERSO offre dans ce cadre de nombreuses fonctionnalités :

1. **Vue d'ensemble et zoom** : VERSO offre une navigation fluide. L'utilisateur peut se déplacer avec la souris pour étudier des éléments cachés par d'autres ou encore se rapprocher d'un endroit qu'il aimerait explorer d'avantage. La caméra reste toujours dirigée vers le plan et elle peut tourner autour d'une demi-sphère, se déplacer ou encore se rapprocher du plan.
2. **Détails sur la demande** : l'utilisateur peut, en sélectionnant une classe donnée, faire apparaître les détails (nom de la classe et la liste de ses métriques) dans

une fenêtre. Cette possibilité donnée à l'utilisateur est très importante car elle lui permet d'avoir les valeurs exactes des métriques. L'utilisateur peut aussi afficher le code source de la classe sélectionnée, pour vérifier d'autres propriétés.

3. **Filtres** : Les filtres permettent de mettre l'emphase sur un sous-ensemble de classes vérifiant une propriété donnée. VERSO offre deux types de filtres (Figure 3.3) : le filtre statistique et le filtre d'association. Le filtre statistique affiche la distribution des valeurs métriques en utilisant des techniques statistiques classiques telles que la représentation par boîte à moustaches (*box plot*). Ceci permet en particulier de faire ressortir les classes qui présentent des valeurs anormalement basses ou élevées de façon relative pour la métrique considérée. Le filtre d'association sert à représenter les liens structuraux entre les classes. En cliquant sur une classe, on peut demander à voir par exemple ses enfants ou les classes qu'elle utilise. Dans ce cas, les classes qui vérifient la condition du filtre restent affichées identiques alors que les autres sont estompées en gris. Ceci permet de représenter les liens sans les dessiner et sans changement du contexte d'exploration.

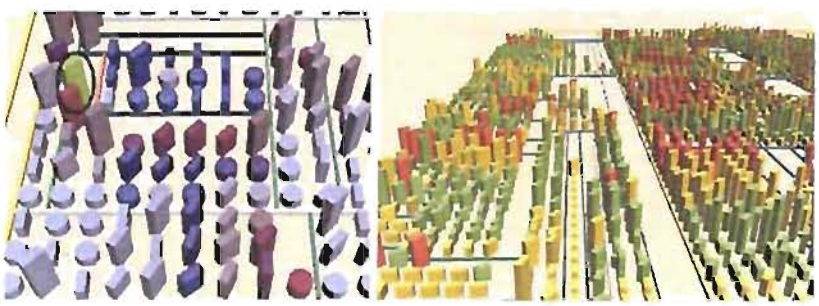

Figure 3.3 – Les filtres en VERSO. (Gauche) le filtre d'association est appliqué sur la classe encerclée, les classes qui ont gardé leur couleur sont en relation avec la classe en question. (Droite) le filtre *BoxPlot* est appliqué sur le système, les classes ayant une valeur extrême sont colorées en rouge [26].

3.4 Quelques applications de VERSO

L'outil VERSO peut être utilisé par au moins deux grands groupes d'usagers : les développeurs et les chercheurs. Les développeurs peuvent l'utiliser pour évaluer et main-

tenir leur code. Ils peuvent aussi analyser l'évolution d'un logiciel en visualisant ses différentes versions par le biais de divers types d'animations. Les chercheurs pour leur part peuvent s'intéresser à cet outil pour comprendre certains phénomènes du génie logiciel.

3.4.1 La détection des anomalies de conception logicielle

Les anomalies de conception détectées par VERSO prennent la forme soit d'une classe unique ayant certaines propriétés non-désirées, soit d'une micro-architecture reliant un groupe de classes. Dans ce dernier cas, une des classes joue le rôle principal, tandis que les autres jouent des rôles de support secondaires [10]. La Figure 3.4 illustre un exemple de détection de l'anomalie *classe mal placée*. Il s'agit d'une classe qui utilise et qui est utilisée par davantage de classes provenant d'autres paquetages que de son propre paquetage. Si ces classes associées se trouvent principalement dans un même paquetage, nous pourrions transférer la classe mal placée vers ce paquetage [31].

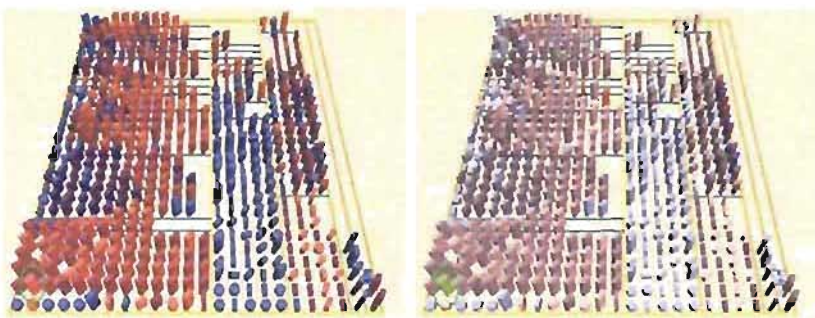

Figure 3.4 – Un exemple de *classe mal placée* découverte dans *Art of Illusion*. (Gauche) *Mapping* initial pour détecter la classe de rôle principal. (Droite) Filtre d'association sur la classe encerclée.

3.4.2 L'analyse de l'évolution logicielle

VERSO permet aussi aux utilisateurs d'étudier l'évolution du logiciel à travers la visualisation. Pour ce faire, il utilise l'animation et le contrôle du temps par l'analyste pour

représenter visuellement l'évolution. La cohérence entre les différentes versions d'un logiciel est représentée par la cohérence entre les différentes images de l'animation [25]. La Figure 3.5 illustre un exemple sur l'animation du placement.

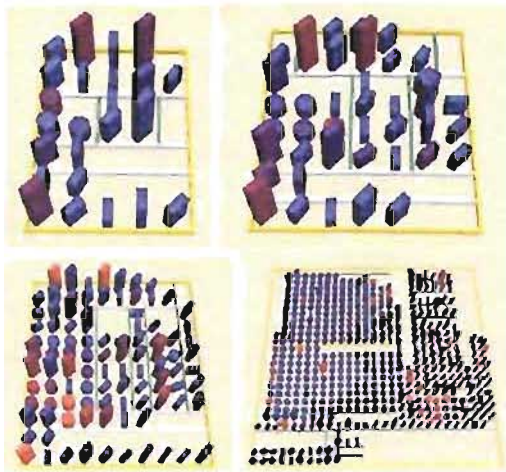

Figure 3.5 – Représentation de l'évolution avec animation du placement. Cette représentation comporte quatre versions spécifiques du logiciel *Freemind*, un outil servant à organiser les idées. Ces images ne sont jamais visibles en même temps et l'analyste navigue dans le temps à l'aide du clavier pour les visualiser. Ici, les classes ne gardent pas nécessairement la même position d'une image à l'autre pour économiser de l'espace. Les déplacements et les modifications des caractéristiques sont animés en deux phases [25].

3.5 Discussion des limites de VERSO

Dans la section précédente (Section 3.4), nous avons présenté quelques applications de la visualisation dans le domaine du génie logiciel, et en particulier pour l'analyse de la qualité de logiciels. Cependant, nous pouvons constater que quelque soit le type d'information recherché lors de l'analyse d'un système informatique orientés objet, trois problèmes récurrents à ce domaine ressortent fréquemment, soient la quantité, les dimensions et la granularité des données récoltées.

- **Quantité de données**. La plupart des logiciels orientés objets contiennent plusieurs centaines d'entités logicielles (classes). Pour remédier au problème d'explosion de la quantité de données, d'une part, VERSO utilise des algorithmes de

placements tels que *Treemap* et *Sunburst*, qui permettent de représenter un arbre en utilisant le mieux possible l'espace disponible. D'autre part, VERSO offre plusieurs interactions qui permettent de manipuler de telles quantités, telles que le zoom, le détail sur demande, les filtres, etc.

- **Dimensions de données**. Chaque classe possède plusieurs métriques de qualité, dont la représentation est destinée à faciliter la réalisation de tâches comme la recherche d'une anomalie de conception d'un logiciel. Certaines tâches d'analyse manipulent plusieurs métriques de logiciels. Or, VERSO ne permet de visualiser que trois métriques simultanément en utilisant la taille, l'orientation et la couleur des boîtes 3D qui représentent des classes. Lorsque l'utilisateur souhaite avoir plus d'information sur les classes visualisées, il effectue un changement de paramètres de la visualisation. Certains de ces changements ont pour effet de rendre visibles ou invisibles certaines métriques. Souvent, ces changements brusques de visibilité rendent l'exploration plus difficile, et ils exigent de l'utilisateur de construire une nouvelle représentation mentale cohérente de la scène visualisée, ce qui augmente l'effort cognitif nécessaire pour accomplir les tâches de l'utilisateur. La Figure 3.6 illustre un exemple de changement de contexte.
- **Granularité de données**. En étudiant VERSO, on remarque qu'il se concentre sur le niveau de granularité des *classes*, ce qui nous permet d'accéder à une compréhension de haut niveau. Cependant, une telle information n'est parfois pas suffisante. En effet, certaines tâches d'analyse nécessitent la manipulation des informations de bas niveau (du code source) du logiciel, comme par exemple, la complexité des méthodes.

Bien que VERSO a démontré son efficacité pour la visualisation de logiciels de grande taille, cet outil souffre encore de certaines limitations liées à la visualisation de données multidimensionnelles et le niveau de granularité visualisé. En plus des améliorations, nous proposons aussi d'aider les utilisateurs durant leurs tâches d'analyse. Nous proposons ce qui suit :

- utiliser des textures pour visualiser un grand nombre d'informations simultané-

ment (i.e. sans intervention de la part de l'utilisateur).

- raffiner le niveau de détails de façon plus graduée, en permettant d'amplifier dynamiquement les détails autour d'un centre d'intérêt tout en gardant le contexte global de la vue.
- automatiser le *mapping* entre les métriques et les indices visuels en utilisant les règles de la perception visuelle et des constructions graphiques, présentées dans le chapitre précédent.
- fournir aux utilisateur un langage de description de tâches d'analyse, puis, la déléguer à un système d'aide pour la transformation d'une telle description en séquence d'interactions offertes par l'outil.

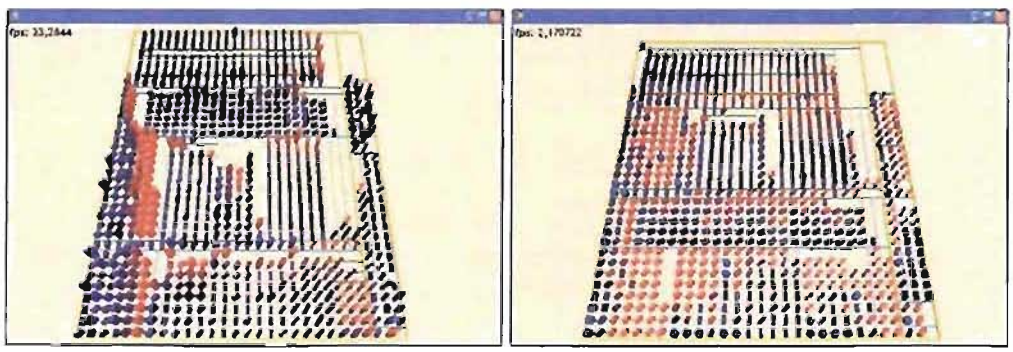

Figure 3.6 – Un exemple de changement brusque de contexte. Les deux images représentent le même système *PCGEN*. (Gauche) Les associations sont (CBO et Couleur, LCOM5 et Orientation, WMC et Hauteur). (Droite) Le changement de paramètres (DIT et Hauteur , WMC et Couleur) a pour effet un changement brusque de contexte.

CHAPITRE 4

UTILISATION DES TEXTURE POUR LA VISUALISATION MULTIDIMENSIONNELLE

Les données multidimensionnelles sont des données qui possèdent n attributs (dimensions). On y retrouve en particulier une entité logicielle qui possède plusieurs métriques de qualité, dont la représentation est destinée à faciliter la réalisation de tâches comme la recherche d'une anomalie de conception dans un logiciel. Parmi les techniques de visualisation de données multidimensionnelles, on trouve un certain nombre de solutions qui visent à enrichir les représentations graphiques avec des indices visuels représentant certaines dimensions des données. En revanche, la multiplication des indices visuels associés à chaque représentation graphique est susceptible de nuire à la perception de l'utilisateur.

Par ailleurs, les utilisateurs gagnent à percevoir des collections entières de données dans leur ensemble. Des études en perception visuelle ont en effet montré que l'être humain appréhendait mieux les informations en allant du général au particulier ; il a une perception d'abord globale (*gestalt perception*) d'une scène avant de porter son attention aux détails [32].

En suivant ces préceptes, il faut tout d'abord fournir une vue d'ensemble du système complexe, afin de permettre d'en déduire rapidement les caractéristiques principales.Tous les détails ne doivent pas être représentés à ce niveau de zoom. La vue d'ensemble doit permettre à l'utilisateur d'identifier les principaux centres d'intérêt du système, afin qu'il décide par où commencer sa navigation. Au cours de celle-ci, il doit pouvoir explorer certaines parties du système et obtenir des informations détaillées. Les données contenues dans le système peuvent posséder de nombreuses dimensions. Tous ces paramètres doivent apparaître dans la visualisation uniquement si l'utilisateur le souhaite.

La notion du niveau de détails permet une approche plus graduée, en permettant d'amplifier dynamiquement les détails autour d'un centre d'intérêt tout en gardant le

contexte global de la vue. Le zoom est une façon de concilier la vue globale et de permettre aussi aux utilisateurs d'accéder aux détails, tels que des textures. Un zoom avant révèle ces détails tandis qu'un zoom arrière révèle le contexte. On peut distinguer deux types de zooms : le *zoom infini* et le *zoom sémantique*. Le premier correspond à un changement d'échelle sur la représentation, tandis que pour le second chaque niveau de zoom correspond à un niveau différent de la granularité. Les représentations graphiques soumises à un zoom sémantique peuvent générer des présentations différentes en fonction du niveau de zoom courant.

