THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

Présentée et soutenue par :
Raphaël Hoarau
le lundi 30 septembre 2013

Titre :

Interaction et Visualisation avec des liens de dépendances

École doctorale et discipline ou spécialité :
ED MITT : Image, Information, Hypermedia

Unité de recherche :
IRIT UMR 5055

Directeur(s) de Thèse :

Stéphane Conversy, encadrant
Philippe Palanque, directeur

Jury :
Michel Beaudouin-Lafon, président et rapporteur
Patrick Girard, rapporteur
Stéphane Chatty
Pierre Dragicevic

TABLE DES MATIERES

REMERCIEMENTS

Je souhaite avant tout remercier Stéphane Conversy, sans qui je n'aurai jamais pu commencer et mener à bien ce projet de recherche. Stéphane m'a donné l'opportunité de faire cette thèse et m'a précieusement aidé dans chacune de ses étapes. J'ai pu grâce à son encadrement, apprendre beaucoup de choses et me positionner dans un domaine de l'Interaction Homme-Machine qui me tient à cœur : l'étude des paradigmes d'interaction et le design d'interaction.

Je remercie l'ENAC et plus particulièrement le Laboratoire d'Informatique Interactive pour m'avoir accueilli pendant ces quatre années. Merci à l'ensemble de mes collègues pour leur aide et implication dans ce projet ainsi que leurs encouragements, Hélène Gaspard-Boulinc, Stéphane Chatty, Christophe Hurter, Yannick Jestin, Catherine Letondal, Mathieu Magnaudet, Daniel Prun, Nicolas Saporito, Jean-Luc Vinot. Merci à Fabien André, avec qui j'ai pu avoir le plaisir de travailler depuis le Master et de partager le même bureau pendant la première moitié de la thèse. Merci à Maxime Cordeil, collègue thésard (courage pour la fin) et ami, avec qui j'ai partagé un autre bureau la seconde moitié de la thèse, sans oublier Benjamin Tissoires, qui l'a brièvement précédé. Merci aussi à Philippe Palanque pour avoir accepté la direction de la thèse et s'être montré disponible et à l'écoute, ainsi qu'aux membres de l'équipe ICS pour leur sympathie.

Je tiens également à remercier tous les utilisateurs qui ont accepté de participer à ce projet. Merci à ceux qui se sont portés volontaires pour être cobayes lors des séances d'évaluation, à ceux qui ont participé aux ateliers de conception pour leurs supers idées, et merci à ceux qui ont bien voulu accepter les interviews pour leurs précieux scénarios.

Merci au jury de thèse pour les échanges grandement intéressants qui ont eu lieu lors de l'ultime épreuve qu'est la soutenance et merci aux rapporteurs pour avoir accepté de relire le manuscrit et de m'avoir fait part de critiques qui m'ont permis de l'améliorer.

Je souhaite par ailleurs remercier mes amis thésards, Arnaud, Jérém, Rémi, Thierry (bon courage à ceux pour qui la fin approche), et mes amis non-thésards, Maroo, Ludo, Sophie, Ben, Magali, ainsi que la team Sandyclaws pour les bons moments passés, les encouragements et pour avoir été à l'écoute. Merci à mes proches, mes parents et ma famille pour m'avoir également soutenu et encouragé jusqu'à la fin. Et merci à Sara, pour avoir été à mes côtés jusqu'au bout de cette aventure.

INTRODUCTION

L'Interaction Homme-Machine est une discipline informatique dont les principales préoccupations sont les moyens mis à disposition de l'utilisateur afin qu'il puisse interagir avec un système de façon efficace, efficiente et satisfaisante. Les champs d'étude de l'Interaction Homme-Machine concernent la conception de systèmes interactifs et l'évaluation de l'utilisabilité de tels systèmes, en passant par du design d'interaction, du prototypage, de la représentation d'information (Figure 1) et d'autres disciplines connexes (Figure 2).

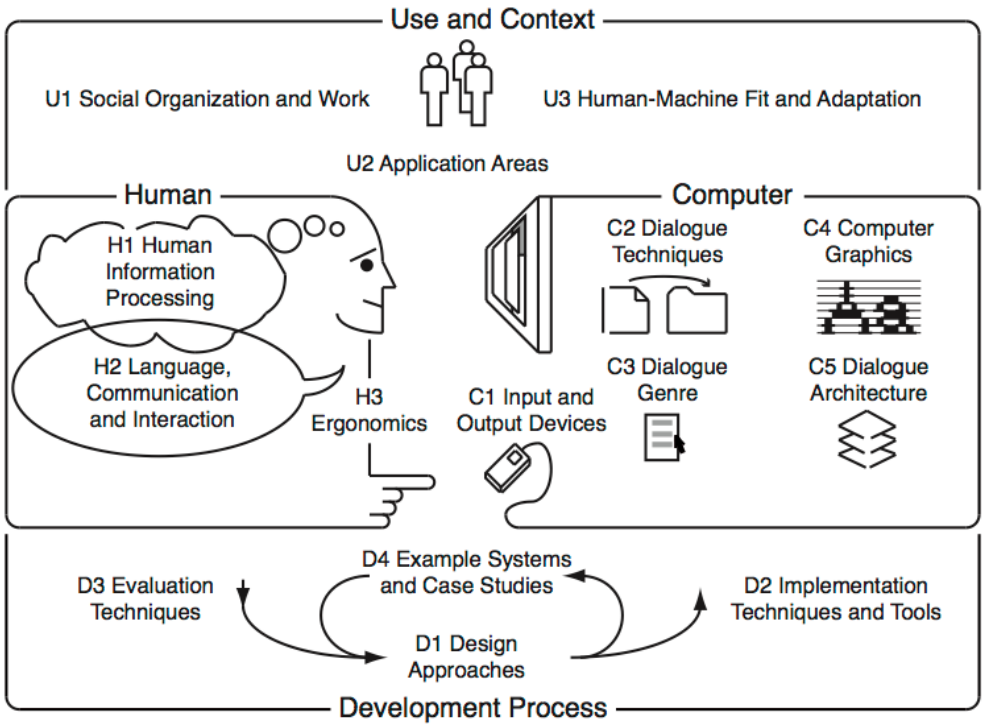


Figure 1. Champs d'étude de l'Interaction Homme-Machine (Hewett et al., 1992, fig. 1).

Dans le cadre de la thèse, nous nous intéressons à de nouveaux moyens d'interaction pouvant « augmenter » le pouvoir d'action de l'utilisateur (Cechanowicz & Gutwin, 2009), c'est-à-dire lui offrir une plus grande capacité à agir efficacement sur ses objets d'intérêts. Nous avons pour cela mené des recherches dans le design d'interaction graphique, afin de concevoir de nouvelles techniques d'interaction permettant de manipuler un grand nombre d'objets en un minimum d'actions tout en permettant la conception exploratoire.

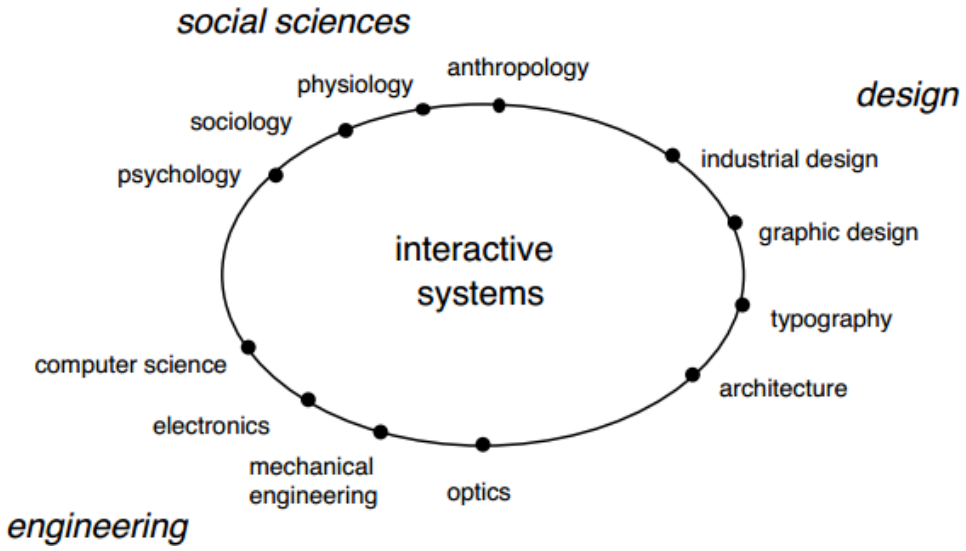


Figure 2.Disciplines connexes à l'Interaction Homme-Machine fig. (Mackay, 2003, fig. 1)

A. L'interaction graphique

L'interaction entre un utilisateur et un système est permise par le biais d'une interface, sensible aux informations saisies via des périphériques d'entrée (frappe au clavier, mouvement de souris, pression sur surface tactile) et apportant de l'information via des périphériques de sortie (écran, enceintes, retour haptique). Les première interfaces, dites interfaces textuelles ou interfaces à lignes de commandes, offraient aux utilisateurs la possibilité d'interagir avec le système en saisissant des commandes textuelles avec un clavier pour modifier l'état du système et pour percevoir son nouvel état.

En 1963, Ivan Sutherland présentait Sketchpad, un des premiers systèmes permettant l'affichage et la manipulation d'objets graphiques à l'écran (Figure 3). Pour illustrer la différence entre une interface graphique et textuelle, prenons l'exemple d'une interaction de Sketchpad comme le redimensionnement d'un objet. En utilisant une interface textuelle, l'utilisateur aurait à saisir dans sa commande les paramètres de redimensionnement de l'objet et devrait utiliser une autre commande pour percevoir le résultat. Seulement si le résultat n'est pas celui escompté (il est difficile d'estimer la nouvelle taille de l'objet au premier essai), il est contraint de réitérer cette succession de commandes jusqu'à obtenir un résultat satisfaisant. En procédant par manipulation directe sur l'objet, il peut en revanche modifier (avec une poignée de redimensionnement par exemple) la taille à souhait et percevoir immédiatement le résultat de ses actions, ce qui permet d'explorer plus aisément différentes possibilités de redimensionnement et d'ajuster précisément le résultat. Ainsi, la latence entre la réalisation d'une action et la perception de son effet est plus faible.

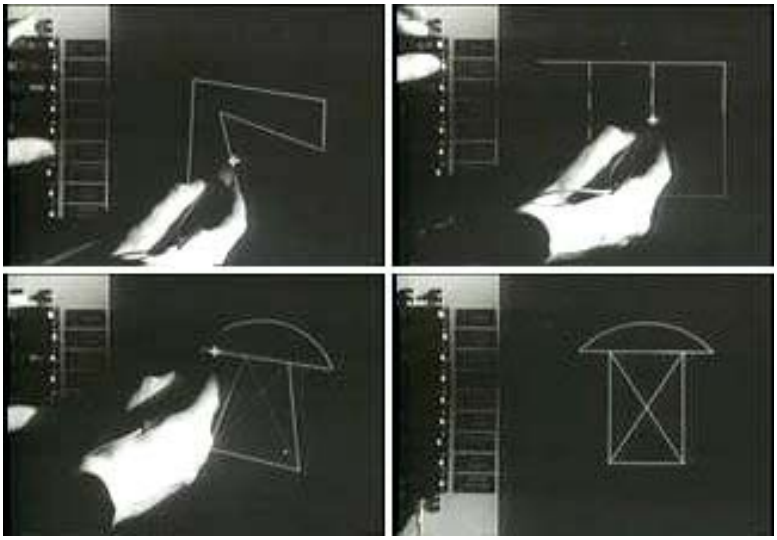


Figure 3. Démonstration de Sketchpad réalisée par Alan Kay en 1987.

De telles interfaces réduisent la distance entre le modèle conceptuel du système (ce qui définit le système, par exemple une arborescence de fichiers d'un disque dur) et le modèle mental que construit l'utilisateur (ce que l'utilisateur perçoit de l'état du système, par exemple une fenêtre contenant un ensemble d'icônes représentant des fichiers et des dossiers) (Norman, 1988). L'utilisateur peut ainsi avoir plus facilement conscience des objets qui peuvent être manipulés (puisqu'ils sont continuellement représentés) et des opérations réalisables sur ces objets. Ces interfaces sont communément qualifiées d'interfaces à manipulation directe (Shneiderman, 1987).

Notre problématique concerne plus précisément les éditeurs graphiques et les paradigmes d'interaction qui leurs sont propres. Par éditeur graphique, nous entendons tout type d'interface 2D qui présente à l'écran des données graphiques manipulables, telle que des éditeurs de dessins (Paint, InkScape, Illustrator…), des éditeurs de texte (Word, OpenOffice), des langages de programmation visuelle (AutoCAD, éditeurs de diagrammes UML, de circuits électroniques…), des outils de présentation (PowerPoint), d'édition d'agenda numérique (iCal, Google Agenda) etc.

B. Problématique

Un grand nombre d'outils permettent de manipuler des objets graphiques (par exemple changer leur couleur ou leur épaisseur de trait). Ces manipulations se font au moyen d'*interactions*, c'est-à-dire d'échanges entre utilisateur humain et ordinateur, au travers de composants logiciels et matériels. Dans cette thèse, nous nous intéressons aux interactions avec de multiples objets interdépendants. Que ce soit les outils de présentation, les traitements de texte, les logiciels de conception 2D ou 3D, ces systèmes permettent d'afficher et d'agir sur de grandes quantités d'objet liés entre eux

par des relations d'interdépendances. Nous verrons par la suite qu'il n'est pas toujours aisé de manipuler ces objets de façon efficace. Par exemple, il est souvent difficile pour un utilisateur de comprendre les relations de dépendances entre objets (problème de dépendances cachées (T. R. G. Green, 1989), ce qui rend difficile la manipulation individuelle, ou la prévision de résultats d'une interaction sur un objet particulier.

Ces interactions avec des objets multiples interdépendants ont été peu étudiées en tant que telles. Certes, il existe des composants interactifs dans les systèmes actuels qui permettent de manipuler des "styles" ou des "masters" afin d'interagir avec des ensembles, ainsi que des guides de conception et des recommandations qui édictent des règles et des propriétés pour représenter et manipuler de façon efficace des objets individuels. Mais peu de concepts ou de propriétés concernent la modification de plusieurs objets à la fois. Par exemple, quelles sont les interactions qui permettent de définir des ensembles ? Quels moyens existent-t-il pour appliquer une interaction sur plusieurs objets ? Quels sont les concepts importants pour la manipulation de telles entités ? Quel cadre conceptuel un concepteur peut-il utiliser pour concevoir des interactions qu'un utilisateur comprendra et contrôlera efficacement ?

Cette problématique se rapproche de celle des chercheurs en Interaction Homme-Machine qui rendent compte du phénomène de l'interaction à l'aide de ce que nous appelons dans ce document des paradigmes[1] d'interaction, c'est-à-dire des manières cohérentes de décrire les constituants d'une interaction et leurs relations. Par exemple, le Dialogue Conversationnel est un paradigme décrivant les interactions comme des échanges de type question-réponse entre un être humain et un ordinateur. La Manipulation Directe est un autre paradigme d'interaction reposant sur un ensemble de principes, dont celui d'opérations rapides, incrémentales, réversibles aux effets immédiats (Shneiderman, 1987), souvent associé aux interfaces de type Bureau (Figure 4). WIMP (pour Windows, Icons, Menus et Pointing devices) est un paradigme issu de la manipulation directe qui définit les interfaces graphiques comme des ensembles de fenêtres, d'icônes, de menus et de dispositifs de pointages. Le paradigme de l'Interaction Instrumentale (Beaudouin-Lafon, 2000) permet de décrire l'interaction avec des objets d'intérêts à l'aide d'instruments (par exemple, un outil pinceau pour changer la couleur d'un objet).

[1] **Paradigme** : "Représentation du monde ; manière de voir les choses ; modèle cohérent de pensée, de vision du monde qui repose sur une base définie, sur un système de valeurs." http://fr.wiktionary.org/wiki/paradigme. *Dictionnaire de l'Académie française, huitième édition, 1932-1935*.

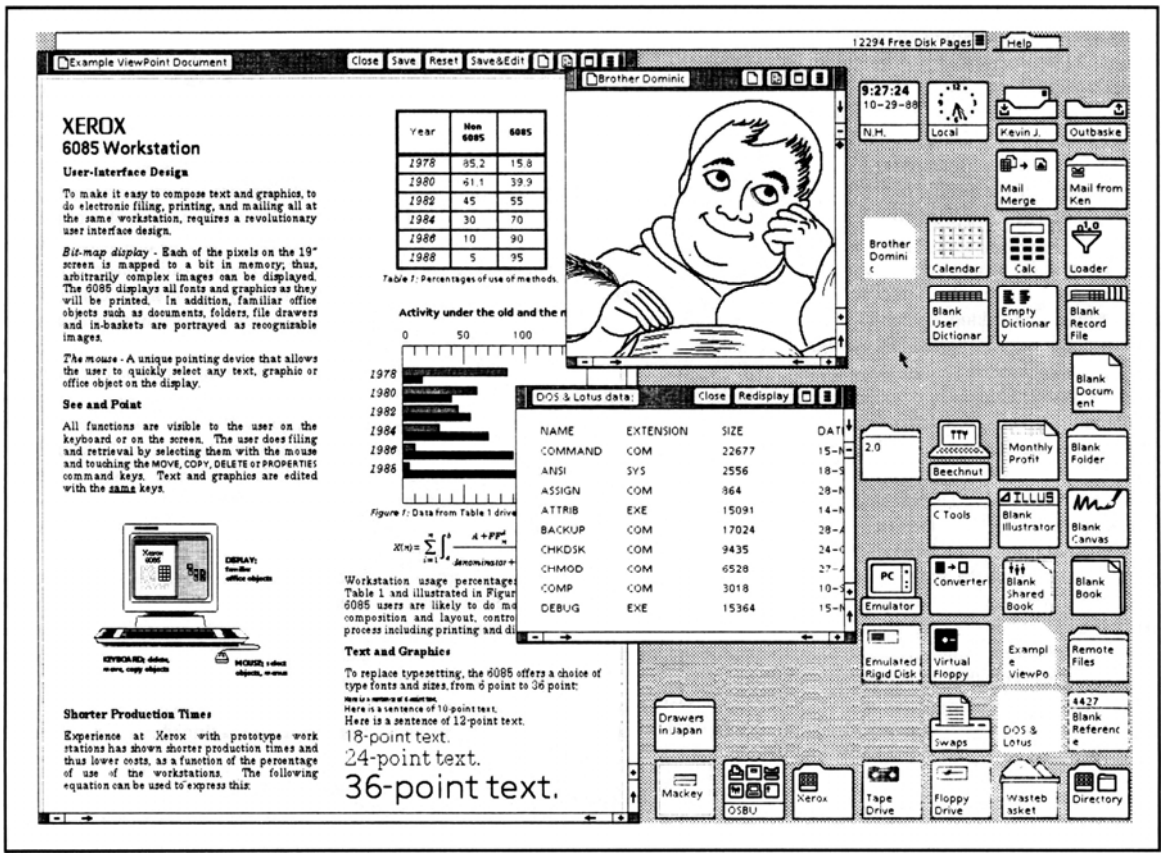


Figure 4. Interface du Xerox Star (ordinateur commercialisé en 1981) qui introduisit la métaphore du bureau et le paradigme WIMP (Johnson et al., 1989, fig. 1).

La définition de ces paradigmes est issue d'analyses d'interactions existantes, qu'elles aient été conçues par des industriels ou des chercheurs. Le besoin de définir de nouveaux paradigmes d'interaction vient notamment de la constatation qu'un paradigme particulier existant ne rend pas compte de façon satisfaisante d'une manière d'interagir. Par exemple, l'interaction instrumentale précise la nature des objets d'intérêt en considérant certains d'entre eux comme des instruments agissant sur d'autres objets d'intérêt, et possédant des propriétés spécifiques telles que le degré d'indirection temporelle (Beaudouin-Lafon, 2000).

Si la définition de nouveaux paradigmes par l'identification de nouveaux concepts et le raffinement de concepts existants est intéressante au point de vue intellectuel, elle est aussi importante pour la conception de systèmes homme-machine efficaces. En effet, ces paradigmes peuvent servir à guider la conception de systèmes interactifs : les concepteurs d'interfaces peuvent s'inspirer des concepts et des exemples issus de ces paradigmes pour concevoir de nouveaux systèmes interactifs (voire pour générer de nouvelles interactions dans le cas de l'interaction instrumentale). Si les concepts utilisés ne sont pas les bons, ou si les propriétés importantes n'ont pas été identifiées, l'interface conçue peut être problématique. Par exemple, bien que les interfaces WIMP

soient issues des interfaces à manipulation directe, elles se sont éloignées des principes de la manipulation directe en introduisant des interactions usant de fenêtres modales et de menus, qui s'apparentent plus à des interactions à base de commandes. En effet, les opérations ne sont exécutées qu'à la fermeture de la fenêtre, ce qui ne respecte pas le principe de la manipulation directe qui stipule que les opérations doivent êtres rapides, incrémentales et réversibles, et dont les effets sur les objets doivent être visibles immédiatement. Le paradigme d'interaction instrumentale permet de requalifier la boîte de dialogue modale en objet d'intérêt (un instrument), qui à ce titre devrait bénéficier des mêmes considérations que les objets d'intérêt de la manipulation directe, et donc respecter le principe d'action à effet immédiat.

Nous considérons que les paradigmes d'interaction existants ne rendent pas compte de façon satisfaisante des interactions avec de multiples objets interdépendants. Comme nous le verrons plus loin dans ce document, nous avons identifié que ces interactions concernent les *structures* d'objets plus que les objets eux-mêmes. Les paradigmes existants ne décrivent pas ce type d'interaction et ce type d'objet d'intérêt. Nous avons défini des besoins et des exigences pour rendre ces interactions efficaces, et nous avons conçu un ensemble d'interactions respectant ces besoins et exigences. L'ensemble de ce travail nous a conduit à l'ébauche d'un nouveau paradigme d'interaction : « l'interaction structurelle », qui étend les paradigmes de la manipulation directe et de l'interaction instrumentale.

C. Démarche de recherche

Le but principal de nos recherches est la production de connaissances sur l'interaction avec de multiples objets. Afin de mener à bien nos travaux, nous avons suivi une démarche dite « design-oriented research » (recherche orientée design) (Fallman, 2003). Plus précisément, nous voulions favoriser la co-construction et la co-évolution de la compréhension des problèmes et des solutions.

Notre processus de recherche s'appuie sur des principes propres à la conception participative (Muller & Kuhn, 1993). Comme la problématique de recherche est liée à des problèmes d'utilisabilité des éditeurs existants, nous avons mené des enquêtes contextuelles auprès d'utilisateurs d'éditeurs graphiques. Analyser l'activité des utilisateurs tandis qu'ils sont confrontés au problème exposé est un moyen permettant de mieux définir notre problématique et d'identifier des besoins concernant la manipulation synchrone de nombreux objets. Nous avons analysé l'état de l'art relatif aux problématiques révélées par les enquêtes contextuelles.

Apporter des solutions à la problématique nécessitait de concevoir des interactions. Aussi avons-nous organisé des ateliers de conception lors desquels nous procédions à des séances de générations d'idées (« brainstorming ») et de conception de prototypes interactifs filmés afin d'élaborer des ébauches de solutions possibles. Nous avons implémenté des prototypes de ces interactions, et procédé à des évaluations de ces interactions afin d'en mesurer l'intérêt.

D. Organisation du mémoire

Dans l'introduction, nous avons présenté le contexte dans lequel se situent nos travaux de recherche et avons défini la problématique. Nous avons également présenté notre démarche de recherche. La suite du manuscrit est organisée en quatre parties :

- La première partie présente trois scénarios de travails issus d'enquêtes contextuelles qui nous ont permis d'identifier un ensemble de besoins (concernant l'utilisation d'éditeurs graphiques) ayant orienté nos recherches.
- La deuxième partie est un état de l'art dans lequel nous étudions les activités des utilisateurs concernées par notre problématique. Nous explorons également les paradigmes d'interaction et les techniques existantes qui permettent l'interaction avec des objets multiples.
- Nous présentons dans la partie suivante des principes de structuration illustrés par des outils interactifs et un ensemble d'exigences propres à l'interaction avec des structures.
- Pour finir, la dernière partie décrit le processus d'évaluation mené auprès d'utilisateurs et présente les conclusions issues de ces évaluations.

ENQUÊTES CONTEXTUELLES ET ANALYSE DES BESOINS

Les premières étapes du travail de thèse nécessitaient d'explorer le problème afin d'identifier les besoins des utilisateurs et comment y répondre. Afin d'étudier des cas à la fois concrets et réalistes, des entretiens en contexte ont été menés auprès de plusieurs concepteurs, ce dernier terme étant à prendre au sens large : création graphique (avec des outils comme Illustrator), emploi du temps pour une formation (iCal), de plan de site géographique (AutoCAD), ou de support de cours (PowerPoint). De ces cas d'études ont été dérivés un total de treize scénarios de travail, permettant la mise en évidence de problèmes et servant de support à la conception de solutions.

L'utilisation de ces scénarios de travail s'inscrit dans le cadre d'un processus de conception participative. Il s'agit d'une étape préalable qui se décrit par l'observation et l'analyse de l'activité d'un utilisateur en temps réel, et dont les bénéfices sont les suivants : mettre en évidence les problèmes identifiés et des besoins spécifiques, obtenir des supports à la conception de solutions. Nous en présentons trois d'entre eux (d'autres scénarios sont également présentés en annexes de ce document page 155).

A. Processus

Lors de chaque rencontre avec les utilisateurs, nous commencions par nous présenter et par décrire la problématique au cœur de la thèse concernant les dépendances et leur visualisation. Nous leur expliquions ensuite le déroulement de la séance. Il leur était demandé de nous présenter leur activité, et de nous permettre de les observer tandis qu'ils utilisaient leurs outils. Afin de comprendre réellement leur intentions, nous leur demandions de procéder en « think aloud » (Anders & Simon, 1980), c'est-à-dire de dire à haute voix ce qu'ils faisaient ou bien essayaient de faire. En fin de séance nous faisions un bilan de celle-ci afin de s'assurer que nous avions bien saisi ce qu'ils faisaient et les problèmes rencontrés.

B. Scénarios

Nous présentons trois scénarios qui mettent en avant un certain nombre d'exigences relatives aux interactions sur plusieurs objets, avec ou sans structure. Les scénarios sont issus de situations réelles, mais légèrement adaptés à des fins d'illustration : certaines

interactions considérées comme impossibles (avec par exemple Inkscape) pourraient l'être avec d'autres outils (par exemple Illustrator, et vice-versa). Certaines étapes sont annotées ainsi : (*étape*). Les annotations sont les suivantes, nous les détaillons plus en détail dans le bilan des scénarios où l'on présente les besoins identifiés :

- (*Désignation*) : L'utilisateur désigne une action à réaliser, ou une cible de cette action.
- (*Structuration*) : L'utilisateur créé ou interagit avec une structure (par exemple un groupe d'objets).
- (*Pouvoir d'action*) : L'utilisateur souhaite réaliser une interaction sur plusieurs objets à la fois.
- (*Recherche*) : L'utilisateur souhaite désigner un ou plusieurs objets selon des critères spécifiques qu'il doit trouver dans son espace de travail.
- (*Exploration*) : L'utilisateur explore des solutions sans avoir d'idées précises concernant le résultat à obtenir.