Pour être capable de définir un nouveau mode de visualisation, il faut évidemment s'intéresser à ce que l'observateur est en mesure de percevoir comme information pertinente dans une représentation graphique. Le pouvoir expressif des textures est important. Ceci étant justifié par le fait que le système visuel humain se caractérise par une capacité à percevoir différents niveaux d'informations dans une même image. La Figure 4.1 illustre ce phénomène. L'image présentée reflète deux niveaux de lecture simultanés : un niveau élémentaire dans lequel sont identifiés les composants de l'image (la lettre E), et un niveau d'ensemble qui traduit l'organisation de ces composants (la lettre H).

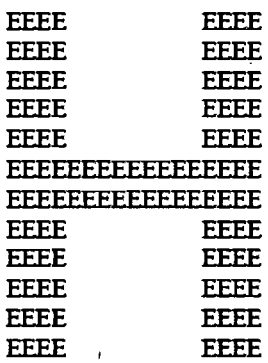

Figure 4.1 – Le système de perception visuel est capable de détecter deux niveaux d'informations simultanés : un niveau élémentaire (la lettre E), et un niveau d'ensemble (la lettre H).

Pour résumer le cahier des charges de la représentation multidimensionnelle, nous devons tout d'abord représenter une vue d'ensemble du logiciel pour faciliter sa com-

préhension globale. Cet aperçu doit refléter les caractéristiques principales du logiciel. Cependant, les utilisateurs doivent pouvoir en étudier n'importe quelle partie de manière détaillée afin de voir toutes les dimensions qu'ils désirent. Ces différentes échelles nous permettent d'éviter les changements brusques de contexte et de construire des représentations graphiques plus riches en indices visuels, sans nuire à la perception de l'utilisateur.

Dans ce chapitre, nous détaillons la manière avec laquelle nous avons étendu VERSO pour représenter les données multidimensionnelles. Ce chapitre se divise en deux grandes parties :

- Section 4.1 présente comment utiliser les textures pour affiner dynamiquement l'affichage des détails de haut niveau (niveau des classes). Ces détails seront révélés lorsque l'utilisateur applique une vue d'ensemble (zoom infini).
- Section 4.2 présente comment raffiner le niveau de granularité (niveau des méthodes), la représentation graphique d'une classe soumise à un zoom sémantique, devient transparente, ce qui nous permet de visualiser sa structure interne (c'est-à-dire la représentation de ses méthodes).

4.1 Introduction des textures

Une texture est une région dans une image numérique qui a des caractéristiques homogènes. Une définition générale peut caractériser une texture comme un ensemble de motifs arrangés selon des règles particulières de placement. Un motif est un ensemble connexe plus ou moins important de pixels de couleurs à peu près semblables. L'importance des textures pour la perception a été initialement remarquée par Gibson [16], mais le travail de Julesz [23] a été l'un des premiers à essayer de donner un sens à la notion de texture. En graphisme 3D, les textures sont des éléments essentiels pour enrichir une image de synthèse.

Dans le contexte de la visualisation multidimensionnelle, nous proposons d'utiliser les textures qui sont basées sur une carte d'information. Ce type de texture est le plus fréquent en informatique graphique ; une carte d'information (discrétisation en éléments

uniformes d'une information quelconque) qui peut être l'orientation des normales à la surface ou la micro-géométrie d'un objet. Bien souvent cette carte d'information se résume à une image couleur utilisée pour modifier les données concernant un polygone ou un objet en utilisant une paramétrisation donnée sur sa surface. Un exemple courant de l'utilisation de textures basées sur une carte d'information est l'utilisation d'une image pour modifier les couleurs d'un objet. Les grands avantages des textures basées sur une carte d'information sont leur simplicité d'utilisation et le fait que cette méthode est implantée en matériel graphique, ce qui rend leur utilisation très rapide.

4.1.1 Texture de roches

La texture des roches est définie comme une variation du nombre des motifs (dans notre cas, des roches) sur une surface (voir Figure 4.2). La variation de ce type de texture se traduit par une variation de la quantité et de la taille des roches. Si celles-ci deviennent trop grandes et/ou trop peu nombreuses, cela signifie que la valeur de la métrique visualisée est grande.

Cette variation peut être considérée comme une variation du grain, qui commence par le grain nul, dans lequel les motifs sont si nombreux, mais en même temps si petits qu'ils ne sont plus identifiables à l'œil nu. Du grain nul, la variation s'étend jusqu'aux grains importants (ou grossiers) qui forment la limite au-delà de laquelle se crée une ambiguïté sur l'implantation du motif (dans un grain trop grossier, la notion du motif risque de disparaître) [2]. L'application de cette texture a pour effet une variation de valeur de la surface, les quantités de noir et de blanc de la surface changent en fonction de la variation de grain, ce qui produit le phénomène d'interférence entre la texture et la couleur de la surface.

La longueur de la variation de grain est donc limitée par l'implantation choisie et par la surface sur laquelle le grain est appliqué. En implantation surface et en perception **ordonnée**, la longueur de la variation est importante. Elle est réduite à 4 ou 5 valeurs en perception **sélective**.

Pour définir les propriétés visuelles de cette texture, nous devons la classifier suivant l'échelle d'organisation de Bertin [2] :

- Le grain est **sélectif** ; il nous permet d'identifier ou d'isoler visuellement toutes les représentations graphiques (des entités du logiciel) d'une même catégorie. La perception sélective est utilisée lorsque nous cherchons à répondre à la question "localiser une telle catégorie". L'oeil doit alors isoler toutes les représentations graphiques de cette catégorie, abstraire toutes les autres classes et percevoir l'image formée par la catégorie cherchée.
- Le grain est également **associatif** ; il nous permet d'égaliser une variation, de regrouper toutes les classes qui possèdent des valeurs semblables du grain (qui représente une certaine métrique).
- Le grain est **ordonné** ; il est utilisé lorsque l'on a à comparer deux ou plusieurs catégories. Le tri visuel des différentes catégories des classes selon la valeur du grain est facile et ne nécessite aucun recours à la légende.

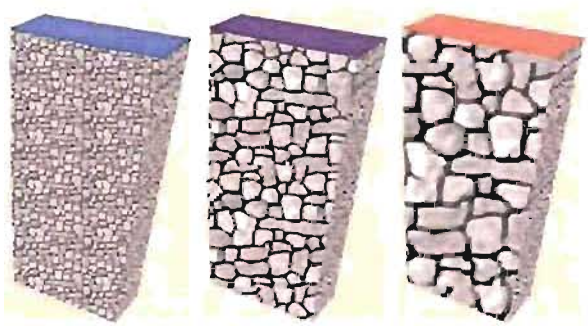

Figure 4.2 – La variation de grain en texture de roches. (Gauche) La valeur du grain nul. (Milieu) La valeur du grain moyen. (Droite) La valeur du grain maximal. La variation du grain sur la face du côté droit ne semble pas correspondante ; car sa perception est sensible au changement du point de vue de l'utilisateur.

4.1.2 Texture de briques

Cette texture consiste en un ensemble de motifs de briques arrangés selon des règles particulières de placement qui peut être horizontale, diagonale et verticale.

La forme peut être géométrique ou figurative et sa variation est infinie. La variation de forme est **associative** mais elle n'est ni ordonnée ni quantitative. Elle n'est pas non

plus sélective selon Bertin [2]. On ne peut regrouper d'un seul coup d'oeil tous les signes d'une même forme car la lecture de la forme nécessite de ne regarder qu'un signe à la fois. L'étude d'un ensemble de signes ne peut donc se faire qu'en les étudiant tous successivement.

La variation de l'orientation des briques est une variation de forme figurative. La longueur de variation de l'orientation des briques est limitée à trois positions possibles : horizontale, diagonale et verticale. Cette texture est plus adaptée aux données de type qualitatif (nominales), qui peuvent avoir trois valeurs au maximum (voir Figure 4.3).

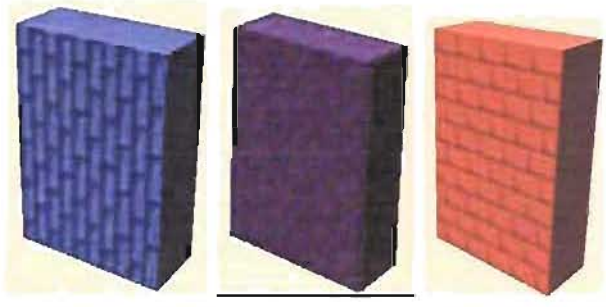

Figure 4.3 – La variation d'orientation en texture de briques.

4.1.3 Texture des fenêtres

Ce type de texture consiste à mémoriser le motif d'une fenêtre une seule fois, puis l'appliquer répétitivement sur la structure de la boîte 3D (voir la Figure 4.5). La variation de ce type de texture se traduit par une variation du nombre des fenêtres. Si celles-ci deviennent trop nombreuses, cela signifie que la valeur de la métrique visualisée est grande.

Cette texture a les mêmes propriétés visuelles que celle des roches (voir Section 4.1.1). Cependant, elle nous permet d'avoir aussi une perception **quantitative** ; car elle nous permet de définir le rapport numérique entre deux classes différentes. Ce rapport est immédiat et ne nécessite aucun recours à la légende (par exemple : la valeur de métrique de la classe *A* est le double de la classe *B*). Elle est plus adaptée aux métriques discrètes et sa longueur de variation est limitée à 20 valeurs (voir Figure 4.4).

Figure 4.4 – La variation de grain en texture de fenêtres. Le nombre maximal de fenêtres par ligne est limité à cinq.

4.1.4 Texture de l'horloge

La variation d'orientation est perceptible du fait de la différence d'angle qui existe entre plusieurs ensembles de primitives visuelles. Pour que la différence d'angle soit significative, les primitives visuelles doivent présenter une forme allongée et il est préférable de se limiter à quatre orientations (0, 30, 60 et 90 degrés). Cependant, la flèche demeure la formule la plus efficace et souvent la seule pour indiquer avec précision l'angle d'orientation. C'est pourquoi nous proposons d'utiliser une horloge comme métaphore. La variation de cette texture se traduit par la variation de l'orientation de l'aiguille de l'horloge (voir la Figure 4.7). L'utilisation de cette métaphore rattache les images observées à l'expérience du monde réel et exploite ainsi la base de connaissances de l'utilisateur (la perception des différentes orientations de l'aiguille est immédiate et ne nécessite aucun effort de l'utilisateur). La variation de cette texture est de longueur de 11 (Figure 4.6).

Pour définir les propriétés visuelles de cette texture, nous devons la classifier suivant l'échelle d'organisation de Bertin [2] :

- la variation d'orientation est **ordonnée** et **quantitative**, ce qui signifie qu'il est possible d'ordonner les éléments graphiques selon l'orientation de l'aiguille de l'horloge, mais qu'il est également possible de les comparer.
- la perception de la variation d'orientation de l'aiguille est **sélective**. Elle permet la

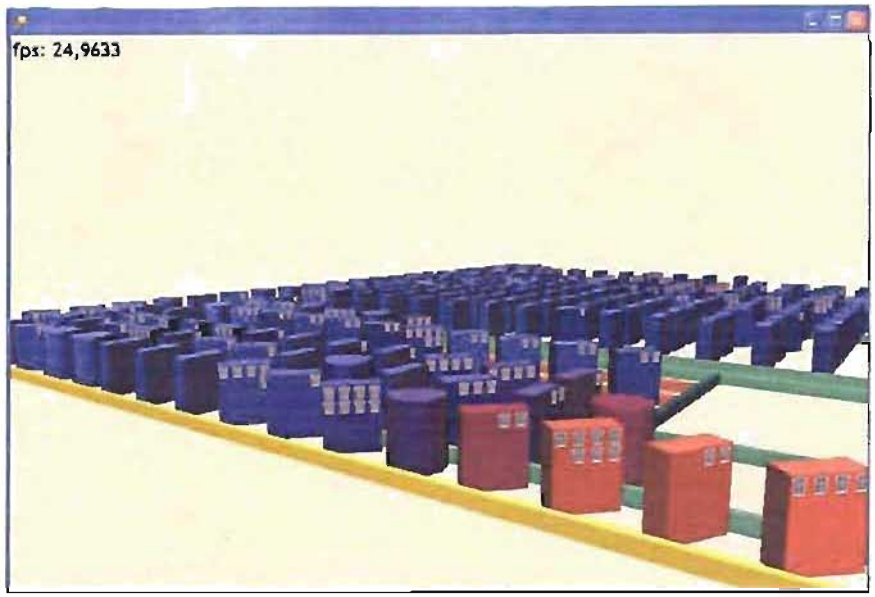

Figure 4.5 – Un exemple de visualisation du système *PMD*. La texture de fenêtres est associée à la métrique NAD.

sélection visuelle des entités graphiques de même orientation.

4.2 Visualisation des informations de bas-niveau

Le zoom sémantique est une façon de concilier la vue globale du logiciel et de permettre aux utilisateurs d'accéder aussi aux détails du plus bas niveau (méthodes). Les représentations graphiques soumises à un zoom sémantique peuvent générer des présentations différentes en fonction du niveau de zoom courant. Par exemple, un logiciel est représenté par un ensemble de paquetages à un faible niveau de zoom ; à mesure que l'on applique un zoom sémantique dessus apparaissent les sous-paquetages, puis les classes, et enfin les structures internes des classes (les méthodes). Pour faire un passage flexible du haut niveau (classes) au bas niveau (méthodes), nous utilisons la transparence qui nous permet de visualiser la structure interne des représentations graphiques des classes sans avoir à changer le contexte. Nous proposons dans ce qui suit la représentation graphique des métriques au niveau des méthodes.