1. Réalisation d'un clavier virtuel sous Illustrator

Élodie est designer graphique. Sa tâche courante consiste en la réalisation d'un clavier virtuel. A l'aide d'un éditeur graphique, elle commence par créer la première touche de ce clavier. Pour ce faire, elle dessine un rectangle blanc aux coins arrondis avec un contour noir. Elle ajoute ensuite un second rectangle dans le précédent, avec un gradient bleu et sans contour. Élodie sélectionne ces deux rectangles à l'aide d'un lasso de sélection (*Désignation*) et les regroupe avec une commande de menu permettant de créer des groupes d'objets (*Structuration*). Après cela, elle ajoute un léger contour ombragé au groupe. Pour finir, elle superpose un label contenant le texte 'A' au-dessus du groupe de rectangle, qu'elle centre en appelant une commande « centrer » depuis sa boîte à outils. Elle forme à nouveau un groupe contenant cette fois-ci le label et le précédent groupe contenant les rectangles et le nomme « touche » dans l'arborescence de la scène graphique fournie par l'application (*Structuration*) (Figure 5).

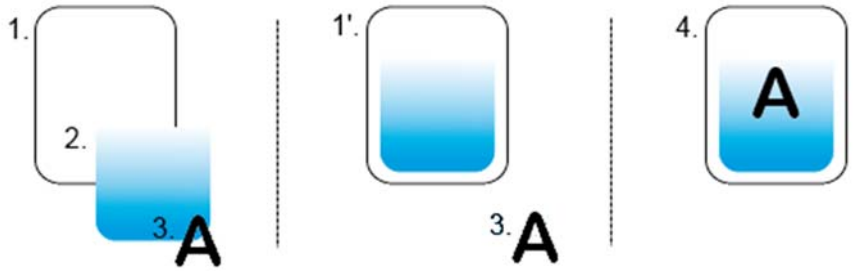


Figure 5. Élodie a créé trois objets graphiques : un rectangle blanc au bord arrondi (1), un rectangle peint avec un gradient bleu (2) et un label (3). Elle groupe (1) et (2) en (1') et groupe (1') et (3) pour obtenir le groupe (4) à droite quelle nomme « touche ».

Lorsque Élodie change la lettre 'A' pour la lettre 'I', elle s'aperçoit que le 'I' n'est pas centré au regard des deux rectangles (Figure 7). Le premier objet n'était pas spécifié correctement : si les trois objets (label, rectangle au gradient bleu et rectangle blanc) sont bien alignés, le texte du label lui n'est pas centré, mais aligné à gauche qui est la valeur par défaut. Le problème n'était pas perceptible avec les premières lettres (A, Z, E, R, T, Y et U) car elles sont de largeur similaire. Chaque label étant inclus dans un groupe hétérogène (les groupes contenant d'autres objets que des labels), le système ne fournit pas de commande d'alignement du texte qui peut être appliqué à la sélection des groupes. Élodie se voit donc contrainte de cliquer à plusieurs reprises sur chaque groupe pour atteindre tous les labels un par un et leur appliquer la commande de centrage du texte (*Pouvoir d'action*). Elle estime qu'il serait bien plus simple de tout recommencer, c'est pourquoi elle décide de supprimer toutes les copies, de dégrouper la première touche, ensuite de centrer de le texte, de grouper à nouveau les objets, de recréer des copies puis de les déplacer, et enfin de modifier toutes les lettres une par une.

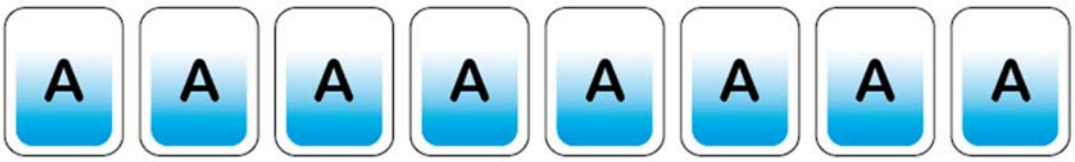


Figure 6. Premières touches du clavier, obtenues après avoir dupliqué la première touche qui contenait le label 'A'.

Figure 7. Les labels des touches ont été modifiés un par un afin de créer le clavier « AZERTY » ci-dessus. On peut apercevoir que le label 'I' n'est pas centré.

Élodie a terminé la réalisation du clavier entier. Certaines touches sont des touches doubles, qui contiennent deux plus petits labels en haut et en bas de la touche (Figure 8). Elle se demande si les touches doubles n'ont pas leur texte trop petit, et elle désire explorer de nouvelles tailles (*Exploration*). En premier lieu, elle doit trouver toute les doubles touches de son design (*Recherche*). Pour ce faire, elle effectue un zoom arrière pour rendre le clavier entièrement visible, ce qui lui permet d'identifier toutes les doubles touches. A nouveau, elle se voit contrainte de modifier la taille de chaque label un par un (*Pouvoir d'action*).

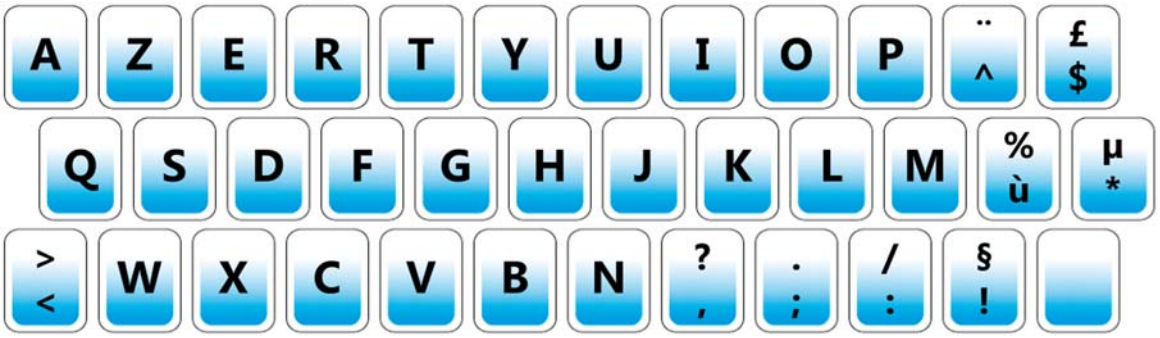


Figure 8. Clavier entier. Certaines touches sont des touches « doubles » : elles détiennent deux labels de taille inférieure aux autres.

2. Conception de plan avec AutoCAD

Fabrice est un concepteur de plan de lieux et bâtiments assisté par ordinateur, il utilise l'outil AutoCAD. Pour commencer Fabrice ouvre un document existant, décrivant le plan d'un campus. Le document s'avère complexe par la quantité d'information et d'éléments qui le constitue.

Fabrice souhaite modifier les portes d'un bâtiment. Chacune de ces portes sont identiques de par leur nature, seule change leur orientation. Il sélectionne la porte qu'il souhaite modifier puis bascule dans une vue ne contenant que la porte, où il devient possible de modifier ses propriétés. Il agrandit la largeur de la porte par manipulation directe, et l'on constate que les autres portes du bâtiment ont été également modifiées. Pour cause, la porte modifiée par Fabrice était en fait un modèle, appelé bloc dans AutoCAD, dont dépendent toutes les portes identiques de la scène (*Structuration*). Cela lui a permis d'éviter d'avoir à modifier toutes les portes une par une.

Fabrice souhaite à nouveau effectuer une modification aux portes de sa conception. Chaque porte est constituée d'un double trait qui représente leur épaisseur. Il est identique pour toutes les portes et a été obtenu par copie. Fabrice souhaite modifier l'espace entre les double traits pour augmenter l'épaisseur des portes, mais se trouve face à un problème complexe : les différents type de portes appartiennent à des blocs différents, et bien qu'elles possèdent toutes le même double trait, il doit modifier chaque bloc concerné un à un pour qu'elles puissent avoir la même épaisseur désirée. Cependant, après avoir modifié plusieurs blocs de portes, il s'aperçoit que le résultat n'est pas satisfaisant et souhaite essayer une largeur différente, il est donc contraint de modifier à nouveaux tous les blocs (*Exploration*).

Afin de retrouver tous les objets appartenant à un même bloc, Fabrice doit au préalable identifier le bloc par son nom. Il utilise ensuite un outil de recherche contenu dans une

fenêtre modale où il est possible de renseigner au moyen de listes déroulantes différents critères de recherche. Fabrice indique donc qu'il souhaite sélectionner tous les objets dépendants du bloc « Porte A », il valide son choix et tous les objets de ce bloc se voient alors sélectionnés dans l'espace de travail (*Recherche, Désignation*). Fabrice souhaite ensuite sélectionner toutes les portes dans leur globalité. Dans sa conception, les éléments sont organisés dans des calques qui se superposent. Il existe un calque pour les arbres, les murs, les portes etc. Il utilise alors le même outil que précédemment pour trouver tous les objets appartenant au calque contenant les portes (*Recherche, Structuration*).

3. Création d'un cours avec PowerPoint

Sébastien est enseignant-chercheur. Pour construire ses cours, il utilise un outil de présentation de diapositives. Dans les cours que construit Sébastien, la diapositive est dupliquée à plusieurs endroits, généralement avant chaque nouvelle partie abordée. La seule différence existante entre les copies concerne le titre de la partie abordée qui est mis en gras dans la diapositive « plan » qui la précède (Figure 9). Il procède alors ainsi lors de ses cours avant d'aborder une nouvelle partie : il rappelle le plan sur une première diapositive, puis sur la diapositive suivante présente le même plan où le titre de la partie qui suit est mis en gras. Il peut éventuellement présenter une troisième diapositive où le sous-plan de la partie est développé s'il existe.

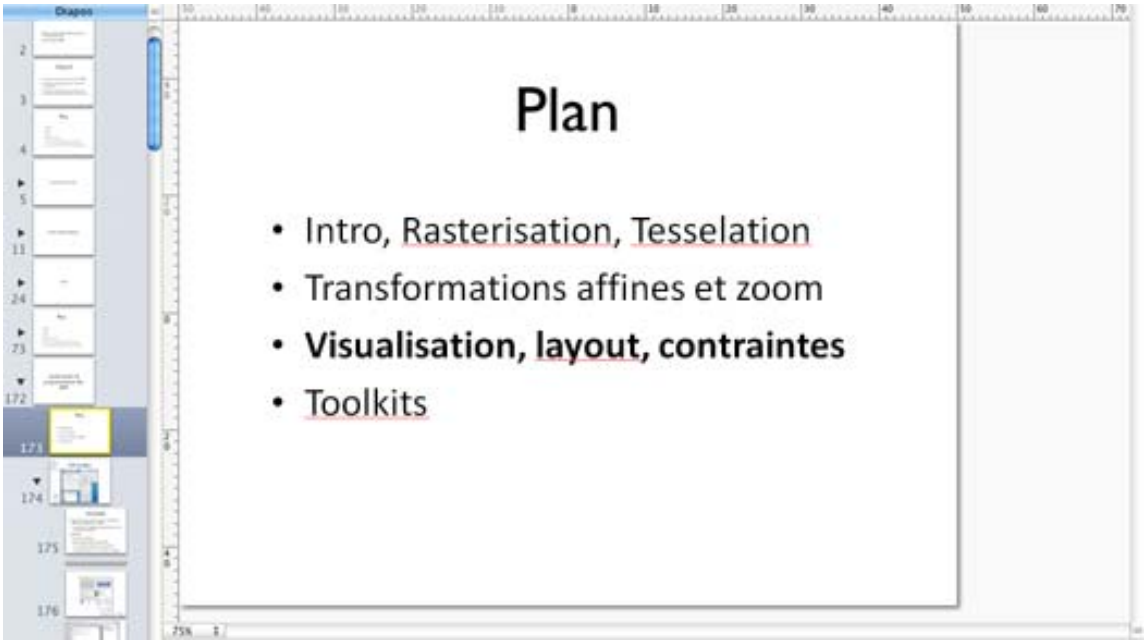


Figure 9. Diapositive contenant le plan. La partie abordée suivant cette diapositive est « Visualisation, layout, contraintes ».

Figure 10. Sébastien a ajouté la partie « Graphe de scène, picking, MDPC » dans son cours, il a alors modifié sa diapositive qui contient le plan.

Sébastien est face à un problème : le cours qu'il construit n'est jamais réellement fini, il lui faut fréquemment rajouter des parties ultérieurement, en modifier ou les déplacer (*Exploration*). Par conséquent, des diapositives contenant le plan sont dupliquées à de nombreux endroits dans le cours, et Sébastien avoue avoir du mal à les retrouver. En l'occurrence, il doit ajouter une partie « Graphe de scène, picking, MDPC » entre les parties « Transformations affines et zoom » et «Visualisation, layout, contraintes » (Figure 9 et Figure 10). Il doit alors trouver toutes les diapositives « plan » de son document, et les modifier une par une pour ajouter la nouvelle partie. Pour ce faire, il parcourt la liste des miniatures représentant les diapositives et repère celles qui ressemblent au plan (*Recherche*), les sélectionne et les modifie (Figure 11 et Figure 12). En parcourant l'ensemble de son document, Sébastien s'aperçoit qu'une des diapositives « plan » est différente des autres. Il lui avait appliqué des modifications et avait oublié d'en faire de même pour toutes les autres. Cela l'oblige à revérifier l'ensemble de ses parties, afin de s'assurer d'avoir un plan correct partout où celui-ci est présent.

4. Autres scénarios

D'autres scénarios de travails sont résumés en annexes de ce document page 155.

Figure 11. Sébastien doit ensuite repérer tous les autres plans pour leur ajouter également la nouvelle partie. Il fait défiler les miniatures sur la gauche pour repérer les diapositives.

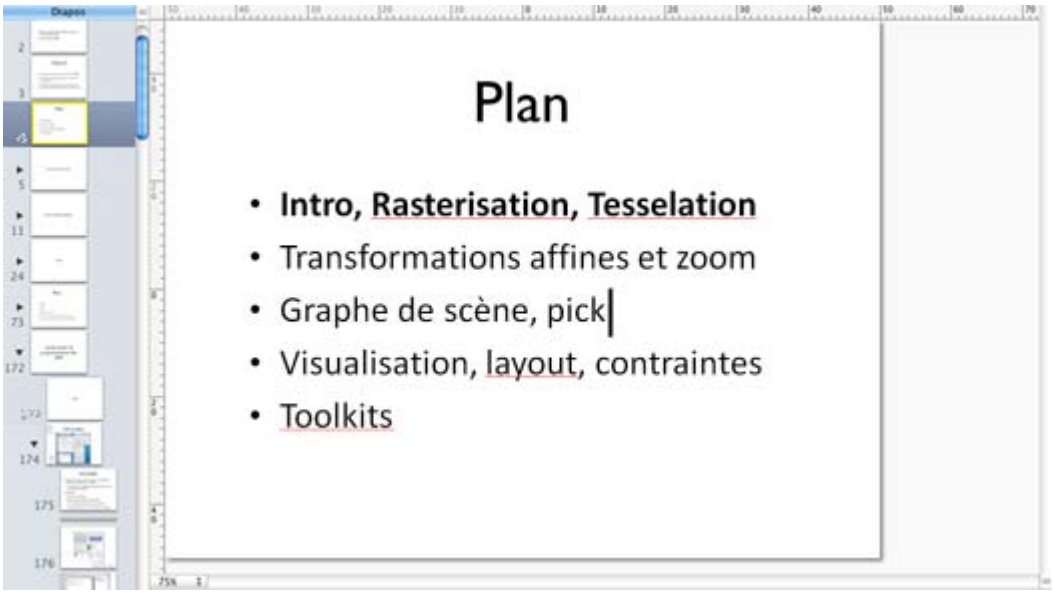


Figure 12. Sébastien sélectionne les plans et les modifie un par un.

C. Besoins identifiés

Les scenarios ont illustré plusieurs besoins qu'il est important de définir afin de mieux cerner la problématique et de concevoir des solutions qui y répondent. Nous avons observé que les utilisateurs avaient besoin de pouvoir *structurer leur conception*, qu'ils devaient pouvoir *désigner* des objets d'intérêt et des actions sur ces objets, qu'il leur fallait un *périmètre d'action* large pour interagir avec plusieurs objets en peu de fois et qu'ils avaient besoin de *trouver* des éléments répondant à des critères particuliers afin d'*explorer des solutions*.

1. Structuration

Élodie comptait sur la capacité du système à permettre la création, la modification et la gestion d'ensembles. C'était sa volonté lorsqu'elle créa un groupe contenant deux rectangles, puis un autre groupe contenant le précédent et un label. Fabrice pouvait utiliser des blocs, c'est-à-dire des modèles d'objets, à des fins conceptuelles et pratiques : en utilisant un modèle de portes, il pouvait définir des objets comme étant des portes et interagir sur le modèle au lieu des portes individuellement. L'utilisation de calques était aussi un moyen de structurer son document, avec un calque par catégorie d'objet.

L'utilisation de structures a donc un intérêt double pour les utilisateurs : organiser des éléments et nommer des concepts (par exemple, l'ensemble formé de deux rectangles et d'un label devient un objet « touche »), et permettre d'agir sur plusieurs éléments de façon synchrone en interagissant avec la structure. L'utilisation de structures s'inscrit donc comme un élément de réponse au besoin d'une capacité supérieure concernant l'interaction avec plusieurs objets.

2. Désignation

Élodie désignait des objets, des propriétés et des actions. Par exemple, en changeant l'alignement du texte, elle désignait la propriété d'alignement, l'objet label, et la valeur « centré ». Fabrice désignait la propriété épaisseur des portes en sélectionnant le double-trait qui la décrivait, et la valeur choisie en ajustant l'espace entre les doubles-traits par manipulation directe. Sébastien souhaitait désigner des objets ayant des contenus similaires, mais le système ne proposait pas de telle fonctionnalité.

La désignation peut être considérée comme une sous-étape d'une interaction : l'utilisateur désigne les objets avec lesquels il souhaite interagir (par exemple par le biais d'une sélection), il désigne également les opérations à réaliser. Désigner peut engendrer des actions supplémentaires, comme ce fut le cas lorsqu'Élodie devait accéder à chaque label un par un. Il nous paraît donc important que les techniques d'interaction puissent offrir une capacité de désignation peu couteuse en efforts, notamment pour interagir avec des objets multiples.

3. Pouvoir d'action

Élodie agissait sur plusieurs objets à la fois. Grouper des objets était un moyen de les considérer comme une seule entité où ses constituants gardaient une position relative les uns par rapport aux autres, ce qui lui permettait d'appliquer une seule translation à trois objets à la fois. En revanche, il ne lui était pas permis de changer l'alignement du

texte de plusieurs objets en même temps. Grâce aux blocs d'AutoCAD, Fabrice pouvait interagir sur toutes les portes d'un même bloc simplement en modifiant celui-ci. Sébastien n'avait pas la possibilité d'agir sur tous les plans de son document à la fois, il lui fallait les modifier individuellement alors que les objets étaient presque identiques et les actions étaient à peu de choses près les mêmes.

L'identification de ce besoin justifie l'importance de la problématique : s'il existe des techniques qui permettent d'agir sur plusieurs objets de façon synchrone, telles que les groupes ou les calques, certaines interactions ne le permettent pas (à l'instar des modifications des diapositives) et contraignent l'utilisateur à les répéter. Augmenter le pouvoir d'action revient alors à trouver des solutions qui permettent d'étendre les capacités des utilisateurs à réaliser des interactions sur une multitude d'objets dans toutes circonstances.

4. Recherche

Élodie avait besoin de trouver certaines touches du clavier virtuel : il lui fallait chercher celles dont le contenu était similaire à d'autres, en l'occurrence toutes les touches détenant deux labels au lieu d'un. Sébastien devait repérer toutes les diapositives « plan » dans son document pour les mettre à jour manuellement et Fabrice souhaitait pouvoir trouver toutes les portes appartenant à un modèle particulier.

L'action de recherche requiert un parcours visuel des objets graphiques et de repérer les objets correspondants aux critères désirés, au risque d'en oublier certains. Plus il y a d'objets et plus il devient difficile de trouver ceux qui nous intéressent, d'autant plus si la façon de les repérer n'est pas pré-attentive (Conversy, Hurter, & Chatty, 2010). Par exemple, une recherche dans un espace visuel est pré-attentive si l'on recherche des objets selon leur couleur (des cercles de couleurs jaune se repèrent immédiatement dans un ensemble de cercles de couleurs différentes), contrairement à une recherche sur la forme (il est difficile d'identifier tous les cercles parmi un ensemble de formes variées telles que des triangles, carrés etc.). Fabrice pouvait se reposer sur l'utilisation d'une fonctionnalité de recherche qui permettait d'identifier tous les objets appartenant à un même modèle, ce qui évitait l'effort de recherche visuelle au prix d'une interaction. Dès que le nombre d'objet augmente, chaque modification individuelle devient coûteuse, non seulement parce que le nombre d'actions à répéter est plus important, mais également parce que cela demande un plus grand effort de recherche visuelle. Qui plus est, le risque d'erreur ou d'oubli augmente dans ces conditions et peut s'avérer également coûteux, d'autant plus s'il devient nécessaire de

recommencer ces étapes de recherche et de modification après avoir découvert une erreur ou un oubli.

5. Exploration

Élodie explorait plusieurs parties de solutions possibles, et modifiait des parties de ces solutions. Ce fut le cas lorsqu'elle essaya de comparer différent résultats pour les tailles des textes inclus dans les touches doubles. Fabrice ajustait l'épaisseur de portes jusqu'à obtenir un résultat satisfaisant. Sébastien construisait un document dont le contenu pouvait changer à tout moment, et devait le réadapter en conséquent.

En combinant action, visualisation des résultats intermédiaires, et réflexion, ils co-découvraient le problème et les solutions. Il s'agit alors d'une activité de conception exploratoire (T. R. G. Green, 1989). Le phénomène est important dans les activités où le résultat attendu n'est pas connu à l'avance : activité d'édition graphique, réalisation de documents textuels ou de hiérarchie de classe… (T. R. G. Green, 1989)(Terry & Mynatt, 2002a). Cette activité se trouve au cœur de notre problématique, dans le sens où le besoin d'explorer implique un besoin de réaliser des modifications multiples et ainsi d'avoir le pouvoir d'action nécessaire pour y parvenir.

D. Synthèse

Nous avons présenté trois scénarios de travails issus d'enquêtes menées en contexte auprès d'utilisateurs d'éditeurs graphiques (Illustrator, AutoCAD et PowerPoint). D'autres scénarios sont présentés en annexe.

Les scénarios de travails ont révélé différents besoins concernant l'utilisation d'éditeurs graphiques : pouvoir structurer afin d'augmenter la portée des interactions et d'agir sur plusieurs objets, pouvoir rechercher et désigner un ensemble d'éléments ou de propriétés et pouvoir explorer des solutions. L'identification de ces besoins a orienté les recherches que nous avons menées par la suite dans la thèse.

L'analyse des besoins et les observations issues des enquêtes contextuelles permettent également de mieux justifier l'importance de la problématique de recherche. Nous avons appliqué pour cela une méthode d'évaluation de nouveaux systèmes (Olsen,Jr., 2007). Bien qu'elle s'applique plus particulièrement à des recherches sur l'architecture de systèmes, cette méthode peut se généraliser à l'interaction. La démarche consiste à identifier le contexte (la situation, la tâche et les utilisateurs), l'importance de la population rencontrant le problème, l'importance de la tâche et de la situation. Il est important de savoir si le problème n'a pas déjà été résolu, et si les solutions peuvent se

généraliser à d'autres contextes. Le Tableau 1 présente le résultat de l'application de cette méthode à partir de nos observations et démontre l'importance du problème pour les raisons suivantes : le problème se manifeste fréquemment pour tout type d'utilisateur de l'outil informatique dans toute situation où l'activité consiste en une exploration de solutions. Par ailleurs, le problème n'est pas encore résolu.

Tableau 1. Importance de la problématique

Contexte	Situation	Édition, activité occasionnelle ou professionnelle.
	Tâches	Conception exploratoire (utilisation d'éditeurs graphiques, d'outils de présentation, d'éditeurs d'agenda etc.).
	Utilisateurs	"Designers" au sens large, professionnels d'outils avancés (ex : Illustrator, AutoCAD) et utilisateurs amateurs de l'outil informatique au quotidien.
Importance	De la population	Le problème est rencontré par la grande majorité des utilisateurs de l'outil informatique : il s'est manifesté au cours de chacun des scénarios présenté dans cette partie ainsi qu'en annexes de ce manuscrit.
	De la tâche	Cela dépend bien entendu du contexte, la tâche peut être importante dans un cadre professionnel.
	De la situation	La situation est récurrente, le problème ayant été observé lors de chaque scénario.
Problème non encore résolu ?		Le problème est toujours d'actualité. Bien que des solutions aient été proposées comme nous le verrons ultérieurement dans le manuscrit, elles imposent cependant des compromis.
Généralisation		Le champ d'utilisateur étant assez large, allant du designer expert dont c'est le métier (moins confronté au problème en raison de son expertise) à l'utilisateur occasionnel qui va souhaiter modifier son agenda personnel sur iCal, la problématique se généralise pour tout utilisateur et éditeur graphique.

ÉTAT DE L'ART

Nous avons sélectionné un ensemble de travaux existants dont les préoccupations sont liées aux cinq besoins (structuration, désignation, pouvoir d'action, recherche et exploration) identifiés lors des enquêtes contextuelles. Ces travaux sont de natures différentes : nous avons étudié des cadres d'analyse (les dimensions cognitives), les paradigmes d'interaction ainsi que diverses techniques d'interaction, l'évaluation des techniques d'interaction, et le domaine de la programmation.

A. Dimensions cognitives et conception exploratoire

L'utilisation d'éditeurs graphiques induit systématiquement le besoin de répéter fréquemment les mêmes tâches, par exemple en devant changer un ensemble de propriétés graphiques, n'importe où dans une large scène, ou encore en devant modifier toutes les occurrences d'une même forme, voire d'établir des relations géométriques répétitivement parmi un ensemble d'objets différents (Kurlander, 1993). Repositionner un objet peut imposer de replacer d'autres objets dépendants. Ce problème de répétition, aussi appelé « viscosité » est un véritable frein à l'activité dite de conception exploratoire, c'est à dire l'élaboration d'un travail dont le rendu final n'est pas connu à l'avance (T. R. G. Green, 1989), en raison du coût engendré par de telles répétitions dans un contexte où l'utilisateur éprouve le besoin d'explorer des solutions. Il doit pouvoir modifier facilement son travail.