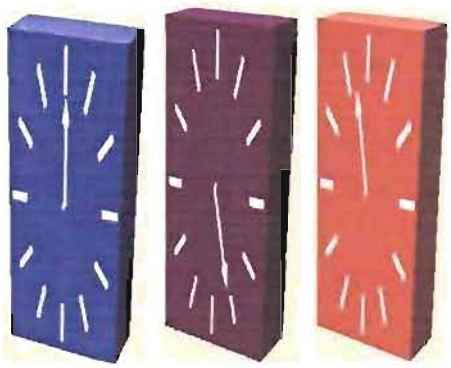

Figure 4.6 – La variation d'orientation en texture d'horloges. (Gauche) La valeur minimal. (Milieu) La valeur moyenne. (Droite) La valeur maximale.

4.2.1 La variation de taille

Cette technique consiste à représenter chaque méthode d'une classe par un segment de droite de taille (longueur) proportionnelle à la métrique de la méthode. La variation de taille entraîne une variation de visibilité des éléments graphiques, elle est donc :

- **dissociative**, ce qui signifie que des éléments graphiques possédant différentes variations de la composante ne peuvent pas être regroupés visuellement au sein d'une même entité visuelle.
- **sélective**, ce qui signifie qu'on peut isoler visuellement tous les éléments graphiques de même taille, ce qui n'est possible que lorsque la longueur de la variation de la composante est réduite à 4 ou 5 valeurs. De ce fait, la détection d'une cible dont la taille diffère de celle des autres signes est aisée.
- **ordonnée et quantitative**, ce qui signifie qu'il est possible d'ordonner les éléments graphiques selon leur taille, mais qu'il est également possible de les comparer.

La longueur de la variation de taille dans une représentation n'est limitée que par la place disponible dans l'espace de représentation. On peut toutefois retenir cette règle de base : l'oeil différencie en moyenne 20 paliers entre deux points dont les tailles varient de 1 à 10 [2]. Pour pouvoir estimer avec précision le nombre et la taille des segments,

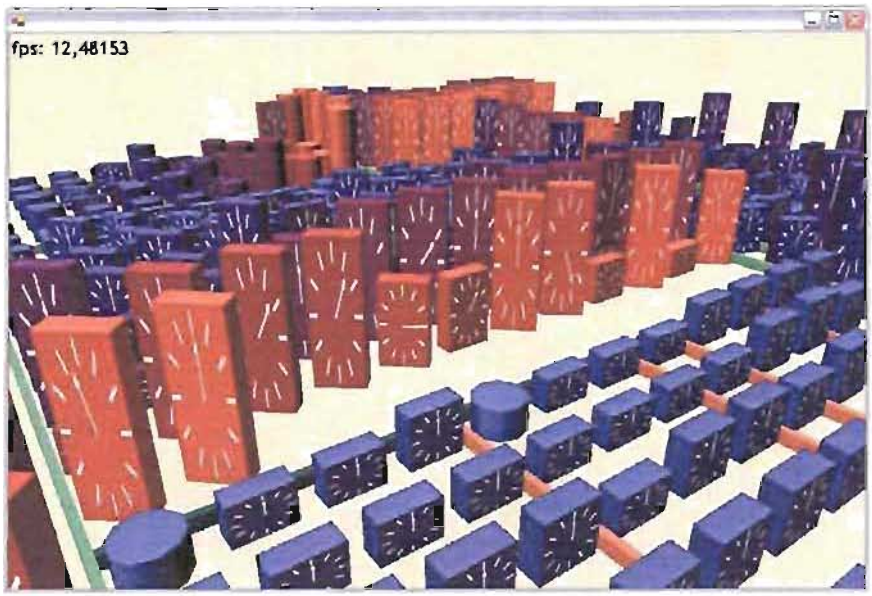

Figure 4.7 – Un exemple de visualisation du système *PCGEN*. La texture de l'horloge est associée à la métrique NOP.

nous proposons d'utiliser deux grilles. La première grille est dessinée sur les surfaces internes des boîtes, elle va nous servir comme une échelle de 1 à 10 sur la taille des segments. La deuxième grille divise le plancher de la classe en dix rectangles (chaque rectangle peut contenir 60 segments), cette grille va nous servir comme un indicateur du nombre des segments.

La variation de taille peut être utilisée pour la réalisation de la tâche de détection de la *Décomposition Fonctionnelle*, une anomalie de conception qui peut avoir lieu lorsqu'une classe est conçue avec l'intention d'accomplir une fonction unique dans le système. La classe possède donc une grande méthode qui est responsable de l'implémentation de cette fonction. Les conséquences d'une telle conception sont que les classes n'utilisent pas le mécanisme d'héritage (DIT très faible), et que les classes n'ont pratiquement qu'une seule méthode ou très peu de méthodes qui sont très grandes et complexes, ce qui influence la complexité de la classe (WMC élevé) [11].

La Figure 4.8 illustre un exemple d'utilisation de la variation de la taille. Dans cet exemple, la couleur des classes est associée à la profondeur d'héritage (DIT), l'orienta-

tion est associée à la cohésion (LCOM5) et enfin la taille est associée à la complexité (WMC). La détection de la *Décomposition Fonctionnelle* consiste à repérer les classes qui sont bleues (DIT très faible), grandes (WMC élevé) et qui ont très peu de méthodes complexes. Pour vérifier la complexité des méthodes, nous devons visualiser les métriques de bas niveau en utilisant la variation de taille, pour repérer celles qui sont très peu nombreuses et complexes. Nous remarquons que la classe qui se trouve à droite répond aux critères de l'anomalie *Décomposition Fonctionnelle*, car elle a juste trois méthodes qui sont très complexe, tandis que la deuxième (à gauche), elle ne l'est pas, car son comportement est bien réparti sur toutes ses méthodes (elle a plus d'une centaine de méthodes de complexité très faible).

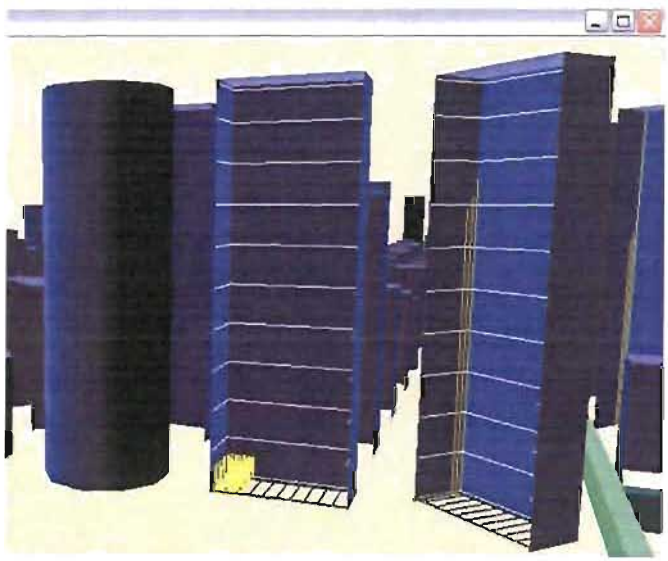

Figure 4.8 – Un exemple d'application de la variation de taille sur le système *PMD*, pour la visualisation de la métrique de complexité des méthodes.

4.2.2 La variation de couleur

Cette technique consiste à représenter chaque méthode d'une classe par une zone sur la surface du plancher de la boîte. La valeur de la métrique d'une méthode est représentée par un dégradé de couleur allant du bleu vers le rouge. La variation de couleur dans ce

cas est une variation de la quantité du bleu et du rouge sur cette zone (voir la Figure 4.9).

Comme la taille, la variation du dégradé de couleur entraîne également une variation de la visibilité. La couleur est donc **dissociative**. Elle est également **sélective** si la longueur de la variation de dégradé de couleur est limitée à **4 ou 5** valeurs. Le dégradé de couleur est **ordonné** mais non **quantitatif** parce que les différents niveaux du dégradé de couleur peuvent être classés mais il est impossible de chiffrer la différence de deux valeurs : le bleu et le rouge ne peuvent servir d'unité de mesure.

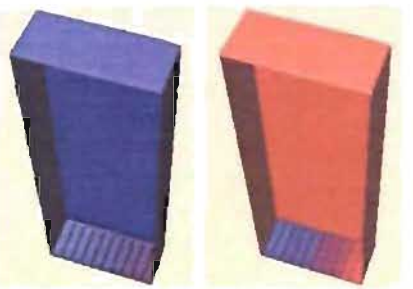

Figure 4.9 – La variation de dégradé de couleur.

4.3 Association entre les métriques et les indices visuels

Les Sections 4.1 et 4.2 ont justifié les indices visuels choisis pour la visualisation des métriques des logiciels et le Chapitre 3 a répertorié certaines métriques intéressantes à l'étude d'un logiciel. Il reste par contre à faire un lien judicieux entre ces deux composantes pour avoir une visualisation utile pour l'analyse des logiciels.

La correspondance est linéaire entre les métriques et les indices visuels (taille, orientation, couleur, etc). Cette correspondance est donnée par la formule suivante :

$$G_v = (G_{max} - G_{min}) \left(\frac{M_v - M_{min}}{M_{max} - M_{min}} \right)$$

où M_v est la valeur de la métrique. M_{min} et M_{max} sont respectivement les minimum et maximum pratiques des métriques. Ils représentent des seuils fixés par le développeur. Les valeurs dépassant ces seuils sont ramenées vers ce minimum ($M_v = M_{min}$, si

$v < M_{min}$) ou ce maximum ($M_v = M_{max}$, si $v > M_{max}$). G_v est la valeur graphique résultante, G_{min} et G_{max} sont respectivement les valeurs minimum et maximum pratiques de l'indice visuel. Ces valeurs seuils sont considérées comme des maxima ou des minima pratiques. Leur utilisation permet de représenter les éléments sans que la forme de certains d'entre eux ne dégénère. En effet, ramener les valeurs considérées comme extrêmement grandes ou extrêmement petites à des niveaux comparables permet toujours à l'analyste de comprendre que ces valeurs sont extrêmes et de prendre une décision en conséquence.

4.4 Discussion

Pour être capable de définir un nouveau domaine d'expression, il faut évidemment s'intéresser à ce que l'observateur est en mesure de percevoir dans une représentation graphique. C'est pourquoi nous devons définir certains critères pour évaluer l'efficacité des indices visuels des textures :

- **intrinsèque** : Les indices visuels doivent être invariants selon le changement du point de vue de l'utilisateur.
- **discriminant** : Les indices visuels doivent posséder des propriétés qui permettent de les discriminer.
- **précis** : Les indices visuels doivent pouvoir être estimés avec précision car la prédiction des valeurs des données visualisées en dépend.

Nous avons déjà évoqué dans le Chapitre 2, les niveaux d'organisation des primitives visuelles proposé par Bertin [2]. Cette caractérisation, qui tient compte des propriétés perceptives des primitives visuelles, peut nous servir comme un guide pour le choix de la texture la plus efficace pour visualiser une métrique donnée. Le Tableau 4.1 présente la classification des différentes représentations graphiques qui ont été proposées dans ce chapitre. Cette classification est basée sur l'échelle de Bertin et les critères définis ci-haut. La texture de roches n'est ni intrinsèque ni discriminante, car sa perception varie par le changement du point de vue de l'utilisateur, et elle ne permet pas d'estimer la

valeur de la métrique qui lui est associée avec précision. Nous estimons que ces deux critères sont importants, c'est la raison pour laquelle nous avons choisi la texture de briques pour visualiser des données qualitatives, la texture de fenêtres pour visualiser des données discrètes, et la texture de l'horloge pour visualiser les données continues. La Figure 4.10 présente l'interface graphique pour l'ajout et suppression de textures.

Indices visuels	Roches	Fenêtres	Horloges	Briques	Taille	Couleurs
Point de vue	*local*	*global*	*global*	*local*	*local*	*local*
Associative	•	•		•		
Sélective	•	•	•		•	•
Ordonnée	•	•	•		•	•
Quantitative		•	•		•	•
Intrinsèque		•	•			
Discriminant	•	•	•	•	•	•
Précis		•	•	•	•	•

Tableau 4.1 – Classification des textures et des représentations de bas niveau.

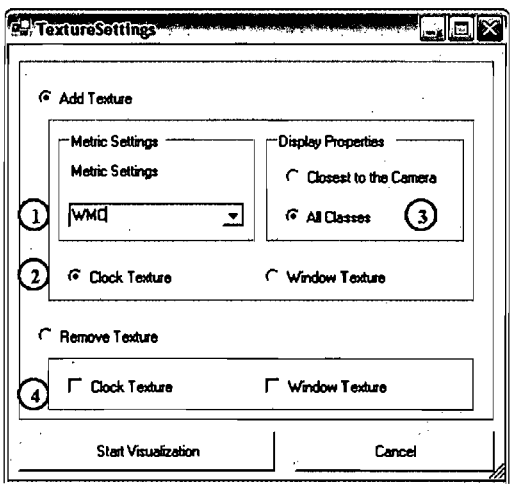

Figure 4.10 – Présentation de l'interface du choix de textures. (1) Sélection de la métrique à visualiser. (2) Ajout d'une texture. (3) Sélection du type d'affichage des textures (sur toutes les classes ou seulement sur les classes proches de la caméra). (4) Suppression d'une texture

CHAPITRE 5

GÉNÉRATION DE TÂCHES INTERACTIVES

La manipulation de grandes quantités d'informations par les systèmes de visualisation n'est pas une opération transparente vis-à-vis de l'utilisateur. C'est ce que l'on peut constater au travers d'exemples d'activités utilisateur, qui mènent celui-ci à s'impliquer dans le processus de traitement d'une grande masse d'informations.

Lorsqu'il s'agit d'explorer pour comprendre une masse importante d'informations, l'utilisateur fait face à plusieurs problèmes. Ceci arrive en particulier lorsque son usage des outils mis à sa disposition est inadéquat ou lorsque ces outils s'avèrent incapables de répondre à ses besoins.

Comme nous l'avons déjà évoqué dans le Chapitre 2, s'intéresser à la visualisation d'informations, c'est s'intéresser aux processus informatiques et humains qui permettent la communication visuelle d'information au travers d'une représentation graphique. Pour *communiquer l'information*, il est nécessaire de disposer d'un moyen d'exprimer graphiquement cette information, mais il est également nécessaire de connaître les règles qui gouvernent la perception et la compréhension de ce mode d'expression. Ce sont ces deux aspects de la visualisation que nous traitons dans ce chapitre : d'une part l'utilisation des règles de la perception visuelle humaine pour la génération de représentations graphiques les plus efficaces aux données à visualiser, et d'autre part la génération automatique des scénarios d'interactions spécifiques à l'outil de visualisation qui permettent à l'usager d'utiliser convenablement cet outil pour effectuer sa tâche d'analyse.

5.1 La génération automatique de tâches d'analyse

Afin de satisfaire les besoins d'utilisateur en termes de navigation et de tâches d'analyse, nous proposons dans ce cadre une approche dirigée par les modèles (*MDA-Model Directed Architecture*), pour la modélisation d'un système d'aide à la visualisation qui assiste l'utilisateur durant tout le processus de la visualisation.

Dans cette approche, nous proposons un modèle d'analyse pour la description des tâches d'analyse qui se base sur les taxonomies de tâches élémentaires qu'un utilisateur pourrait accomplir dans un environnement visuel. Les taxonomies des tâches élémentaires n'étant pas assez riches, nous avons proposé un langage étendu qui permet à l'utilisateur l'expression fine de ses tâches d'analyse, en déléguant à un système d'aide la génération automatique des interactions et de la visualisation la plus adéquate à ses besoins.

En plus de proposer des modes d'interaction à l'utilisateur pour manipuler les données qui lui sont présentées, notre système d'aide doit pouvoir répondre aux besoins liés aux tâches de l'utilisateur, mais aussi l'aider à trouver des représentations graphiques (par raffinage du niveau de granularité, textures, couleurs, etc.) efficaces pour visualiser ses données. Nous avons décrit les modes d'interaction par un modèle d'interaction qui se base sur la taxonomie de navigation élaborée par Shneiderman [36]. Le rôle du système d'aide est de faire la transformation d'une tâche d'analyse qui est conforme au modèle d'analyse en une suite d'interactions offertes par l'outil de visualisation qui représente une instance du modèle d'interaction. Cette transformation inclut aussi le *mapping* entre les données manipulées par cette tâche d'analyse et les représentations graphiques disponibles dans l'outil de visualisation.

Par ailleurs, il est nécessaire de connaître les règles qui gouvernent la perception et la compréhension de ce mode d'expression. Il faut cependant noter que l'efficacité d'une représentation graphique est difficile à évaluer de façon absolue. Nous proposons de l'évaluer par rapport à un contexte particulier : tâches de l'utilisateur et caractéristiques des données.

Plusieurs techniques d'Intelligence Artificielle permettent de faire de l'assistance à la conception en général en exploitant la connaissance d'expert du domaine concerné. Les principales approches utilisées sont les raisonnements à base de cas et à base de contraintes. Les raisonnements à base de cas s'appuient sur des connaissances implicites regroupées dans des cas, alors que les raisonnements à base de contraintes, sur des connaissances explicites et formalisées. En effet, la modélisation du problème d'évaluation de l'efficacité d'une représentation graphique par un problème de satisfaction

de contraintes est presque naturelle, et sa résolution en programmation par contraintes s'avère souvent plus efficace. La base des contraintes regroupe l'ensemble des règles qui évaluent l'efficacité des représentations graphiques par rapport à un contexte particulier. Ces règles se basent sur trois aspects fondamentaux : les caractéristiques des données visualisées, la nature de la tâche de l'utilisateur, et la perception de ces représentations graphiques par l'utilisateur (règles issues des études menées en psychologie cognitive sur la perception visuelle, décrites dans le Chapitre 2).

5.2 Aperçu de l'approche

Notre approche consiste à proposer des modules qui sont en charge de l'exécution des tâches d'analyse ou de visualisation des données. Afin d'adapter chacun des modules aux besoins particuliers de leurs applications (outils de visualisation spécifiques), nous avons réalisé des modèles qui décrivent la structure et le comportement dynamique des outils de visualisation. Pour aider les concepteurs dans la réalisation de tels modèles, pour chacun des modules nous proposons des modèles généraux, à partir desquels les concepteurs peuvent créer leurs modèles spécifiques à leurs outils de visualisation par spécialisation ou instantiation.

Notre approche est centrée sur les buts de l'utilisateur ; elle couvre toutes les étapes du processus de visualisation, de la description de tâches d'analyse jusqu'à la génération des séquences d'actions ou d'interactions que l'utilisateur doit exécuter pour atteindre son objectif. La Figure 5.1 illustre notre approche qui consiste en quatre modules :

- **modèle d'analyse** : Ce modèle offre un langage pour la description fine des tâches utilisateur. Ce modèle est indépendant de la visualisation.
- **modèle d'interaction** : Ce modèle décrit les modes d'interaction qu'un outil de visualisation devrait supporter pour répondre aux besoins de l'utilisateur en terme de navigation.
- **mécanisme de transformation** : Ce mécanisme joue le rôle de chef d'orchestre de notre système en assurant les liens entre les deux modèles. Il définit les règles de transformations d'un scénario de tâche d'analyse en une séquence d'interactions.

– **session de transformation** : C'est une instance du mécanisme de transformation qui prend en compte la spécification de l'outil de visualisation.

Dans les sections suivantes, nous allons décrire chacun de ces modules en détails.

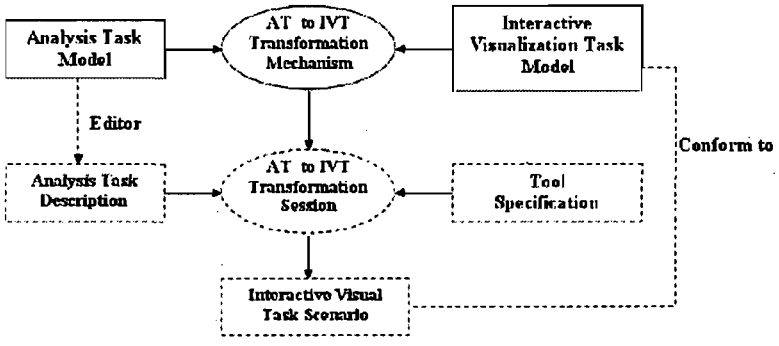

Figure 5.1 – Aperçu du processus de transformation.

5.3 Modèle d'analyse

Le modèle d'analyse permet d'identifier les types de connaissances que l'utilisateur doit avoir pour réaliser sa tâche. La Figure 5.2 illustre le modèle d'analyse, qui décrit les tâches d'analyse en termes de buts, méthodes, opérateurs et règles de sélection. Les procédures décomposent récursivement les tâches en sous-tâches jusqu'à atteindre les tâches élémentaires, appelées opérateurs.

Un but définit un état recherché. Il lui est associé un ensemble de méthodes qui toutes conduisent à cet état. Les buts sont organisés de manière hiérarchique : un but complexe est atteint lorsque tous ses sous-buts sont satisfaits, et ceci récursivement jusqu'à atteindre les buts élémentaires. Les buts élémentaires sont réalisés par l'exécution d'une suite d'opérateurs. Les buts forment donc une structure arborescente dont les feuilles sont des opérateurs.

Un opérateur est une action élémentaire dont l'exécution provoque un changement d'état (état mental de l'utilisateur et/ou état de l'environnement). Un opérateur est une tâche d'analyse élémentaire qu'un utilisateur pourrait exécuter sur un ensemble de don-

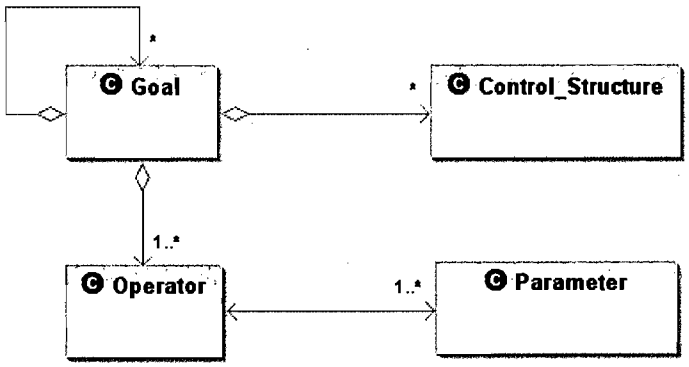

Figure 5.2 – Modèle d'analyse.

nées. Nous avons défini un ensemble d'opérateurs, par adaptation et l'exécution de la taxonomie d'opérateurs proposée par Amar *et al.* [1].

Plus spécifiquement, nous avons caractérisé chaque opérateur par les paramètres qu'il manipule et la portée de son application. Un opérateur est alors décrit par une paire $< operation, parametres >$ où l'opération est une action à exécuter sur les données utilisant les paramètres spécifiés. Les paramètres incluent une entité de logiciel (x) ou l'ensemble d'entités (X), une métrique logicielle ou une relation entre entités (a), une condition (c), une étiquette (l) et une propriété (p) comme le nom de classe, la position de classe en ce qui concerne l'architecture de paquetage, etc.

Chaque opérateur a une portée de son application : globale ou locale. Un opérateur global peut être appliqué à un grand ensemble d'entités de code (des classes ou des interfaces), alors qu'un opérateur local doit être exécuté sur un sous-ensemble réduit d'entités. Notre ensemble d'opérateurs est décrit dans le Tableau 5.1. Par exemple, pour rechercher les classes qui sont fortement couplées (ayant une métrique CBO très élevée) dans un programme P, l'opérateur à employer serait *Find Extremum(C, CBO, CBO = HIGH)* avec les paramètres C, l'ensemble de classes dans P, *CBO* comme métrique de couplage, et *HIGH* comme condition.

Une méthode décrit le procédé qui permet d'atteindre un but. Elle s'exprime sous la forme d'une suite conditionnelle de buts et d'opérateurs où les conditions font référence à l'état de l'environnement. Les méthodes représentent la suite des étapes à suivre,

Opérateur	Portée	Description
Locate$\langle X, c_1, ..., c_n \rangle$	global	Étant donné l'ensemble X, trouver les entités qui satisfont aux conditions $c_1, ..., c_n$.
Select$\langle X, c_1, ..., c_n \rangle$	global	Trouver et marquer le sous ensemble de X qui satisfait aux conditions $c_1, ..., c_n$.
Retrieve Value$\langle x, a \rangle$	local	Étant donnée une entité x, trouver la valeur de l'attribut a.
Relate$\langle x, r \rangle$	global	Déterminer le sous ensemble d'entités qui sont en relation r avec l'entité x.
Find Extremum$\langle X, a \rangle$	global	Étant donné l'ensemble X, trouver les entités qui possèdent une valeur extrême de attribut a.
Sort$\langle X, a \rangle$	global	Étant donnée un ensemble d'entités X, l'ordonner selon l'attribut a.
Characterize Distribution$\langle X, a \rangle$	global	Trouver la distribution des valeurs de l'attribut a dans l'ensemble X.
Find Anomalies$\langle X, c \rangle$	global	Identifier n'importe quelle anomalie dans l'ensemble d'entités X en respectant les conditions c.
Cluster$\langle X, a_1, ..., a_n \rangle$	global	Étant donné l'ensemble d'entités X, trouver les *clusters* dont les valeurs d'attributs sont similaires $a_1, ..., a_n$.
Correlate$\langle X, a_1, ..., a_n \rangle$	global	Étant donné l'ensemble d'entités X, déterminer les relations entre les valeurs des attributs $a_1, ..., a_n$.
Verify Value$\langle x, a, c \rangle$	local	Vérifier si la condition c est satisfaite pour un attribut a de l'entité x.
Verify Property$\langle X, p \rangle$	global	Vérifier si la propriété p est satisfaite pour l'ensemble d'entités X.
Inspect$\langle x, p_1, ..., p_n \rangle$	local	Obtain detailed information about properties $p_1, ..., p_n$ for an entity x.
Decide$\langle c, op \rangle$	N/A	L'utilisateur prend une décision acceptant ou refusant l'exécution de l'opérateur op, selon la satisfaction de la condition c.
Save$\langle x, l \rangle$	local	Étiqueter l'entité x par l.

Tableau 5.1 – Opérateurs.

chaque étape correspond, soit à l'accomplissement d'un but ou l'exécution d'un opérateur, soit à une décision à prendre. Les décisions sont exprimées par des règles de sélection.

Une règle de Sélection exprime le choix d'une méthode lorsqu'il y a conflit (c'est-à-dire lorsque plusieurs méthodes conduisent au même but), ou lorsqu'il y a une boucle à répéter. Une règle peut être de la forme :

1. IF <condition-sur-la-situation-actuelle-est-vraie>
   THEN utiliser la méthode $M1$;
   ELSE utiliser la méthode $M2$
2. FOR EACH <condition-sur-la-situation-actuelle-est-vraie>
   utiliser la méthode M ;
   END FOR

Le Tableau 5.2 présente la description de notre exemple de détection de *décomposition fonctionnelle* utilisant le modèle de tâche d'analyse. L'anomalie *décomposition fonctionnelle* se produit lorsqu'une classe est conçue avec l'intention d'accomplir une fonction unique dans le système. La classe possède donc une grande méthode qui est responsable de l'implémentation de cette fonction. La plupart des attributs de la classe sont privés et utilisés principalement à l'intérieur de la classe. Les conséquences d'une telle conception sont que les classes n'utilisent pas le mécanisme d'héritage (DIT très faible), et n'ont pratiquement qu'une seule méthode publique déclarée (NPDM = 1). Cette méthode est très grande et complexe, ce qui influence la complexité de la classe (WMC élevé) [11].