1. Les dimensions cognitives

(T. R. G. Green, 1989) présente les dimensions cognitives des notations et ce qu'il définit comme la « planification opportuniste ». Selon la théorie de l'action[2] de Norman (Norman, 1988), l'acteur traduit un ensemble de buts à atteindre en une séquence d'actions qu'il exécute. Théorie que Green décrit comme « naïve » dans le sens où cela implique que l'on procède du début jusqu'à la fin sans faire d'erreur ou d'omission, que l'on soit en train d'écrire un programme, un article, ou de réaliser une conception

[2] Toute intention d'interagir avec un système est issue d'un but initial. De cette intention nait une spécification d'une suite d'actions, qui sont exécutées afin d'agir sur le système. Après exécution des actions, l'état du système est perçu puis interprété, afin d'être évalué par rapport au but à atteindre. Il s'agit des sept étapes de la théorie de l'action (Norman, 1988).

graphique. Cependant, selon Green, tout but ou sous-but devrait pouvoir être considéré à n'importe quel moment, dans un contexte où toute éventualité peut se produire. C'est pourquoi la stratégie cognitive prépondérante dans la conception est ce que Green appelle la planification opportuniste : on réévalue sans cesse, on change d'avis en cours de route, on observe. On a besoin de modifier et de reconstruire.

Les corollaires établis par Green pour supporter la planification opportuniste sont les suivants : pouvoir considérer tout but à tout niveau, pouvoir faire des comparaisons et percevoir la structure de la notation. Permettre également la modification, car toute solution à un but donné pourrait être sujette à un changement futur. On doit être capable d'écrire une notation (c'est-à-dire la structure d'information : un document texte, du code, une scène graphique), la lire et changer ce qui est écrit. Les dimensions cognitives d'une notation caractérisent donc la façon dont l'information est structurée et représentée.

Les dimensions cognitives sont avant tout des outils de discussions. Donner des noms à des concepts identifiés est essentiel pour des discussions précises entre concepteurs sur les qualités d'une interface particulière : il n'y a plus besoin de réexpliquer une énième fois chaque concept dès l'instant où un nom lui a été donné. L'analyse par les dimensions cognitives se veut très différente des approches d'analyses traditionnelles (T. Green & Blackwell, 1998) telles que GOMS et le « Keystroke Level Model » de Card (Card, Moran, & Newell, 1980) et l'évaluation heuristique de Nielsen (Nielsen & Molich, 1990). Le « Keystroke Level Model » offre des prédictions de temps requis pour accomplir une tâche déterminée, ce qui demande une analyse détaillée des tâches, c'est un investissement considérable en temps et en expertise, d'autant plus que des approches inventives de la part des utilisateurs peuvent ne pas avoir été conçues par les analystes. L'évaluation heuristique sert à identifier des erreurs de conception, elle est plus rapide à appliquer que l'analyse des tâches, mais cette approche ne peut être appliquée que par des spécialistes de l'IHM et ne tient pas compte de la structure de l'information sous-jacente. Les dimensions cognitives se veulent complémentaires à ces deux approches. Elles visent à être rapides à comprendre et à appliquer, elles différencient les types d'activités de l'utilisateur et sont compréhensibles par des non-spécialistes de l'IHM. Elles sont utiles pour évaluer (T. R. G. Green & Petre, 1996) et concevoir les artefacts d'information, c'est à dire les outils que l'on utilise pour stocker, manipuler et présenter de l'information (T. R. G. Green, 1989), qu'ils soient interactifs ou non. Un éditeur graphique ou un téléphone portable sont des exemples d'artefacts

d'information interactifs, une notation musicale ou bien un langage de programmation sont eux des artefacts non-interactifs.

a) Les activités de l'utilisateur

Les principales activités d'un utilisateur sont l'incrémentation, la transcription, la modification et la conception exploratoire (T. R. G. Green, 1989). L'activité d'incrémentation consiste en l'ajout d'information à une structure existante sans l'altérer d'aucune sorte. Par exemple, ajouter une nouvelle entrée dans une base de données ou bien dessiner un nouvel objet dans un éditeur graphique. L'activité de transcription consiste en la copie du contenu d'une structure à une autre : recopier les informations d'un article dans la bibliographique, convertir une formule mathématique en une formule dans un tableur. L'activité de modification consiste à changer une structure existante, potentiellement en ajoutant du nouveau contenu. Par exemple, dans le scénario de travail du clavier virtuel présenté page 10, l'utilisateur devait modifier l'alignement de toutes les lettres. La conception exploratoire quant à elle, combine incrémentation et modification. Elle se manifeste lorsque l'état final désiré n'est pas connu à l'avance : conception typographique, dessin, programmation à la volée… Dans (Instructions and description: some cognitive aspects of programming and similar activities, 2000) Green présente deux autres activités : l'activité de recherche, c'est-à-dire « chasser » une cible connue, par exemple l'endroit où une fonction est appelée, et la compréhension exploratoire, qui se manifeste lorsque l'on découvre la structure d'un algorithme ou les bases d'une classification par exemple.

Les concepteurs voient facilement un dispositif comme un pur système d'incrémentation, et négligent le besoin de restructurer (T. R. G. Green, 1989). Selon la tâche à accomplir et les outils à disposition, certaines activités sont plus mises en avant que d'autres. Green et Blackwell présentent l'exemple suivant : la conception d'un jardin est différente de celle d'un pont (T. Green & Blackwell, 1998). La première a trait à l'esthétique, il faut explorer les options quitte à changer à un moment ce qu'on avait en tête, c'est de la c*onception exploratoire*. La seconde est plus critique en termes de sécurité et moins créative, il s'agit d'une activité d'*incrémentation*. Chacune de ces activités est supportée par un profil de dimensions cognitives. Les activités de modification et la conception exploratoire sont au cœur de notre problématique, c'est pourquoi nous nous intéressons aux dimensions cognitives permettant de les supporter au mieux.

b) La viscosité

Une des dimensions au cœur du sujet se nomme « viscosité » par analogie au phénomène physique de résistance au changement. La viscosité se manifeste lorsqu'un but simple requiert un nombre important d'actions individuelles pour être atteint (comme nous avons pu le voir dans les scénarios de travail) (T. R. G. Green, 1989). Plus précisément, on peut aussi parler de viscosité « knock-on », percutante : on a un changement en tête que l'on réalise mais qui oblige à réaliser d'autres actions afin de restaurer la consistance entre l'élément modifié et les autres. On y est confronté par exemple si l'on doit insérer une figure dans un document et que cela crée le besoin de mettre à jour tous les numéros de figures suivantes, ainsi que la table des figures.

La viscosité devient un véritable problème pour la planification opportuniste, lorsque l'utilisateur modifie son plan. Elle casse le « train de pensée » et peut rendre trop coûteux ou risqué de procéder à des changements. En revanche elle encourage la réflexion et la planification (ce qui peut aussi réduire les risques d'erreur). Il faut noter que la viscosité n'est pas toujours mauvaise puisqu'elle peut encourager une plus profonde réflexion et apprentissage. Qui plus est, elle peut être désirable pour les systèmes critiques (T. Green & Blackwell, 1998). Si la viscosité est acceptable pour la transcription et l'incrémentation, elle s'avère problématique pour la modification et l'exploration.

Quels sont alors les remèdes à la viscosité ? Une solution de contournement consiste à introduire un média intermédiaire peu visqueux servant de version d'ébauche que l'on transcrit ensuite sur le média original. Ainsi le processus est divisé en deux partie : de l'exploration puis de la transcription. Un exemple : dessiner au crayon avant de passer à l'encre, ou bien écrire du code sur une feuille de brouillon avant de coder sur machine. De façon alternative, il peut s'agir d'adopter une autre notation au lieu d'un autre média : écrire en pseudo-langage avant de le transcrire dans le langage cible. Un autre remède, plus classique, consiste à introduire une nouvelle abstraction.

c) Abstraire pour structurer

« Une abstraction est une classe d'entités, ou un groupement d'éléments que l'on peut traiter comme une seule entité, autant pour réduire la viscosité que pour rendre la notation plus proche de la structure conceptuelle de l'utilisateur » (T. R. G. Green, 1989). Les styles dans un éditeur de texte sont un exemple d'abstraction. Face à un problème visqueux, tel que la numérotation de figures dans un document texte qui nécessite d'être mis à jour à chaque insertion de figure, créer une fonctionnalité de mise à jour automatique des indices de figure est un exemple d'abstraction pouvant

résoudre ce problème. En structurant son travail, l'utilisateur aura un plus grand pouvoir d'action sur celui-ci.

Une abstraction apporte de la concision dans un système. Cependant cela à un coût, d'autant plus si les abstractions doivent être maintenues (on parle d'« abstraction management »). Leur création demande du temps et un certain effort, qui peuvent être contrebalancés par la répétition d'une courte séquence d'action pour résoudre un petit problème. Elles sont alors utiles pour les activités de modification et de transcription, plutôt utiles pour l'incrémentation, mais problématiques pour l'exploration, car cette activité nécessite la remise en question de la structure établie. Qui plus est l'expertise est à prendre en compte, par exemple les gens peu familiers d'un éditeur de texte ne vont pas utiliser les styles, formatages automatiques ou les macros.

Il est préférable que la structure soit visible afin d'éviter les *dépendances cachées*, et que l'utilisateur ne se voit pas imposé de créer des structures trop tôt pour éviter *l'engagement prématuré*. Les dépendances cachées sont une autre dimension cognitive. Elles décrivent des relations entre deux éléments (l'un dépendant de l'autre), entraînant une dépendance qui n'est pas entièrement visible, ce qui peut rendre la prédiction du résultat d'un changement difficile. Par exemple dans un outil de type tableur tel qu'Excel qui présente une grille de calcul constituées de cellules, certaines cellules de la grille peuvent se voir liées et dépendantes à d'autres par le biais de formules. Les dépendances sont cachées car l'utilisateur ne sait pas quelles cellules dépendent des autres, il ne peut donc prévoir les conséquences d'une modification d'une cellule sur d'autres (Figure 13). L'engagement prématuré est également une dimension cognitive, il se caractérise par des contraintes sur l'ordre imposé pour réaliser une tâche, forçant l'utilisateur à prendre des décisions avant que l'information ne soit disponible. Si l'on reprend l'exemple du tableur de la Figure 13, un cas d'engagement prématuré se manifesterait si l'on était contraint de remplir les cellules A1 et A2 avant d'établir que la valeur de la cellule C3 est égale à la somme de ces deux cellules.

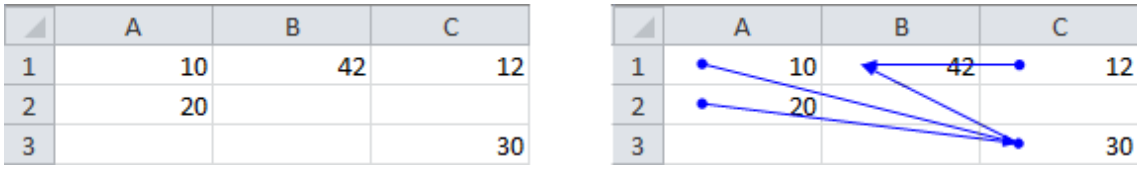


Figure 13. La cellule C3 dépend des cellules A1 et A2, et la cellule B1 dépend de C3 et C1. Par défaut ces dépendances ne sont pas visibles (figure de gauche), si l'utilisateur modifie A1 ou A2 il ne peut pas prévoir que les cellules C3 et B2 vont être modifiées à leur tour. Une commande de menu permet d'afficher temporairement ces dépendances pour une cellule donnée (à droite).

Dans un système requérant des abstractions, la barrière d'abstraction est une variable qui définit le nombre minimal d'abstractions à réaliser avant de pouvoir utiliser le système. Il est donc souhaitable qu'elle soit la plus basse possible. Permettre des abstractions peut s'avérer bénéfique, les imposer l'est moins. Un système requérant des abstractions est dit « abstraction-hungry », un système permettant les abstractions est dit « abstraction-tolerant » et un système les empêchant est appelé « abstraction-hating ». Il est préférable de faire des systèmes tolérants, mais avec une barrière la plus basse possible. Ainsi, l'utilisateur peut utiliser les abstractions s'il le désire sans qu'il ne soit contraint de le faire. Pour conclure, l'utilisateur peut avoir besoin de créer des abstractions et de structurer son travail afin d'agir plus efficacement (temps et nombre d'actions) sur un plus grand nombre d'éléments à la fois, mais il faut considérer le temps et l'effort nécessaire pour créer ces structures, éviter les dépendances cachées, et ne pas les imposer pour limiter l'engagement prématuré.

d) Maximiser les dimensions : profil idéal

Dans l'absolu une dimension n'est ni bonne ni mauvaise. S'il semble être du bon sens de tendre vers une viscosité minimale, il faut considérer le fait que la viscosité est une dimension et non un problème, avec des implications selon son importance. Par ailleurs, vouloir maximiser une dimension se fait généralement au détriment d'une ou plusieurs autres. Par exemple, on peut réduire la viscosité en augmentant les coûts d'abstractions et d'engagement prématuré. Il faut savoir faire des compromis, une dimension ne s'ajuste pas arbitrairement et en changer une risque d'affecter les autres (Figure 14), comme en physique : changer la température d'un corps va changer son volume, sauf s'il est compressé auquel cas c'est sa pression qui va changer (T. R. G. Green & Petre, 1996). Dans le milieu des systèmes critiques il peut être nécessaire d'avoir des systèmes visqueux et compliqués à apprendre afin de limiter leur utilisation à ceux qui savent l'utiliser. La pire combinaison de dimensions possible combine viscosité et engagement prématuré (T. R. G. Green, 1989) : l'utilisateur devant faire des suppositions préalables (engagement prématuré) pour découvrir plus tard que corriger ces suppositions est coûteux en temps ou en effort (viscosité).