La tâche présentée en Tableau 5.2, consiste en un seul but (détection de *décomposition fonctionnelle*) et une seule méthode pour accomplir ce but. Cette méthode consiste en une suite d'étape à suivre. Pour la première étape, l'analyste doit choisir des classes du système, ayant une valeur élevée de WMC, une valeur de NPDM faible, et une valeur faible de DIT. Par la suite, d'autres opérateurs sont appliqués sur chacune de ces classes sélectionnées ; ceci est représenté par la boucle *FOR EACH*. Pour chaque classe

sélectionnée, l'analyste inspecte d'abord son code source, et recueille des informations sur le nom et des attributs de la classe. Puis, l'analyste doit décider si le nom de la classe et le nombre d'attributs privés impliquent que cette classe choisie est une occurrence de *décomposition fonctionnelle*. Si c'est le cas, cette classe doit être étiquetée.

Method to accomplish Goal : *Funct. Decomposition* Detection
1 : Select(whole system, $WMC = HIGH$, $NPDM = LOW$,
$DIT = LOW$)
2 : FOR EACH (c, selected classes)
3 : Inspect(c, class name, attributes)
4 : Decide(class name indicates a function and
there are several private attributes,
Save(c, functional decomposition))
5 : END FOR EACH

Tableau 5.2 – Exemple de description de la tâche de détection de *Décomposition Fonctionnelle* par le modèle d'analyse.

5.4 Modèle d'interaction

Le modèle d'interaction décrit les modes des interactions qu'un outil de visualisation devrait supporter pour répondre aux besoins utilisateur en terme de navigation. Ce modèle est basé sur la taxonomie proposée par Shneiderman [36]. Cette taxonomie présente sept tâches interactives de haut-niveau qu'un outil de visualisation de l'information devrait supporter, comme la vue d'ensemble, le zoom, le détails-sur-demande, etc. Ces tâches interactives sont des actions indépendantes du domaine de visualisation, et elles permettent aux utilisateurs de manipuler leurs données dans un environnement visuel.

Dans un environnement visuel, les entités (données) sont visualisées par des représentations graphiques. Les propriétés de ces entités sont associées aux attributs visuels de leur représentation. Les attributs visuels d'une représentation peuvent changer selon la tâche interactive effectuée là-dessus. Les diverses tâches interactives sont décrites comme suit :

- **overview** : Avoir une vue d'ensemble de la collection entière.

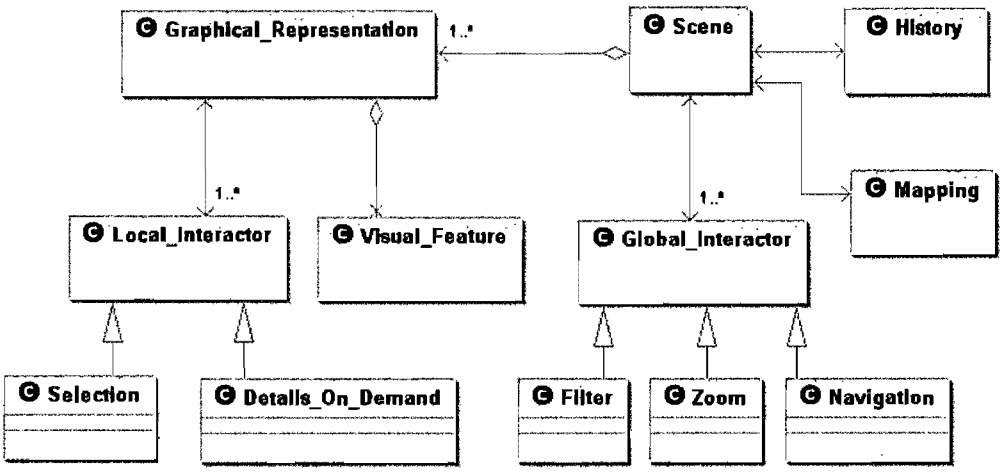

Figure 5.3 – Interactive Visual Task Model.

- **zoom** : Appliquer un zoom-in sur les entités.
- **filter** : Mettre l'emphase sur un sous-ensemble d'entités vérifiant une propriété donnée.
- **details-on-demand** : Afficher les détails d'une entité ou d'un groupe d'entités.
- **relate** : Visualiser les relations entre les entités.
- **history** : Conserver un historique des étapes franchies.
- **extract** : Extraire les sous-ensembles.
- **selection** : Mémoriser certaines entités de la collection.
- **navigation** : Naviguer autour d'une collection.

5.5 Transformation

Le mécanisme de transformation assure les liens entre le modèle d'analyse et le modèle d'interaction. Il identifie les règles de transformations d'un scénario de tâche d'analyse en séquence d'interactions. Le mécanisme de transformation peut être décomposé en deux étapes, une pour la transformation des attributs et l'autre pour la transformation des tâches d'analyse.

1. **La transformation des tâches** consiste à trouver pour une tâche analytique don-

née quel mode d'interaction utiliser pour accomplir une tâche dans l'environnement visuel.

2. **La transformation des attributs** consiste à établir les associations entre les primitives visuelles (couleur, taille, orientation, texture, etc.) et les données à visualiser (les métriques de logiciel, dans le cas de VERSO).

5.5.1 Transformation des tâches

Le Tableau 5.3 présente le scénario d'interaction généré pour chaque opérateur. Par exemple, l'exécution de l'opérateur *Find Extremum*$\langle X, a \rangle$ nécessite tout d'abord d'avoir une vue globale des entités X, ce qui correspond à la tâche interactive *Overview*. En VERSO cet dernière correspond à l'application de *zoom-out*. Ensuite, un filtre doit être appliqué pour visualiser la distribution des valeurs pour l'attribut a, ce qui correspond au *filtre statistique* en VERSO. Finalement, pour identifier des classes ayant des valeurs de l'attribut a très élevées, un *itérateur* doit être employé en VERSO pour visiter et choisir la représentation graphique de ces classes.

Revenons à notre exemple, une fois que la description de la détection de la *décomposition fonctionnelle* spécifiée par le modèle de tâches d'analyse, un scénario spécifique de VERSO est généré automatiquement. Puisque chaque attribut impliqué dans la tâche d'analyse est employé par au moins un opérateur global, les attributs visuels disponibles sont la couleur, la hauteur et l'orientation.

Le scénario généré est présenté dans le Tableau 5.4. Bien que le scénario de VERSO contienne plus d'instructions que la description de départ (voir le Tableau 5.2), nous avons gardé la même numérotation pour montrer de quelle façon les opérateurs de la description ont été transformés en séquences d'actions dans le scénario.

5.5.2 Transformation des attributs

Parmi les principaux formalismes que l'on peut utiliser pour représenter le problème de transformation des attributs, on peut citer la satisfaction de contraintes. Nous avons choisi ce formalisme pour sa simplicité et son efficacité. En effet, la modélisation du

Opérateurs	*Interacteur*	Génération VERSO Description
Locate$\langle X, c_1, ..., c_n \rangle$	Overview	Among X, locate classes whose $c_1, ..., c_n$
Select$\langle X, c_1, ..., c_n \rangle$	Overview	Among X, select classes whose $c_1, ..., c_n$
Retrieve Value$\langle x, a \rangle$	Zoom Details-on-demand	Apply the `Zoom-In` if necessary `Display list of properties` for class x Read the value for metric a
Relate$\langle x, r \rangle$	Overview Relationship Filter	Apply the `Zoom-Out` if necessary Apply the `Relationship Filter` r on class x
Find Extremum$\langle X, a \rangle$	Overview Filter Selection	Apply the `Zoom-Out` if necessary Apply the statistical filter on a for X Using the `Iterator`, `Select` the classes as long as the metric a is judged high enough Apply the `Statistical Filter` on a for X Remove the Statistical Filter
Sort$\langle X, a \rangle$	Overview Navigation	N/A
Characterize Distribution$\langle X, a \rangle$	*Overview*	Apply the `Zoom-Out` if necessary Apply the `Statistical Filter` on a for X
Cluster$\langle X, a_1, ..., a_n \rangle$	*Overview* *Filter* *Selection*	Apply the `Zoom-Out` if necessary Find clusters in X sharing similar values for attributes $a_1, ..., a_n$ `Select` classes in every cluster
Correlate$\langle X, a_1, ..., a_n \rangle$	*Overview*	Apply the `Zoom-Out` if necessary Find if $a_1, ..., a_n$ are correlated in X
Verify Value $\langle x, a, c \rangle$	*Zoom*	Apply the `Zoom-In` if necessary Verify that (graphical attribute mapped to metric a) is c for class x
Verify Property $\langle x, p \rangle$	*Overview*	Apply the `Zoom-Out` if necessary Verify that x has p
Inspect$\langle x, p \rangle$	*Zoom* *Details-on-demand*	Apply the `Zoom-In` if necessary `Display source code` of x Inspect p
Decide$\langle c, op \rangle$	N/A	if c, generated description of operator op
Save$\langle X, l \rangle$	*Zoom* *Selection*	Apply the `Zoom-In` if necessary `Mark` X as l

Tableau 5.3 – La transformation des tâches.

Method to accomplish Goal : *Funct. Decomposition* Detection
1 : Among whole system, select classes whose color is BLUE, height is HIGH, twist is LOW
2 : FOR EACH c in selected classes
3 : Display source code of c
Inspect class name
Inspect attributes
4 : If class name indicates a function and there are several private attributes, mark c as functional decomposition
5 : END FOR EACH

Tableau 5.4 – Scénario de détection de la *décomposition fonctionnelle* spécifique à VERSO.

mécanisme de transformation par un problème de satisfaction de contraintes est presque naturelle, et sa résolution en programmation par contraintes s'avère souvent plus efficace.

Dans les sections qui suivent, nous rappelons les notions de base liées aux problèmes de satisfaction de contraintes (*Constraint Satisfaction Problem* CSP) [40] ainsi que leur utilisation pour la génération du *mapping*.

5.5.2.1 Problème de satisfaction de contraintes

Le cadre des problèmes de satisfaction de contraintes offre la possibilité de modéliser des problèmes dont les variables doivent prendre certaines valeurs, appartenant à leur domaine (ensemble des choix possibles), sous certaines conditions, appelées contraintes. Plus formellement :

Un *CSP* est défini par le triplet $\langle X, D, C \rangle$ avec :

- $X = \{X_1, X_2, ..., X_n\}$ un ensemble fini de n variables ;
- $D = \{D_1, D_2, ..., D_n\}$ où chaque D_i représente l'ensemble fini des valeurs possibles pour chaque variable X_i (son domaine) ;
- $C = \{C_1, C_2, ..., C_m\}$ un ensemble de m contraintes reliant des variables.
  Une contrainte C_i relative à l'ensemble des variables $var(C_i) = (X_{i1}, ..., X_{ij})$ où j

est l'arité de la contrainte, est un sous-ensemble du produit cartésien $D_{i1} \times ... \times D_{ij}$ qui spécifie les combinaisons de valeurs autorisées pour les variables $X_{i1}, ..., X_{ij}$.

Ainsi, pour les problèmes d'association des primitives visuelles aux métriques à visualiser, nous retrouvons la notion d'interférences (voir Chapitre 2) où :

- X, l'ensemble des variables est composé des métriques,
- D_i, les domaines regroupent l'ensemble des indices visuels associés à chaque donnée (ils peuvent être identiques),
- C, les contraintes sont modélisées par de simples équations et inéquations généralement binaires pour les associations, ces contraintes sont basées essentiellement sur le type des données (métriques) et les propriétés des indices visuels.

La fonction $f_S(variable, operator, visualfeature)$ regroupe les contraintes C_i. Elle évalue l'efficacité d'un indice visuel pour une métrique donnée, en se basant sur :

- L'échelle de mesure de la métrique (continue ou discrète, ordinale, nominale). Par exemple, la texture des fenêtres est caractérisée par une perception **quantitative**. Elle nous permet de définir le rapport numérique entre différentes variations de cette texture, ce qui est très efficace pour une variable discrète. Tandis que la texture de l'horloge est en VERSO plus appropriée aux variables continues.
- La portée (locale ou globale) de l'opérateur (voir Section 5.3) que l'utilisateur souhaite appliquer sur cette variable. Par exemple, les textures sont plus utilisées pour visualiser les variables manipulées par un opérateur de portée locale, car elles sont plus efficaces lorsque le point de vue de l'utilisateur est proche (par l'application du zoom-in), mais perd son efficacité lorsqu'il s'éloigne par l'application du zoom-out.
- Le phénomène d'interférence (voir Section 2.1.1.2) qui peut avoir lieu entre les indices visuels. Par exemple, la variation de brillance interfère avec la variation de la teinte, ce qui interdit l'affectation de la brillance et de la teinte à deux variables simultanément.

Si le formalisme des problèmes de satisfaction de contraintes offre un cadre de modélisation à de nombreux problèmes, en revanche, il ne semble pas apporter une réponse satisfaisante à la modélisation des problèmes dans lesquels entre en jeu la notion de coût ou de préférence. Ainsi, alors que dans les CSP classiques, la notion de contrainte est définie au sens strict du terme (on emploie alors la terminologie de contrainte dure), dans laquelle une combinaison de valeurs peut-être autorisée ou interdite. Pour de nombreux problèmes réels, le concept de contrainte ne traduit en fait qu'une préférence pour certains tuples de valeurs, ou encore un coût de violation (notion de contrainte molle).

L'expérience montre pourtant que la plupart des problèmes d'association des indices visuels aux données à visualiser, sont des problèmes mixtes de satisfaction et d'optimisation, où coexistent des contraintes dures qui traduisent des obligations et doivent impérativement être satisfaites, et des contraintes souples qui représentent des préférences et sont à satisfaire au mieux. Comme nous l'avons déjà évoqué dans le Chapitre 2, les chercheurs en perception humaine et sciences cognitives ont défini des échelles d'expressivité et d'efficacité pour évaluer les indices visuels pour chaque type de données. Ces différents niveaux de priorité justifient l'extension du cadre CSP au cadre des CSP valués (*VCSP* pour *Valued Constraint Satisfaction Problems*) par l'introduction de valuations pour les contraintes et les affectations.

5.5.2.2 Définition des contraintes

Nous avons construit une base de contraintes, qui peut servir comme un guide lors de la génération automatique du *mapping* entre les attributs des opérateurs (*e.g.* les métriques de logiciels) et les indices visuels. Pour exploiter au mieux les qualités des attributs à visualiser, chaque contrainte doit vérifier les points suivants :

1. **L'échelle de mesure de l'attribut** : Certaines contraintes portent sur l'échelle de mesure de l'attribut (discret ou continu, ordonné, nominal) et les propriétés de perception de l'indice visuel qui peut être utilisé, pour le visualiser. Ces contraintes vérifient si le nombre des valeurs possibles de cet attribut est supporté par l'indice visuel. Un facteur de pénalité est attribué à la violation d'une contrainte. Il est dé-

fini à partir de l'échelle d'expressivité de Mackinlay [29].

Par exemple, les contraintes qui portent sur la teinte sont définies comme suit : chaque *attribut discret* est défini par un ensemble fini des valeurs possibles. Ce nombre dicte le nombre de couleurs uniques (teintes) que nous devons utiliser pour représenter l'attribut. Il faut vérifier si le nombre des valeurs possibles de cet attribut est supporté par la primitive teinte : le champ de variation de l'attribut ne doit pas dépasser sept valeurs possibles (car le système visuel humain prouve des difficultés à détecter plus de sept couleurs distinctes [2]).

Lorsque l'on a affaire à *un attribut à échelle nominale* (variable qualitative), le nombre de valeurs possibles que peut prendre cet attribut ne doit pas dépasser sept valeurs. Dans le cas des *attributs à échelle ordonnée*, non seulement le nombre de valeurs possibles ne doit pas dépasser sept, mais aussi il faut considérer l'échelle d'expressivité [29]. Selon cette échelle, si le type de l'attribut est ordonné, la variation de teinte est classée au troisième rang (voir Tableau 5.5) ; Alors, un facteur de pénalité de trois est attribué à la violation de cette contrainte.

Lors du traitement des attributs quantitatifs (continus ou discrets) nous attribuons un facteur de pénalité de huit à la violation de cette contrainte. Ceci est fait parce que selon l'échelle d'expressivité de Mackinlay la teinte est la primitive la moins expressive pour un attribut quantitatif. Si l'attribut est discret, alors, le nombre de valeurs discrètes doit être réduit à sept valeurs ; si l'attribut est continu, alors nous regroupons les valeurs possibles de cet attribut en cinq intervalles, une couleur sera associée à chaque intervalle.

2. **La fréquence spatiale de l'attribut** : Nous avons construit un ensemble de contraintes, qui vérifient si la fréquence spatiale de l'attribut est appropriée pour l'indice visuel en question. Par exemple, la primitive teinte n'est pas appropriée pour des attributs

ayant une fréquence spatiale élevée. Ceci est basé sur des résultats bien connus de perception.

3. **Le phénomène d'interférence** : D'autres contraintes sont aussi définies pour vérifier si les autres indices visuels peuvent interférer avec l'indice visuel en question, comme par exemple, l'interférence entre la teinte et la saturation.
4. **La tâche d'analyse (opérateurs)** : Il est important de vérifier si l'indice visuel supporte l'opérateur qui manipule cet attribut. Actuellement, les couleurs discrètes (teintes) peuvent supporter la plupart des tâches, et elle est plus adéquate aux tâches de recherche et d'évaluation.

	Quantitative	Ordonnée	Qualitative
Position	0	0	0
Longueur	1	7	8
Angle	2	8	9
Orientation	3	9	10
Surface	4	10	11
Volume	5	11	12
Valeur	6	1	5
Saturation	7	2	6
Teinte	8	3	1
Texture	N	4	2
Connexion	N	5	3
Inclusion	N	6	4
Forme	N	N	7

Tableau 5.5 – Classification des indices visuels selon l'échelle d'expressivité. Les indices visuels qui ne sont pas classifiés sont ceux qui ne peuvent pas être utilisés pour exprimer ce type de donnée [29].

5.5.2.3 Le mécanisme de résolution à base de contraintes

Résoudre un VCSP (*Valued Constraint Satisfaction Problems*) revient à trouver une affectation complète des variables qui soit de valuation minimale (qui minimise le coût de violation des contraintes).

Les méthodes de recherche complètes (basées sur des parcours arborescents) sont souvent utilisées pour obtenir des solutions optimales et effectuer des preuves d'optimalité. Mais, du fait de leur comportement (souvent) exponentiel, ces méthodes peuvent s'avérer trop gourmandes en temps de calcul.

Grâce à leur façon opportuniste d'explorer l'espace de recherche, les méthodes approchées, basées sur les recherches locales (comme le recuit simulé ou les recherches tabou), sont supposées produire de bonnes solutions en des temps de calcul abordables. Malheureusement, de telles méthodes ne sont pas toujours capables de se sortir facilement d'optima locaux et peuvent perdre beaucoup de temps dans l'exploration de voisinages inintéressants.

Nous proposons d'utiliser des algorithmes hybrides, qui offrent un réel compromis entre ces deux types d'approches. Plus précisément, ils sont capables de combiner efficacement les avantages de la propagation de contraintes (associée généralement aux méthodes de recherche complète) avec l'exploration opportuniste des méthodes de recherche locale. Nous proposons un algorithme hybride, où la recherche complète et la recherche locale sont étroitement liées durant la résolution.

La recherche tabou est une méthode qui fait appel à des concepts et mécanismes généraux pour exécuter la recherche dans l'ensemble des configurations de manière plus intelligente. La recherche tabou examine un échantillon de configurations du voisinage (potentiellement toutes) et réalise le mouvement en direction de la meilleure configuration de cet échantillon. Ce mouvement peut conduire à dégrader la fonction de coût, ce qui permet de ne pas arrêter la recherche sur le premier optimum local rencontré.

Dans notre algorithme (voir Tableau 5.6), la recherche locale est définie selon trois notions : les échantillons qui sont des points représentatifs de l'espace de recherche, un voisinage qui détermine comment se déplacer d'un échantillon vers un autre et une fonction d'évaluation qui permet d'estimer la qualité des échantillons. La recherche locale explore alors l'espace de recherche en se déplaçant d'échantillon en échantillon afin d'atteindre un optimum.

De plus, notre algorithme diffère des méthodes classiques (recherche tabou) par la taille des voisinages qu'elle explore, taille qui varie au cours de la résolution. L'algo-

```
Stratégie de recherche : ( p_i : Problème, Size : Entier)
1 : size ⟵ Size
2 : s* ⟵ Solution-initiale (p_i)
3 : s ⟵ s*
4 : c ⟵ Coût(s*)
5 : for (i ⟵ 1 à MAXMOVES)
6 :     rel ⟵ Relaxer(s, size) // relaxer solution
7 :     s ⟵ Reconstruire(s, rel) // reconstruire solution
8 :     if(Coût(s) < Coût(s*))
9 :       s* ⟵ s
10 :    end if
11 :    size ⟵ Contrôler-taille-voisinage(size)
7 : end for Retourner s*
```

Tableau 5.6 – Stratégie de recherche d'une solution à un problème VCSP.

rithme part d'une solution initiale s^* ; lors de chaque mouvement, il relaxe une partie de la solution courante s, puis la reconstruit en sélectionnant la meilleure solution voisine de s. L'algorithme s'arrête dès que le nombre maximum de mouvements locaux autorisés (MAXMOVES) est atteint. Le Tableau 5.6 présente notre algorithme de stratégie de recherche qui consiste en quatre fonctions :

- **reconstruire une solution** : À chaque itération, la méthode tabou remplace la configuration courante par une nouvelle obtenue par une transformation locale appelée un mouvement. Un mouvement, noté $mvt(x_i, v_i)$, consiste a remplacer l'ancienne valeur de la variable x_i par la nouvelle valeur v_i. Le voisinage $N(s)$ de la configuration courante s est l'ensemble de toutes les configurations pouvant être atteintes par l'application d'un mouvement.

- **relaxer une solution** : Cette fonction sélectionne l'ensemble v de toutes les variables en conflit (variables apparaissant dans les contraintes violées par la solution (instanciation) courante s). Pour constituer v, l'ensemble des variables à désinstancier, un tirage aléatoire est réalisé sur l'ensemble v, jusqu'à ce que le nombre

requis de variables à désinstancier soit atteint.

- **la fonction d'évaluation du voisinage** : Les configurations du voisinage étant partielles, le meilleur mouvement à effectuer pour avancer dans la recherche de solutions est une affectation sans réparations. Par conséquent, la fonction d'évaluation (ou fonction de coût) qui permet de déterminer si une configuration est meilleure qu'une autre est simplement le coût de violation des contraintes.

- **gestion de la taille des voisinages** : Au départ, la taille du voisinage (*size*) est initialisée à une valeur minimale. Pour contrôler la taille du voisinage, plusieurs stratégies ont été implémentées. La meilleure stratégie, que nous ayons trouvée, consiste à augmenter systématiquement la taille du voisinage d'une unité à chaque fois que notre méthode ne parvient plus à améliorer la solution courante.

5.6 Étude de cas : tâche de détection des anomalies de conception

Dans cette section, nous présentons un exemple sur une tâche d'analyse de détection du *blob*, une anomalie bien connue dans la communauté du logiciel [4].

Le *blob* consiste en une classe qui monopolise tout le traitement du programme, tandis que les autres classes servent principalement à encapsuler des données. Il est caractérisé par un diagramme de classes composé d'une classe unique, complexe et non cohésive *controller class*, associée à des classes de données simples.

5.7 Description de la tâche d'analyse

La première étape est d'exprimer la détection du *blob* par une séquence d'actions à accomplir, comme décrit par le modèle de tâches d'analyse. La spécification de la tâche de détection du *blob* est présentée dans le Tableau 5.7.

Cette description est générée à l'aide de notre éditeur (voir Figure 5.4) qui fournit à l'utilisateur la liste des métriques logicielles disponibles ainsi que les différents opé-

rateurs de notre langage. Cet éditeur permet aussi à l'utilisateur de sauvegarder ou de charger la description de sa tâche sous format de fichier *XML*.

Method to accomplish Goal : *Blob* Detection
1 : Find Extremum(whole system, WMC)
2 : FOR EACH(c, selected classes)
3 : IF NOT(Verify Value(c, $LCOM5$, $HIGH$)) CONTINUE
4 : IF NOT(Verify Value(c, DIT, LOW)) CONTINUE
5 : Inspect(c, class name, method properties)
6 : Decide(the inspection confirmed that c is a controller class, Save(c, controller class))
7 : END FOR EACH

Method to accomplish Goal : Data Class Verification
1 : FOR EACH(m, controller classes)
2 : Relationship Filter(m, association out)
3 : Locate(associated classes, $WMC = LOW$, $LCOM5 = LOW$, $DIT = LOW$)
4 : Decide(located classes are too numerous, Save(m, Blob))
5 : END FOR EACH

Tableau 5.7 – Détection du *blob* exprimée par le modèle de tâches d'analyse.

5.8 Génération des tâches interactives

Une fois que la détection du *blob* a été spécifiée utilisant le modèle de tâche d'analyse, un scénario spécifique de VERSO est produit. Puisque chaque attribut impliqué dans la tâche d'analyse est employé par au moins un opérateur global, les attributs visuels disponibles sont la couleur, la hauteur et l'orientation. Le *mapping* d'attributs le plus approprié est donné dans le Tableau 5.8, basé sur les règles de perception visuelle. Quant aux opérateurs, ils sont transformés en suivant les substitutions présentées dans le Tableau 5.3.

Le scénario généré est présenté dans le Tableau 5.9. Bien que le scénario de VERSO contienne plus d'instructions que la description de départ (voir le Tableau 5.7), nous

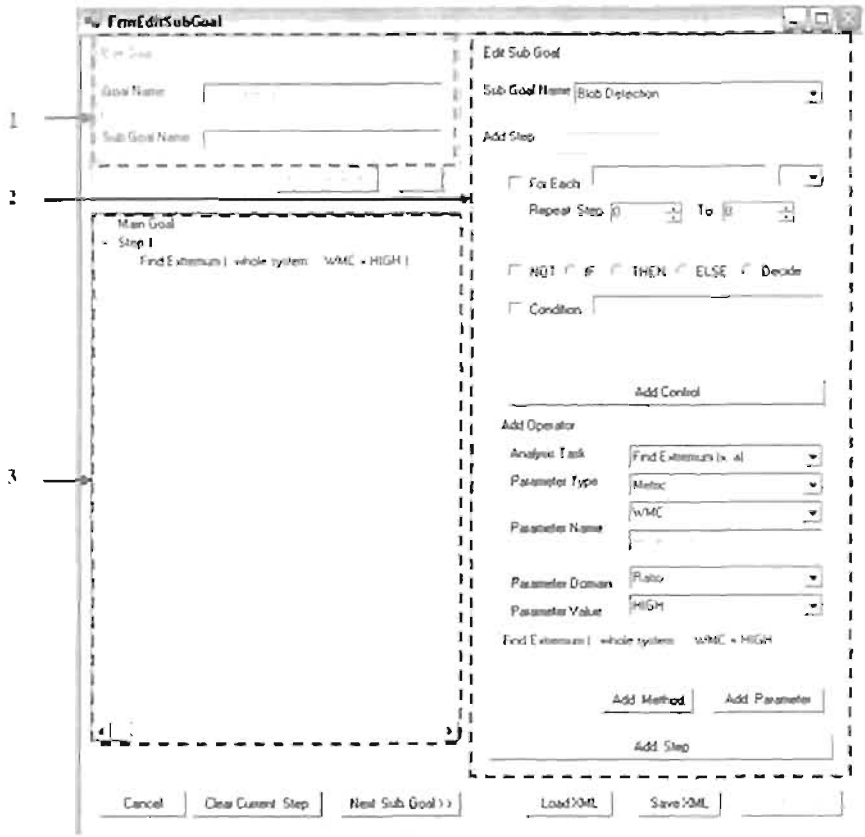

Figure 5.4 – Présentation de l'interface de l'éditeur de tâches d'analyses. (1) Édition de la liste des buts à accomplir. (2) Ajout d'un opérateur ou d'une structure de contrôle. (3) Une vue sur la description de la tâche d'analyse entière.

avons gardé la même numérotation pour montrer de quelle façon les opérateurs de la description ont été transformés en séquences d'actions dans le scénario. La Figure 5.5 présente l'interface graphique du générateur de tâches interactives.