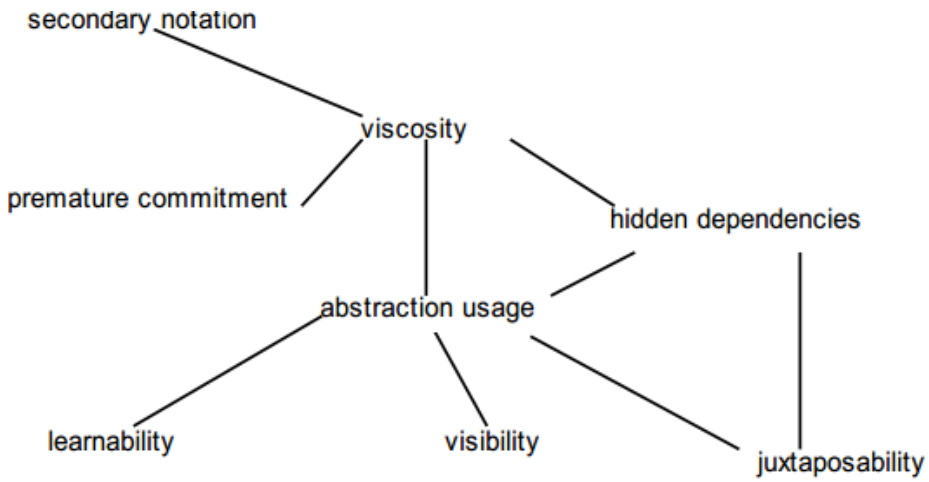


Figure 14. Les liens entre dimensions indiquent les compromis possibles : ajuster l'une de ces dimensions peut avoir des répercussions celles qui lui sont liées (T. Green & Blackwell, 1998, fig. 20)

Il est donc important de comprendre les relations entre dimensions. Une façon d'envisager de bonnes interactions serait peut-être d'essayer de maximiser les dimensions qui supportent aux mieux les activités cibles. Green et Blackwell présentent dans (Cognitive Dimensions of Information Artefacts: a tutorial, 1998) un tableau récapitulatif des dimensions et de leur impact sur les différentes activités de l'utilisateur (Figure 15), ainsi que des « design manœuvres », c'est-à-dire des solutions possibles pour augmenter ou réduire une dimension dans le but de résoudre un problème donné (Figure 16). [3]

	transcription	*incrementation*	*modification*	*exploration*
viscosity	acceptable	acceptable	harmful	harmful
hidden dependencies	acceptable	acceptable	harmful	acceptable for small tasks
premature commitment	harmful	harmful	harmful	harmful
abstraction barrier	harmful	harmful	harmful	harmful
abstraction hunger	useful	useful (?)	useful	harmful
secondary notation[3]	useful (?)	–	v. useful	v. harmful
visibility / juxtaposability[3]	not vital	not vital	important	important

Figure 15. Table des activités de l'utilisateur. (T. Green & Blackwell, 1998, fig. 19).

[3] La notation secondaire est une dimension qui se caractérise par l'information supplémentaire apportée par d'autres moyens que la syntaxe officielle (indentation dans les programmes, groupement de chiffres de téléphones deux par deux). La visibilité c'est la capacité à voir les composants facilement, la juxtaposition c'est celle de placer n'importe quels éléments côtes à côtes.

Il est évident que pour favoriser au mieux les activités de conception exploratoire ainsi que de modification, la viscosité doit être la plus basse possible. Permettre des abstractions est utile, il ne faut pas cependant qu'elles soient imposées au risque de contraindre l'utilisateur.

AIM	MANOEUVRE	AT THIS COST
to reduce viscosity:	add abstractions (so that one 'power command' can change many instances	increases need for lookahead (to get the right abstractions); raises the abstraction barrier; may also increase hidden dependencies among the abstractions
to improve comprehensibility:	allow secondary notation – let users choose where to place, things, how to use white space, what font and colour to use; allow commenting	increases viscosity (because layout, colour etc are not usually well catered for by the environment)
to make premature commitment less expensive:	reduce viscosity (so that users can easily correct their first guess)	see above, re viscosity
to remove need for lookahead:	remove internal dependencies in the notation; or allow users to choose an easier order to make decisions	may make notation diffuse, or increase errors allow free order needs a cleverer system
to improve visibility:	add abstractions (so that the notation becomes less diffuse)	see above re abstractions

Figure 16. Table des manœuvres de conception (T. Green & Blackwell, 1998, p. 43).

2. La conception exploratoire

La conception exploratoire est donc l'activité majoritairement ciblée par nos travaux. L'utilisateur ne sait pas quel est encore l'état final, il est dans un processus créatif et les objectifs peuvent changer à tout moment, nécessitant de procéder à des modifications du travail préalablement fait. Des chercheurs se sont efforcés de comprendre et de définir des mécanismes dans les systèmes pour qu'ils supportent et améliorent ce processus.

Par exemple, (Terry & Mynatt, 2002a) ont présenté trois cas d'études pour démontrer les besoins typiques des utilisateurs, lorsqu'ils progressent dans les tâches dites « open-ended ». Ces trois cas d'études démontrent quels sont les besoins des utilisateurs afin de supporter la conception exploratoire : Expérimenter, explorer des variations, et évaluer. L'utilisateur a besoin d'expérimenter pour mieux comprendre le

problème et les options disponibles, de générer des variations pour attaquer le problème sous plusieurs angles et d'évaluer ses efforts pour réfléchir sur sa progression et s'informer des actions futures. Le besoin d'expérimentation à court-terme est important pour des tâches ouvertes. Savoir quelle est la prochaine action à faire peut demander un certain effort : l'utilisateur peut ne pas savoir quelle commande invoquer ainsi que les paramètres optimaux. Il doit faire des hypothèses sur ce qu'il doit faire ensuite et teste ces hypothèses en invoquant une commande et en ajustant ses paramètres pour obtenir le résultat escompté. Dès que ce résultat est disponible (par le résultat de l'action, ou par une prévisualisation de celui-ci), l'utilisateur peut alors l'accepter, ou bien l'ajuster à nouveau, ou alors l'annuler et essayer autre chose. Tandis que l'expérimentation à court-terme aide l'utilisateur dans le choix des commandes, il peut y avoir des situations où une exploration plus profonde est nécessaire (Terry, Mynatt, Nakakoji, & Yamamoto, 2004). La variation, un support aux versions, permet alors de mieux comprendre le problème et ses frontières et de découvrir d'autres solutions potentielles. Et enfin, faciliter l'évaluation permet à l'utilisateur d'expérimenter avec plus d'aisance. Par exemple, offrir une prévisualisation de commande demande moins d'efforts que d'appliquer réellement la commande, puis d'évaluer son résultat, et de l'annuler ensuite.

Les cas d'études de Terry et Mynatt portaient sur différents aspects inhérents à la conception exploratoire, révélant des processus fortement itératifs ainsi que la nécessité de maintenir plusieurs versions d'un même design. Une technique adoptée par l'utilisateur consiste à faire des copies du travail en cours afin d'explorer d'autres possibilités en profondeur. Afin d'évaluer le résultat d'une opération, une utilisation répétée des commandes *annuler* et *refaire* permet de révéler très brièvement des différences entre les versions courantes et précédentes. Ces techniques permettent à l'utilisateur d'évaluer le résultat de ses actions à différentes échelles, variant les niveaux de détails et via des représentations multiples. Utiliser les commandes *annuler* et *refaire* pour générer des variations est problématique à cause de sa nature éphémère : il n'existe qu'une seule version à la fois, rendant la comparaison difficile, il s'agit d'une comparaison faite dans le temps et non dans l'espace. Nous sommes alors témoins d'une utilisation détournée des outils mis à dispositions (annuler et refaire, ou multiplication des fichiers) pour pouvoir faire de l'exploration. Des outils permettant l'utilisation de calques (comme des outils de dessin vectoriels tels qu'Illustrator, InkScape, Paint.Net etc.) peuvent aider plus facilement à la gestion de versions, même s'il s'agit là encore d'une utilisation détournée de la fonctionnalité première, puisqu'ils permettent plus facilement le changement d'une version à une autre. Cela ne permet là

encore qu'une comparaison dans le temps et non dans l'espace : il n'est pas possible de comparer deux designs côte à côte. Lorsque les outils ne permettent pas nativement de comparer des versions d'un design (*dimension cognitive : visibilité/juxtaposition*), l'utilisateur doit gérer lui-même ses versions.

Bien sûr le mécanisme des actions réversibles est une fonctionnalité indispensable à la conception exploratoire, ou plus spécifiquement : la possibilité de revenir à un état précédent car la solution explorée ne convient pas. Ce mécanisme autorise l'utilisateur à prendre des risques ce qui lui permet d'explorer sans crainte d'avoir à tout recommencer s'il commet une erreur. Notons que la correction d'erreurs est un critère d'ergonomie (Bastien & Scapin, 1993). (Fekete, 1996) présente l'annulation comme étant l'un des trois services indispensables à l'IHM.

B. Paradigmes d'interaction

La manipulation directe ainsi que l'interaction instrumentale sont des techniques efficaces pour interagir avec un objet simple : elles réduisent le nombre d'actions requises comparées à d'autres techniques telles que les lignes de commandes ; le dialogue conversationnel, ou les interactions modales. Les principes de conception appliqués à l'interaction instrumentale, tels que la réification (transformer un concept en objet), le polymorphisme (appliquer une même opération à différentes classes d'objets) et la réutilisation (de sélections passées et de précédents résultats d'interactions) étendent le pouvoir d'action sur des objets multiples.

1. La manipulation directe

L'un des principes d'interaction les plus répandus dans les applications graphiques actuelles depuis de nombreuses années est celui de la manipulation directe (Shneiderman, 1987).

a) Principes de la manipulation directe

Ce paradigme d'interaction est caractérisé par la représentation d'objets dits « d'intérêts » par le système, manipulables « physiquement » par l'utilisateur. Au lieu d'exprimer des instructions via une syntaxe compliquée, l'utilisateur va pouvoir agir directement sur ses objets d'intérêts et en percevoir immédiatement les conséquences. La représentation continue des objets d'intérêts, les actions physiques, les opérations rapides, incrémentales et réversibles dont l'impact sur les objets d'intérêts est immédiatement visible sont les trois principes qui définissent la manipulation directe. La manipulation directe s'avère facile d'utilisation et s'apprend vite, elle réduit les

risques d'erreurs et facilite leur correction, et diminue l'anxiété de l'utilisateur grâce à une meilleure compréhension du système (Shneiderman, 1987).

Dans (The History and Future of Direct Manipulation, 1993), Frohlich présente une analyse de la manipulation directe, et s'interroge sur la pérennité de philosophie de la manipulation directe, en explicitant ses limitations. En s'appuyant sur les travaux de (Hutchins, Norman, & Hollan, 1986) Frohlich réexplique ce qui rend la manipulation directe si attrayante à partir de deux notions particulières : la distance et l'engagement. La distance correspond à la mesure de ce qui sépare la façon de penser de l'utilisateur concernant un problème donné et la représentation de celui-ci par le système. Ainsi les systèmes réduisant cette distance réduisent également l'effort cognitif concernant ce que l'on peut faire (on parle de distance sémantique) et comment le faire (distance articulatoire) (Hutchins et al., 1986). L'engagement (Laurel, 1986) caractérise l'implication émotionnelle de l'utilisateur lorsque le système lui donne l'impression d'être l'acteur principal pouvant agir sur le monde. Ce sentiment est encouragé par les systèmes qui représentent les objets graphiquement et qui permettent de les manipuler physiquement, plutôt que par l'intermédiaire d'instructions à énoncer.

b) Limitations et défis

Si l'on peut affirmer que les systèmes à manipulation directe ont tendance à minimiser la distance et maximiser l'engagement, qu'il est indéniable que la manipulation directe est riche de qualités (feedback immédiat, facilité à traduire naturellement des intentions en actions, métaphore du monde réel), elle possède aussi des faiblesses. Il s'avère que ces systèmes ne sont pas nécessairement adaptés à tous types de problèmes, et plus spécifiquement, qu'ils ne supportent pas les opérations répétitives, la capacité à identifier des ensembles d'objets et celle de spécifier des valeurs très précises via la manipulation (Hutchins et al., 1986). Un exemple donné par Hutchins, (1989) et repris par Frohlich dans son analyse (1993) : une tâche demandant de réaliser une action particulière sur tous les mots commençant par la lettre 's' serait très fastidieuse par manipulation directe puisqu'il faudrait trouver toutes les occurrences de ces mots et les modifier une par une. Cela montre, selon Hutchins, que la faible distance de la manipulation directe présente aussi des inconvénients puisqu'elle ne permet pas d'agir sur des abstractions ou des objets non perceptibles... Cela peut donc s'avérer problématique dès lors que l'on souhaite manipuler plusieurs objets à la fois, puisqu'une solution pour ce faire repose justement sur l'utilisation d'abstractions comme nous avons pu le voir précédemment.

(Kwon, Javed, Elmqvist, & Yi, 2011) pointent dans leurs travaux les limitations de la manipulation directe et présentent les défis qu'elle soulève : l'accès, la multiplicité, et les propriétés intangibles. Manipuler des objets petits, distants ou bien trop riches de propriétés, dans des espaces limités ou denses, avec une certaine précision est le défi de l'accès. Il est fastidieux avec la manipulation directe de sélectionner un objet trop petit, de saisir une poignée si elle est sous une autre, d'accéder à un objet dans une scène qui en contient un nombre important ou dans une scène où les objets sont très distants voire hors de l'espace visuel. Une solution partielle pour pallier ce problème est celle du zoom, qui donne accès à plus de détails ou de précision au détriment d'un effort cognitif supplémentaire.