Mapping	Coût
DIT = Couleur	3
WMC = Hauteur	1
LCOM5 = Orientation	2
Total	6

Tableau 5.8 – *Mapping* d'attributs généré de coût minimal pour la détection du *blob*.

Attribute Mapping
DIT ↔ color
WMC ↔ height
LCOM5 ↔ twist

Method to accomplish Goal : *Blob* Detection
1 : Apply the `Zoom-Out` if necessary
  Apply the `Statistical Filter` on *WMC* for whole system
  Using the `Iterator`, `Select` the classes as long as *WMC* is judged high enough
  Remove the `Statistical Filter`
2 : FOR EACH *c* in selected classes
3 : IF NOT(verify that twist is HIGH for *c*) CONTINUE
4 : IF NOT(verify that color is BLUE for *c*) CONTINUE
5 : Apply the `Zoom-In` if necessary
  `Display source code` of *c*
  Inspect class name
  Inspect method signatures
6 : If the inspection confirmed that *c* is a controller class,
  `Mark` *c* as controller class.
7 : END FOR EACH

Method to accomplish Goal : Data Class Verification
1 : FOR EACH *m* in controller classes
2 : Apply the `Association-Out Filter` on *m*
3 : Among the associated classes, locate classes whose height is LOW, twist is LOW, color is BLUE
4 : If located classes are too numerous, `mark` *m* as Blob
5 : END FOR EACH

Tableau 5.9 – Scénario de détection du *blob* spécifique à VERSO.

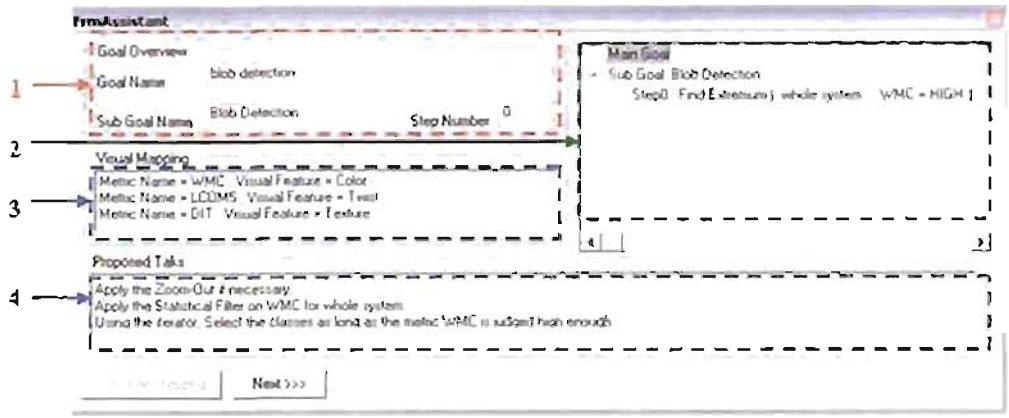

Figure 5.5 – Présentation de l'interface du générateur de tâches interactives. (1) Affichage du nom de but à accomplir. (2) Affichage de la description détaillée de l'opérateur en cours d'exécution. (3) Transformation des attributs en indices visuels. (4) Génération de scénario de tâches interactives

5.8.1 Application du scénario de détection du *blob*

Cette section présente comment le scénario de détection du blob généré dans la Section 5.8 peut être employé par un analyste en utilisant VERSO.

La première instruction à l'étape 1 est d'appliquer un zoom-out au besoin. Il s'agit de s'assurer que l'utilisateur a un point de vue global du système avant d'appliquer le filtre statistique. Une vue d'ensemble de *PCGEN* est illustrée sur la partie gauche de la Figure 5.6. Le filtre statistique est ensuite appliqué sur le système entier pour la métrique WMC. Les classes ayant des valeurs extrêmement élevées pour cette métrique sont colorées en rouge. Le reste des classes sont colorées en jaune, vert ou bleu, si leur valeur de WMC est haute, normale, ou basse, respectivement. Dans notre cas, nous sommes intéressés par les classes extrêmement élevées. L'itérateur est employé pour visiter et sélectionner les classes contrôleurs candidates. Une étape de ce processus est illustrée sur la partie droite de la Figure 5.6, où la classe encerclée en noir a été choisie comme candidat de classe contrôleur.

Une fois que tous les candidats de classes contrôleurs ont été choisis, le filtre statistique est enlevé, et les classes reprennent leur couleur originale (variant de bleu à rouge

selon la valeur de DIT). L'analyste vérifie ensuite l'orientation et la couleur de chaque classe sélectionnée. Si l'orientation n'est pas élevée ou la couleur n'est pas bleue, l'analyste passe à la classe suivante (la profondeur et la cohésion sont trop élevés pour que cette classe soit considérée comme une classe contrôleur). Si une classe sélectionnée est bleue et tournée, l'analyste peut inspecter son code source pour vérifier le nom de la classe et les signatures de méthodes. Si l'inspection confirme que la classe sélectionnée est une classe contrôleur, elle doit être marquée comme *blob*.

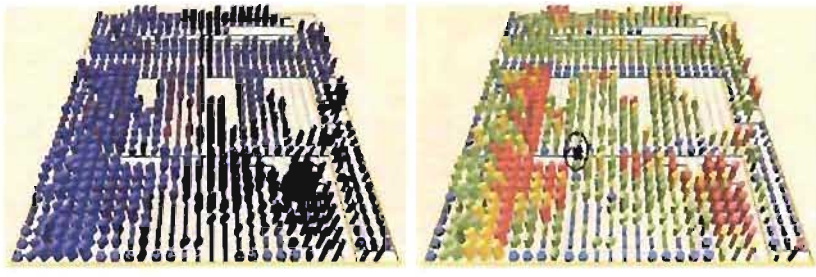

Figure 5.6 – Un exemple de *blob* découvert dans *PCGEN*. (Gauche) Opération zoom-out appliquée sur le logiciel *PCGEN* (Droite) Le filtre statistique est appliqué pour la métrique WMC. La classe encerclée en noir a été sélectionnée comme classe contrôleur.

Pour chaque classe contrôleur, l'analyste doit maintenant considérer les classes associées. Le filtre d'association sortante est ainsi appliqué pour chaque classe contrôleur. Cette opération est illustrée sur la partie gauche de la Figure 5.7. Les classes associées à la classe contrôleur gardent leur couleur originale, alors que les autres classes subissent une baisse de saturation. Une fois que le filtre est appliqué, l'analyste doit localiser, parmi les classes associées, les classes qui sont petites, droites et bleues. En conclusion, si ces classes sont nombreuses, la classe contrôleur est marquée comme étant une occurrence de *blob*. La partie droite de la Figure 5.7 montre qu'un bon nombre de classes associées sont petites, droites et bleues. Ainsi, la classe contrôleur a été marquée comme *blob*.

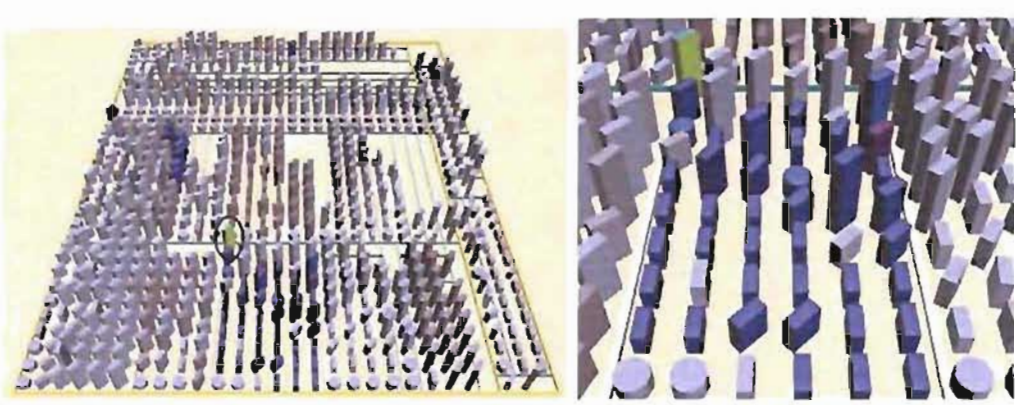

Figure 5.7 – Exemple de *Blob* détecté dans *PCGEN*. (Gauche) Le filtre d'association sortante est appliqué sur la classe contrôleur encerclée en noir. Classes associées à la classe contrôleur gardent leur couleur originale, le reste des classes du système ont eu une baisse de saturation. (Droite) Parmi les classe associées, celles qui sont petites (*height* = *LOW*), droites (*twist* = *LOW*), et bleue (color = BLUE) sont localisées.

CHAPITRE 6

CONCLUSION

Ce mémoire s'est intéressé à développer une extension de VERSO, un outil de visualisation qui représente un logiciel en trois dimensions dans le but d'en évaluer la qualité. Lorsqu'a débuté ce travail, l'un de nos objectifs était d'améliorer la visualisation de données multidimensionnelles. Pour être capable de définir un nouveau domaine d'expression, il faut évidemment s'intéresser à ce que l'observateur est en mesure de percevoir dans une représentation graphique. C'est pourquoi nous avons étudié le système visuel humain et les règles de constructions graphiques. Cependant, la visualisation ne doit pas être dissociée d'une éventuelle possibilité d'interaction avec l'information via la représentation produite. Cette analyse nous a, de fait, conduit à considérer un cadre d'étude beaucoup plus large, englobant aussi les travaux réalisés autour des taxonomies de tâches d'analyse et d'interactions.

Ce mémoire est donc organisée en deux parties. L'une est consacrée au raffinement du niveau de détails pour visualiser des données multidimensionnelles. Et l'autre est consacrée au développement d'un système d'aide à la visualisation, pour assister les utilisateurs durant leurs tâches d'analyse. Dans ce qui suit, nous présentons un bilan et des perspectives pour chacune de ces deux parties.

6.1 Contributions

6.1.1 La visualisation multidimensionnelle

L'étude de l'outil de visualisation VERSO, présentée dans le Chapitre 3, a montré que quelque soit le type d'information recherché lors de l'analyse d'un système informatique orienté objet, trois problèmes récurrents à ce domaine ressortent fréquemment, soient la quantité, la dimensionnalité et la granularité des données récoltées.

Dans le cadre du Chapitre 4, nous nous sommes intéressés tout particulièrement à la visualisation de données multidimensionnelles et au raffinement du niveau de granula-

rité. Nous avons montré tout au long du Chapitre 2 l'importance de la prise en compte des aspects cognitifs lors de l'étude des aspects informatiques de la visualisation. Nous avons tiré de cette analyse des facteurs essentiels à la construction d'une visualisation efficace. Ainsi, une bonne visualisation doit fournir une vue d'ensemble du système afin d'aider l'utilisateur à en avoir une compréhension générale. Il est néanmoins indispensable de pouvoir afficher les détails du système, et ce de manière interactive, pour que l'utilisateur utilise les données selon ses besoins. De plus, les visualisations dynamiques permettent de s'adapter aux besoins des utilisateurs en temps réel.

Nous avons proposé d'utiliser les textures pour affiner dynamiquement l'affichage des détails. Nous avons défini quatre types de textures :

- la texture de roches est définie comme une variation de la quantité et de la taille des roches sur une surface.
- la texture de briques est définie comme une variation de l'orientation des briques qui peut être horizontale, diagonale et verticale.
- la texture des fenêtres est définie par une variation du nombre des fenêtres.
- la texture de l'horloge est définie par une variation de l'orientation de l'aiguille de l'horloge.

L'utilisation de métaphores rattache les images observées à l'expérience du monde réel et exploite ainsi les connaissances de l'utilisateur. Nous avons défini pour chacune de ces textures leurs propriétés perceptives, qui peuvent nous servir comme un guide pour le choix de la texture la plus efficace pour visualiser une métrique donnée. Nous avons défini trois critères pour évaluer l'efficacité des textures. Une texture doit être intrinsèque, discriminante et précise. Nos expérimentations ont montré que la texture de roches n'est ni intrinsèque ni discriminante, car sa perception varie par le changement du point de vue de l'utilisateur et elle ne permet pas d'estimer avec précision la valeur de la métrique qui lui est associée. Nous estimons que ces deux critères sont importants ; c'est la raison pour laquelle nous avons choisi la texture de briques pour visualiser des données qualitatives, la texture de fenêtres pour visualiser des données discrètes et la texture de l'horloge pour visualiser les données continues.

Nous avons proposé d'utiliser la transparence pour affiner le niveau de granularité.

Ceci nous permet de visualiser la structure interne des représentations graphiques des entités logicielles (classes) sans avoir à changer le contexte. Nous avons proposé de visualiser la structure interne (méthodes), soit par un segment de droite de taille (longueur) proportionnelle à la métrique, soit une zone sur la surface du plancher de la boîte dont la variation du dégradé de couleur est associée à la métrique.

6.1.2 Système d'aide à la visualisation

La deuxième contribution de ce mémoire est de fournir un système d'aide qui assiste les utilisateurs durant tout le processus de la visualisation [11, 17]. Nous avons proposé une approche qui couvre toutes les étapes du processus de visualisation, de la description de tâches d'analyse jusqu'à la génération des séquences d'actions ou d'interactions que l'utilisateur doit exécuter pour atteindre son objectif.

Nous avons défini un langage de haut niveau qui permet à l'utilisateur de décrire sa tâche d'analyse. Ce langage est basé sur une taxonomie de tâches élémentaires qu'un utilisateur pourrait accomplir dans un environnement visuel. Cette description conduit le système d'aide à choisir à la fois les associations (*mapping*) entre les données et les indices visuels, et également à choisir les moyens d'interaction qui seront mis en œuvre pour l'accomplissement de la tâche.

Nous avons décrit les modes d'interaction par un modèle d'interaction qui se base sur la taxonomie de navigation élaborée par Shneiderman [36]. Le rôle du système d'aide est de faire la transformation d'une tâche d'analyse qui est conforme au modèle d'analyse en une suite d'interactions offertes par l'outil de visualisation qui représente une instance du modèle d'interaction. Cette transformation inclut aussi le *mapping* entre les données manipulées par cette tâche d'analyse et les représentations graphiques disponibles dans l'outil de visualisation.

Nous avons présenté dans le Chapitre 2 plusieurs travaux [2, 6, 15, 18, 19, 29, 45] qui ont montré comment exploiter, de façon intuitive ou ad hoc, les caractéristiques de perception visuelle afin de construire des visualisations efficaces. L'objectif de notre système d'aide est d'exploiter ces caractéristiques de façon plus systématique.

La modélisation du problème d'évaluation de l'efficacité d'une représentation gra-

phique par un problème d'optimisation avec contraintes (modélisés sous forme de VCSP) s'avère souvent plus efficace. La base des contraintes regroupe l'ensemble des règles qui évaluent l'efficacité des représentations graphiques par rapport à un contexte particulier. Ces règles se basent sur trois aspects fondamentaux : les caractéristiques des données visualisées, la nature de la tâche de l'utilisateur et la perception de ces représentations graphiques par l'utilisateur.

6.2 Perspectives

6.2.1 La visualisation multidimensionnelle

La représentation de systèmes complexes sous forme de mondes virtuels habités peut aider les utilisateurs à travailler collectivement. Les techniques de réalité virtuelle incluent l'interaction et l'utilisation de plusieurs niveaux de détails. Le niveau de détails consiste à n'afficher des informations précises sur un élément que si l'utilisateur en est proche. L'immersion totale dans des mondes virtuels implique davantage les utilisateurs dans la visualisation. Ils peuvent alors explorer le monde et interagir avec les données. Cependant, ils risquent de se perdre dans le monde virtuel. Afin d'éviter ceci, il est souhaitable de proposer aussi des chemins de navigation prédéfinis.

Les techniques de réalité virtuelle facilitent la navigation parmi les données et la visualisation de nombreuses dimensions, en faisant référence à des environnements familiers aux utilisateurs. Cependant, le choix de métaphores adaptées passe par la compréhension des données et ce qu'elles recèlent.

Nous avons étudié dans ce projet la perception de certaines composantes dans l'espace à trois dimensions comme la couleur, la transparence et la taille. Il reste à étudier les facteurs déterminants pour la perception de la troisième dimension. Faut-il toujours rechercher un plus grand réalisme du rendu de la scène tridimensionnelle ou au contraire existe-t-il des modes de rendu qui privilégient la transmission de l'information au travers de la troisième dimension ? Les réponses qui seront apportées à ces différentes questions mèneront peut être à la spécification de moteurs de rendu adaptés à la visualisation d'informations plus qu'à la réalité virtuelle.

6.2.2 Système d'aide à la visualisation

Notre modèle d'interaction est basé sur la taxonomie de Shneiderman [36]. Cette dernière fait l'hypothèse qu'une grande quantité d'information est constituée d'un grand nombre d'éléments où chaque élément possède des attributs variés. Shneiderman se restreint ainsi : "*I assume that users are viewing collections of items, where items have multiple attributes*". Il nous semble cependant intéressant de compléter cette taxonomie en définissant d'autres types de données. La classification proposée par Tweedie [41] est sans aucun doute une voie dans cette direction. La taxonomie de Tweedie fait l'hypothèse que les données à visualiser peuvent se décomposer en deux catégories : les valeurs et les structures. Nous considérons qu'une grande quantité d'information repose sur une structure générale où chaque élément est une valeur ou une petite structure contenant des valeurs. Ensuite, pour chaque type de données, nous proposons de définir l'ensemble des techniques d'interaction (en termes d'action et de conséquences [41]) potentiellement envisageables.

Un autre axe de recherche est notamment l'évaluation de l'efficacité des représentations graphiques. Il serait en particulier intéressant d'enrichir notre base de contraintes par des règles qui prennent en compte la distribution de données [20].

BIBLIOGRAPHIE

[1] Robert A. Amar, James Eagan et John T. Stasko. Low-level components of analytic activity in information visualization. Dans *INFOVIS*, page 15, 2005.

[2] Jacques Bertin. *La graphique et le traitement graphique de l'information.* Nouvelle Bibliothèque scientifique. Flammarion, 1977.

[3] Roland Bertuli. *Compréhension de systèmes orientés-objet par l'utilisation d'informations dynamiques condensées.* Mémoire de DEA, École Supérieure en Sciences Informatiques, Sophia-Antipolis, France, 2003.

[4] William J. Brown, Raphael C. Malveau, III Hays W. McCormick et Thomas J. Mowbray. *AntiPatterns : Refactoring Software, Architectures, and Projects in Crisis.* John Wiley Press, 1998.

[5] Christophe Bruley. *Analyse des représentations graphiques de l'information - Extension aux représentations tridimenionnelles-.* Thèse de Doctorat, Université Joseph Fourier, 1999.

[6] Tara C. Callaghan. Interference and dominance in texture segregation : Hue, geometric form, and line orientation. *Perception and Psychopysics*, 46(4):299–311, 1989.

[7] S. R. Chidamber et C. F. Kemerer. A metric suite for object-oriented design. *IEEE Transactions on Software Engineering*, 20(6), June 1994.

[8] T. A. Corbi. Program understanding : challenge for the 1990's. *IBM Syst. J.*, 28(2): 294–306, 1989. ISSN 0018-8670.

[9] Serge Demeyer, Stéphane Ducasse et Michele Lanza. A hybrid reverse engineering approach combining metrics and program visualization. Dans *Working Conference on Reverse Engineering*, pages 175–186, 1999.

[10] Karim Dhambri. *Détection visuelle d'anomalies de conception dans les programmes orientés objet.* Mémoire de maîtrise, Département d'informatique et recherche opérationnelle, Université de Montréal, 2007.

[11] Karim Dhambri, Salima Hassaine, Houari Sahraoui et Pierre Poulin. Détection visuelle d'anomalies de conception. 14^{e} *Colloque international sur les langages et modèles à objets*, mars 2008.

[12] Stéphane Ducasse et Michele Lanza. The class blueprint : Visually supporting the understanding of classes. *IEEE Trans. Softw. Eng.*, 31(1):75–90, 2005. ISSN 0098-5589.

[13] Stephen G. Eick, Joseph L. Steffen et Eric E. Sumner Jr. Seesoft-a tool for visualizing line oriented software statistics. *IEEE Transactions on Software Engineering*, 18(11):957–968, 1992. ISSN 0098-5589.

[14] Norman E. Fenton et Shari Lawrence Pfleeger. *Software Metrics : A Rigorous and Practical Approach.* Course Technology, 1998.

[15] Wendell R. Garner. *The Processing of Information and Structure.* Lawrence Erlbaum Associates, 1975.

[16] James J. Gibson. *The Perception of the Visual World.* Houghton Mifflin, 1950.

[17] Salima Hassaine, Karim Dhambri, Houari Sahraoui et Pierre Poulin. Automatic generation of strategies for visual anomaly detection. 11th *ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering*, pages 7–18, July 2007.

[18] Christopher G. Healey, Kellogg S. Booth et James T. Enns. Visualizing real-time multivariate data using preattentive processing. *ACM Trans. Model. Comput. Simul.*, 5(3):190–221, 1995. ISSN 1049-3301.

[19] Christopher G. Healey, Kellogg S. Booth et James T. Enns. High-speed visual estimation using preattentive processing. *ACM Trans. Comput.-Hum. Interact.*, 3 (2):107–135, 1996. ISSN 1073-0516.

[20] Christopher G. Healey, Robert St.Amant et Mahmoud S. Elhaddad. Via : A perceptual visualization assistant. Dans *28th Workshop on Advanced Imagery Pattern Recognition (AIPR-99), Washington, DC.*, 1999.

[21] Danny Holten, Roel Vliegen et Jarke J. van Wijk. Visual realism for the visualization of software metrics. Dans Stéphane Ducasse, Michele Lanza, Andrian Marcus, Jonathan I. Maletic et Margaret-Anne D. Storey, éditeurs, *VISSOFT*, pages 27–32. IEEE Computer Society, 2005. ISBN 0-7803-9540-9.

[22] Brian Johnson et Ben Shneiderman. Treemaps : A space-filling approach to the visualization of hierarchical information structures. Dans *IEEE Visualization Conference*, pages 335–344, October 1991.

[23] B. Julesz. Visual pattern discrimination. *IRE Transactions on Information Theory*, 8(2):84–92, 1962.

[24] Claire Knight et Malcolm Munro. Virtual but visible software. Dans *IV '00 : Proceedings of International Conference on Information Visualisation*, pages 198–205, July 2000.

[25] Guillaume Langelier. *Visualisation de la qualité des logiciels de grandes tailles.* Mémoire de maîtrise, Département d'informatique et recherche opérationnelle, Université de Montréal, 2006.

[26] Guillaume Langelier, Houari Sahraoui et Pierre Poulin. Visualization-based analysis of quality for large-scale software systems. Dans *ASE '05 : Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 214–223, New York, NY, USA, 2005. ACM. ISBN 1-59593-993-4.

[27] Michele Lanza et Stéphane Ducasse. Polymetric views - a lightweight visual approach to reverse engineering. *IEEE Trans. Softw. Eng.*, 29(9):782–795, 2003. ISSN 0098-5589.

[28] Gerard Lommerse, Freek Nossin, Lucian Voinea et Alexandru Telea. The visual code navigator : An interactive toolset for source code investigation. Dans *INFOVIS '05 : Proceedings IEEE Symposium on Information Visualization 2005*, pages 25–32, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7803-9464-x.

[29] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, April 1986. ISSN 0730-0301.

[30] Andrian Marcus, Louis Feng et Jonathan I. Maletic. *3d* representations for software visualization. Dans *SOFTVIS*, pages 27–36, 207–208, 2003.

[31] Robert C. Martin. *Agile Software Development, Principles, Patterns, and Practices*. Prentice Hall, 2002.

[32] Jon May et Philip J. Barnard. Modelling multimodal interaction : A theory-based technique for design analysis support. Dans *INTERACT '97 : Proceedings of the IFIP TC13 Interantional Conference on Human-Computer Interaction*, pages 667–668, London, UK, UK, 1997. Chapman & Hall, Ltd. ISBN 0-412-80950-8.

[33] Thomas M. Pigoski. *Practical Software Maintenance : Best Practices for Managing Your Software Investment.* Wiley, New York, 1996.

[34] Steve P. Reiss et Manos Renieris. Chapter 11 : The bloom software visualization system. Dans Kang Zhang, éditeur, *Software Visualization : From Theory to Practice*, pages 311–358. Kluwer Academic Publishers, 2003.

[35] Hikmet Senay et Eve Ignatius. A knowledge-based system for visualization design. *j-IEEE-CGA*, 14(6):36–47, nov 1994. ISSN 0272-1716.

[36] Ben Shneiderman. The eyes have it : A task by data type taxonomy for information visualizations. *IEEE Visual Languages (UMCP-CSD CS-TR-3665)*, pages 336–343, 1996.

[37] Ian Sommerville. *Software Engineering*. Addison-Wesley, 6^{th} edition, 2000.

[38] John Stasko et Eugene Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. Dans *INFOVIS '00 : Proceedings of the IEEE Symposium on Information Vizualization 2000*, page 57, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0804-9.

[39] Maurice Termeer, Christian F. J. Lange, Alexandru Telea et Michel R. V. Chaudron. Visual exploration of combined architectural and metric information. Dans *VISSOFT*, pages 21–26, 2005.

[40] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press Limited, 1993.

[41] Lisa Tweedie. Interactive visualisation artifacts : how can abstractions inform design ? Dans *HCI '95 : Proceedings of the HCI'95 conference on People and computers X*, pages 247–265, New York, NY, USA, 1995. Cambridge University Press. ISBN 0-521-56729-7.

[42] Stephen Wehrend et Clayton Lewis. A problem-oriented classification of visualization techniques. Dans *VIS '90 : Proceedings of the 1st conference on Visualization '90*, pages 139–143, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press. ISBN 0-8186-2083-8.

[43] Richard Wettel et Michele Lanza. Visualizing software systems as cities. Dans *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*, pages 92–99, 2007.

[44] Jarke J. Van Wijk et Huub van de Wetering. Cushion treemaps : Visualization of hierarchical information. Dans *INFOVIS '99 : Proceedings of the IEEE Symposium*

on Information Visualization 1999, pages 73–78, Washington, DC, USA, 1999. IEEE Computer Society. ISBN 0-7695-0431-0.

[45] Jiajie Zhang. A representational analysis of relational information displays. *Int. J. Hum.-Comput. Stud.*, 45(1):59–74, 1996. ISSN 1071-5819.

[46] Michelle X. Zhou et Steven K. Feiner. Visual task characterization for automated visual discourse synthesis. Dans *CHI '98 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 392–399, New York, NY, USA, 1998. ACM Press/Addison-Wesley Publishing Co. ISBN 0-201-30987-4.