Manipuler directement plusieurs objets à la fois, comme des groupes, n'est pas aussi intuitif que manipuler des objets seuls. Pour manipuler plusieurs objets d'un groupe on est contrait d'interagir avec un seul de ses objets arbitrairement, dans l'optique de reproduire la même interaction sur les autres. Cependant il n'est pas aisé de prédire comment le système va interpréter cette manipulation sur les autres objets. Par exemple, si l'on sélectionne plusieurs objets graphiques, et que l'on redimensionne l'un d'entre eux cela va-t-il modifier uniquement cet objet ou bien l'ensemble de la sélection ? Auquel cas, seront-ils modifiés avec le même décalage calculé à partir de l'objet manipulé ou bien avec une valeur relative à leur taille d'origine respective ? On est donc confronté à un manque de prédictibilité de la manipulation directe lorsque l'on interagit avec plusieurs objets ou des groupes d'objets, il s'agit du défi de multiplicité.

La manipulation directe ne permet pas d'interagir avec des « propriétés intangibles », c'est-à-dire des propriétés abstraites qui n'ont pas de représentation visuelle, telle que l'espacement entre deux objets. Ces propriétés sont manipulables indirectement, avec des outils, des menus, ou bien des boites de dialogue, en revanche il n'existe pas d'interactions permettant de les manipuler directement.

Plusieurs travaux ont pour but de pallier les limites de la manipulation directe. Frohlich propose un remaniement de la manipulation directe, la « manipulation élégante » ou gracieuse, en réintroduisant des parties de l'interaction conversationnelle. (Beaudouin-Lafon, 2000) étend les principes de la manipulation directe et expose un principe d'interaction reposant sur l'utilisation instruments. Des travaux récents proposent un mode d'interaction avec des substituts d'objets (appelés « surrogates ») afin de contourner les limites de la manipulation directe (Kwon et al., 2011).

2. L'interaction instrumentale et principes de conception

Pour interagir avec des objets du monde réel, nous utilisons au quotidien des outils, ou instruments. L'interaction instrumentale, est un paradigme d'interaction qui transpose cette métaphore de l'outil au monde informatique : l'utilisation d'instruments pour interagir avec des objets d'intérêt.

a) L'interaction instrumentale

L'interaction instrumentale étend le principe de la manipulation directe en réifiant des commandes en instruments d'interactions (Beaudouin-Lafon, 2000). L'instrument d'interaction est un intermédiaire entre l'utilisateur et les objets d'intérêts : on agit sur l'instrument, qui agit sur l'objet (Figure 17). Cela suit une métaphore du monde physique, où l'interaction avec les objets du quotidien se fait à l'aide d'instruments (stylos, poignées, marteau etc.). Des exemples d'instruments communs des systèmes interactifs sont les éléments de menus, les boutons des palettes d'outils, les poignées de redimensionnement, les barres de défilement… Une barre de défilement est un objet visible, et manipulable par l'utilisateur qui permet indirectement de manipuler le document en le faisant défiler.

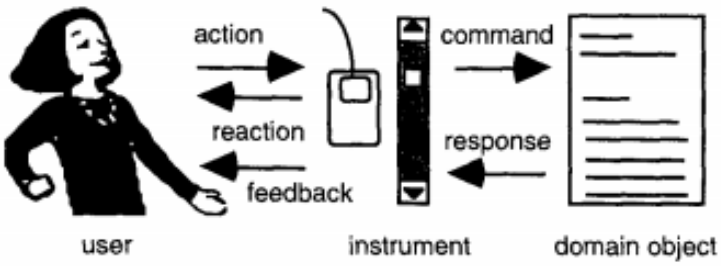


Figure 17. Illustration de la métaphore de l'interaction instrumentale entre un utilisateur et l'objet du domaine (Beaudouin-Lafon, 2000, fig. 1).

Les instruments ont des propriétés permettant d'évaluer et comparer des designs alternatifs : le degré d'indirection est une mesure du décalage spatial et temporel générés par un instrument. Par exemple, des poignées ont une faible distance spatiale alors que les boites de dialogues ont une distance importante. Il faut noter qu'une distance spatiale importante n'est pas toujours indésirable, comme le montre l'exemple d'un interrupteur servant à allumer ou éteindre la lumière donnée par Beaudouin-Lafon : monter sur une chaise pour atteindre l'ampoule à chaque fois que l'on souhaite l'allumer serait plus contraignant qu'utiliser l'interrupteur.

Le décalage temporel mesure le temps de réponse entre l'action sur l'instrument et le retour de l'objet. Il est instantané lors d'un appui sur la flèche d'une barre de défilement mais il est long pour une boîte de dialogue car les changements ne se

produisent qu'après l'appui sur le bouton « Ok » (qui applique les changements en fermant la boite de dialogue) ou « Appliquer » (applique les changements mais ne ferme pas la boîte de dialogue).

Le degré d'intégration mesure le ratio entre les degrés de libertés fournis par la part logique de l'instrument et ceux capturés par l'instrument d'entrée. Ainsi, le degré d'intégration d'une barre de défilement monodimensionnelle contrôlée par une souris qui agit dans un espace bidimensionnel est de ½. Faire défiler un document est plus efficace avec un panner 2D que deux barres de défilement car dans ce cas, le degré d'intégration vaut 1, tandis que celui des barres de défilement est à ½ et requiert alors des actions supplémentaires.

Pour finir, le degré de compatibilité mesure la similarité entre les actions physiques et la réponse de l'objet. Une opération de « drag&drop », ou glisser-déposer, a un degré élevé puisque l'objet suit les mouvements de la souris. En revanche, faire défiler un document avec des barres de défilement à un degré faible puisque l'appui sur le bouton du bas fait défiler le document vers le haut (Beaudouin-Lafon, 2000).

Le paradigme de la manipulation directe définit des interactions dont le degré d'indirection est minimal puisque l'utilisateur agit physiquement et directement sur un objet d'intérêt. Pour cette raison, la manipulation directe n'est pas assez efficace pour manipuler plusieurs objets de façon synchrone. En revanche, l'interaction instrumentale permet de réaliser des instruments pouvant agir sur plusieurs objets en introduisant une distance spatiale plus élevée. C'est le cas des « surrogates » (Kwon et al., 2011) par exemple, des substituts d'interaction. Il s'agit d'instruments pouvant agir sur plusieurs objets d'intérêts à la manière d'une « poupée vaudou » : on manipule le surrogate, ce qui modifie les objets concernés. L'interface de CPN2000 (un outil d'édition de réseaux de Pétri) s'affranchit totalement du principe de sélection ainsi que d'interactions introduites par le paradigme WIMP (boîtes de dialogues, menus, barres de défilement) (Beaudouin-Lafon et al., 2001). Toutes les interactions reposent sur l'utilisation d'instruments qui permettent de manipuler un ou plusieurs objets à la fois, comme les guides magnétiques.

b) Principes de conception : Réification, Polymorphisme, Réutilisation

(Beaudouin-Lafon & Mackay, 2000) présentent trois principes pour la conception d'interfaces visuelles. La réification est le processus qui transforme des concepts en objets, par exemple : dans un éditeur graphique, le concept de création de cercle est réifié en un instrument représenté par l'image d'un cercle dans une palette d'outil, qui

une fois sélectionné, est « tenu en main » par l'utilisateur. La réification crée de nouveaux objets qui peuvent être manipulés par l'utilisateur, ainsi elle augmente l'ensemble des objets d'intérêts.

Le polymorphisme est une propriété qui permet à une commande simple de s'appliquer sur des objets de nature différente, ce qui a pour avantage de maintenir un nombre limité de commandes. Il s'agit d'une propriété essentielle permettant de garder des interfaces simples, tout en augmentant son pouvoir. En guise d'exemple : les commandes de copier/coller sont polymorphiques puisqu'elles peuvent s'appliquer aussi bien à du texte qu'à une image ou encore des fichiers.

La réutilisation est un principe qui implique une entrée précédente, une sortie précédente, voire les deux afin de les rendre accessibles pour une utilisation future. Ce concept se combine bien avec le polymorphisme. Cela permet de « capturer » des schémas d'utilisation sous la forme d'objets et de commandes, et de les réappliquer avec des paramètres différents. Par exemple, dans un outil disposant des fonctionnalités *annuler* et *refaire*, la commande *refaire* peut être réutilisée indéfiniment afin de reproduire la dernière action réalisée autant de fois que nécessaire.

3. Synthèse sur les paradigmes d'interaction

La manipulation directe et l'interaction instrumentale sont des techniques d'interaction efficaces pour manipuler des objets individuels : elles réduisent le nombre d'actions requises par rapport à d'autres techniques telles que les lignes de commandes, le dialogue conversationnel ou les interactions modales. Mais étant inspirés de métaphores du monde réel, ces principes en ont également certaines limites : l'accès à des objets significativement trop petits ou non visibles implique un effort de *recherche* et entrave la capacité de *désignation*, le manque de prédictibilité des interactions avec des objets multiples est limitant pour la *conception exploratoire*, et l'impossibilité de manipuler des propriétés non-réifiées (comme l'espacement entre deux objets) induit un *pouvoir d'action* limité.

Les principes de conception, en lien avec l'interaction instrumentale, tels que la réification (transformer un objet en un concept manipulable), le polymorphisme (appliquer une même opération à différentes classes d'objets) et la réutilisation (de sélections précédentes et de résultats d'interactions) permettre d'étendre le pouvoir d'action d'un utilisateur à des objets multiples. Nous nous appuyons sur les principes de conception et ceux de la manipulation directe et de l'interaction instrumentale, afin d'élaborer des techniques d'interaction pouvant répondre aux besoins des utilisateurs

que nous avons identifiés. Pour cela nous avons au préalable étudié des techniques d'interaction existantes afin d'analyser en quoi elles ne couvrent pas intégralement ces besoins.

C. Techniques d'interaction

Dans les éditeurs graphiques traditionnels, il existe des techniques d'interaction qui permettent à l'utilisateur de structurer son travail et ainsi d'interagir avec un ensemble d'objet à la fois. Les groupes, par exemple, permettent d'appliquer des opérations à l'ensemble des objets le constituant. Qu'elles soient WIMP ou Post-WIMP[4], nous nous sommes intéressés aux techniques qui offrent un plus grand pouvoir d'action et qui favorisent la conception exploratoire.

1. Techniques d'interaction pour manipuler plusieurs objets

Il est courant d'utiliser, dans les éditeurs graphiques traditionnels, des outils qui permettent à l'utilisateur d'agir sur plusieurs objets à la fois, tels que les groupes (utilisables dans les outils de dessin, par exemple Illustrator) qui permettent de réaliser des opérations sur tous les objets du groupe, les styles des éditeurs de texte, ou encore les masters (par exemple des masques de diapositive dans PowerPoint).

a) Groupes

Les utilisateurs peuvent créer des groupes d'objets à partir d'un ensemble d'objets préalablement sélectionnés et agir directement sur le groupe. Cela permet de modifier une propriété donnée de chacun de ses consistants en agissant qu'une seule fois sur le groupe. « Les groupes permettent aux utilisateurs de travailler à un plus haut niveau d'abstraction, ce qui augmente le pouvoir de l'interface » (Beaudouin-Lafon & Mackay, 2000). Cependant, la seule opération disponible pour un groupe est celle qui permet de dégrouper ses objets (dégrouper a pour effet de sélectionner tous les objets après avoir supprimé le groupe). Il est donc possible de créer des groupes et de les détruire, mais pas de les modifier : il n'est pas possible d'ajouter ou de supprimer un élément par exemple. Pour ce faire, il est donc obligatoire de supprimer le groupe, et de le recréer sans les éléments qui étaient à supprimer, et avec ceux qui devaient être ajoutés. Ce processus est donc coûteux puisque le nombre d'actions à effectuer pour interagir avec le groupe n'est pas minimal.

[4] Les techniques Post-WIMP sortent du carcan « fenêtres, icones, menus et dispositifs de pointage » en proposant des interactions basées sur la gestuelle comme les « Marking Menus », des outils semi-transparents et déplaçables, ou encore l'interaction bi-manuelle. Les techniques Post-WIMP se veulent plus concises et directes que les techniques WIMP (Dragicevic, 2004).

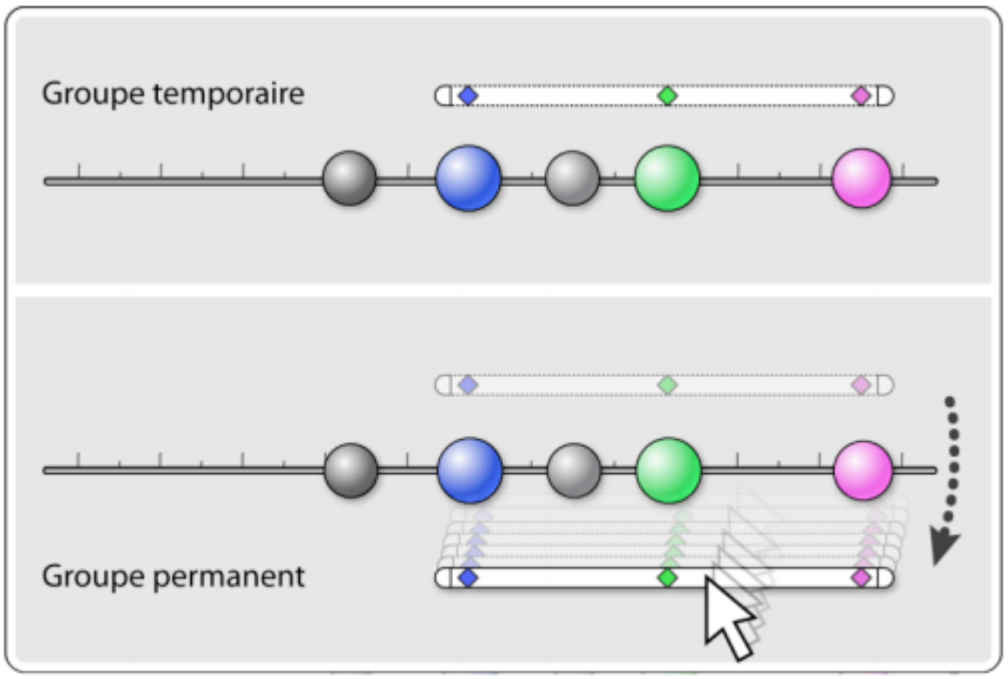


Figure 18. Transformation d'un groupe temporaire en groupe permanent dans Kabuki. (Tabart, Conversy, Vinot, & Athènes, 2009, fig. 4). L'utilisateur saisit le groupe temporaire et le fait glisser dans une partie de l'interface où il sera sauvegardé en tant que groupe permanent.

Kabuki, un outil d'aide à la conception de rendu graphiques, permet de créer des groupes de propriétés graphiques afin de modifier ces propriétés en minimisant les interactions (Tabart et al., 2009). Certains groupes ne sont utiles que le temps de réaliser des opérations sur ses objets, c'est pourquoi Kabuki offre la possibilité de créer des groupes temporaires, ou transitoires (durant quelques secondes). Ces groupes de nature éphémère ne perdureront que le temps nécessaire pour réaliser ces opérations. Par ailleurs il est possible de transformer un groupe temporaire en groupe permanent et vice-versa à l'aide d'une interaction de glisser-déposer (Figure 18). Ces notions de groupes temporaires et permanents étendent la capacité d'action sur les groupes et facilitent la structuration d'une scène.

La sélection peut se voir comme un groupe éphémère, dans le sens où elle ne perdure que jusqu'à la prochaine sélection. Ce type de groupe est doté des fonctionnalités d'ajout et de retrait qui se font traditionnellement en maintenant la touche « shift » ou la touche « ctrl » du clavier et en cliquant sur les objets individuellement. Certains outils offrent la possibilité de créer des ensembles hétérogènes, mais limitent les opérations à des propriétés spécifiques, par exemple avec des translations, rotations, déformations : tous les éléments sont transformés selon la transformation spécifiée. Utiliser des groupes n'est alors pas plus compliqué qu'utiliser des objets individuels dès l'instant ou une même interaction peut s'appliquer sur l'un ou l'autre. En revanche dans certains

logiciels, d'autres opérations (par exemple, changer la couleur) peuvent ne pas être réalisées sur un groupe, vraisemblablement parce que l'opération ne peut pas s'appliquer à certains des éléments du groupe (l'opération est alors « grisée » pour indiquer qu'elle ne peut pas être utilisée). Ceci force l'utilisateur à dégrouper et à appliquer l'opération sur chacun des objets. Dans ce cas de figure, l'interaction avec la structure n'est pas bien intégrée avec l'interaction avec le contenu.

Le concept de groupe est intéressant dans le sens où il résulte d'une combinaison de *réification* et de *polymorphisme*, tout en encourageant la *réutilisation* puisque cela induit une intention de travailler ultérieurement avec cet ensemble d'objets (Beaudouin-Lafon & Mackay, 2000). Dans CPN2000, un éditeur de réseaux de Petri ayant pour particularité de ne pas disposer de mécanisme de sélection (Beaudouin-Lafon et al., 2001), plusieurs aspects prennent avantage de la notion de groupe : les « folders » représentent des groupes de pages, les guides magnétiques forment des groupes d'objets selon des contraintes géométriques (Figure 19), et les styles des groupes d'objets partageant des propriétés graphiques.

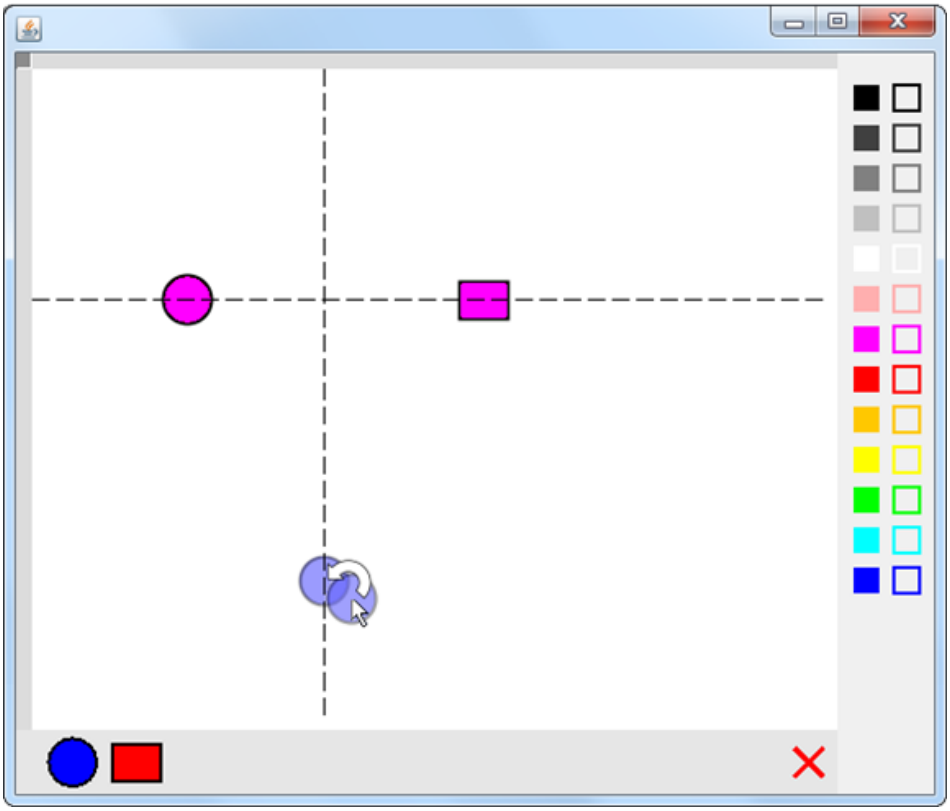


Figure 19. Exemple d'interface utilisant les guides magnétiques. Le cercle bleu déplacé par la souris est attiré par le guide magnétique vertical à proximité. Une opération peut être appliquée à un guide afin de modifier tous les objets attachés, par exemple, la couleur magenta a été donnée au guide horizontal ce qui a modifié le cercle et le rectangle.

Des groupes peuvent être inclus dans d'autres groupes, créant ainsi des arbres ou des hiérarchies. Il n'y a cependant que très peu de support à la gestion de ces hiérarchies, à la navigation dans la hiérarchie de parents (Maloney & Smith, 1995) et aux structures arborescentes dans les éditeurs graphiques (tels que Inkscape or Illustrator). Une « treeview » permet à l'utilisateur de rétablir des liens de parenté entre éléments avec du glisser-déposer, cependant d'autres opérations ne sont pas autorisées, comme appliquer une couleur à un nœud afin de changer tous les fils.

b) Masters et styles

Les masters sont des éléments qui servent de modèles à d'autres. Par exemple, PowerPoint permet aux utilisateurs de définir une diapositive « master » dont l'apparence est héritée par toutes les autres diapositives liées au master. Le concept de master a été introduit par Sketchpad, comme des objets pouvant être disposés à de multiples emplacements dans la scène (Sutherland, 1964). Changer une des propriétés du master a pour effet de propager ce changement à tous les objets qui en dépendent (Figure 20). Cette technique réduit ainsi le nombre d'actions requises par l'utilisateur lorsqu'un même changement doit être effectué sur plusieurs objets. De même, Kabuki (Tabart et al., 2009) permet de spécifier lors de la duplication d'une couleur s'il s'agit d'une copie, ou d'un « clonage ». La modification ultérieure d'une copie ne modifie pas l'objet copié, alors que la modification d'un clone modifie l'ensemble des clones liés.

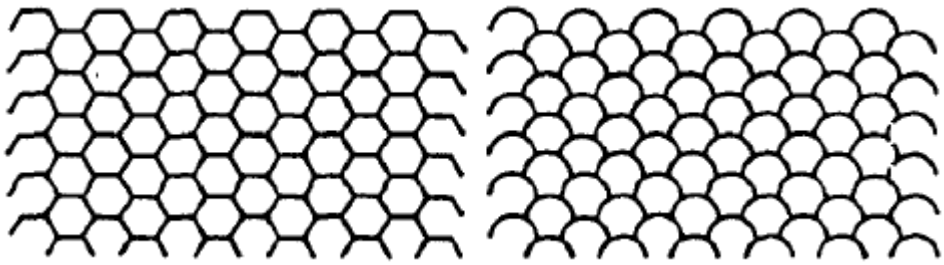


Figure 20. A gauche, une scène remplie d'hexagones dans Sketchpad (Sutherland, 1964, fig. 4). L'hexagone « master » est modifié en cercle, transformant tous les autres hexagones également (scène de droite).

Un style est une entité qui définit des ensembles de propriétés qui sont attribuées à tous les éléments partageant le style (par exemple : styles de mise en page d'un éditeur de texte, style de mise en forme de pages web comme le langage CSS). Les styles peuvent s'apparenter à des masters, dans le sens où ils définissent également un modèle partagé par d'autres éléments. Ainsi, modifier un style (par exemple une police et une taille de caractères) modifie également tous les éléments qui dépendent de ce style (par exemple des titres de section). Généralement, les styles ne sont utilisés qu'à

des fins de mise en page, et permettent de modifier l'apparence d'un document entier ou d'une page web en quelques actions. A la différence d'un master, qui est représenté par un objet graphique manipulable, un style est sous forme de code (Figure 21, à gauche) ou de champs de formulaires inclus dans des fenêtres modales (Figure 21, à droite). Les styles et les masters permettent de structurer une scène ou un document et offrent ainsi à l'utilisateur la capacité d'agir sur plusieurs éléments en une seule fois.

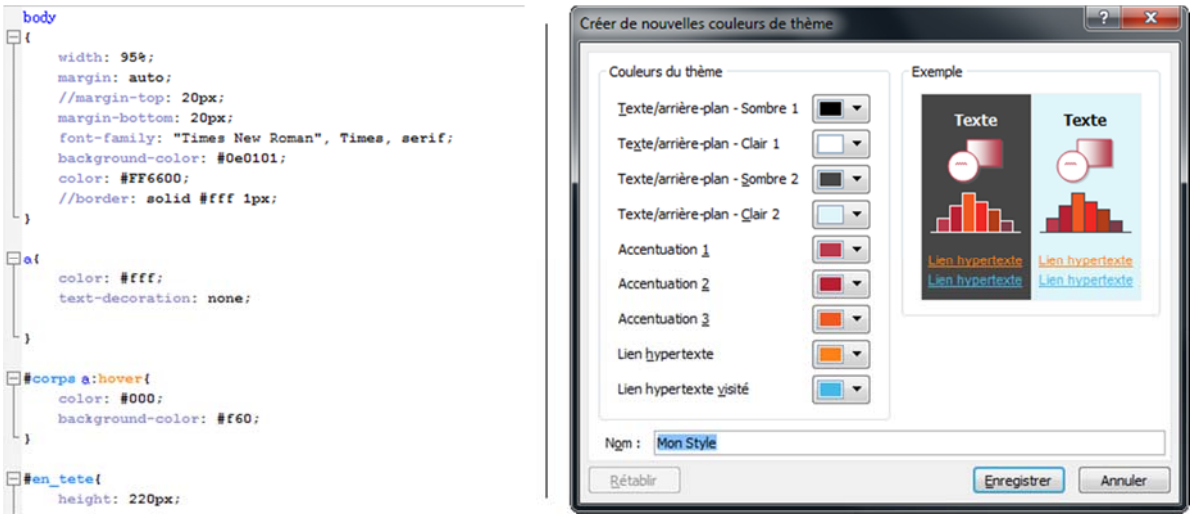


Figure 21. Exemple de feuille de style CSS (à gauche) et de style dans Word 2010 (à droite).

c) Techniques d'interaction post-WIMP : toolglasses

Dans la mouvance de l'interaction post-WIMP, des chercheurs se sont intéressés à des techniques d'interactions pouvant augmenter le pouvoir d'action (Cechanowicz & Gutwin, 2009). Les « toolglasses » (Bier, Stone, Pier, Buxton, & DeRose, 1993) en sont un exemple. Basée sur de l'interaction bi-manuelle, elles consistent en l'utilisation d'instruments déplaçables et transparents. Un des intérêts premiers est d'augmenter la visibilité, qui est, comme nous avons pu le voir, une dimension cognitive importante pour les activités de modification et d'exploration (Figure 15). Par exemple, l'utilisateur peut voir le contenu du presse-papier dans la toolglass lors d'un copier/coller. Par ailleurs, il devient même possible de créer des masters et de les instancier, en créant des librairies de formes et de propriétés grâce à une interaction sur l'outil (Figure 22). On dispose alors d'un outil permettant d'établir des structures avec un degré d'indirection plutôt faible.

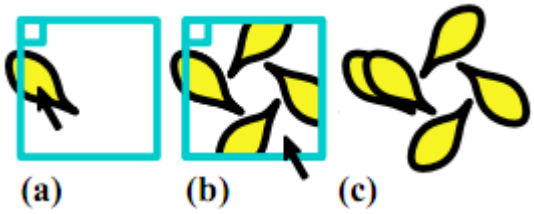


Figure 22. Outil qui permet de sélectionner un objet dans la toolglass (a), de le modifier (rotation) et d'en faire des copies (b) et de copier et coller cet ensemble d'objets (c) (Bier et al., 1993, fig. 6).

En amenant l'outil directement à l'objet, il est possible de changer ses propriétés localement (Figure 23). L'avantage réside dans le fait que cela évite à l'utilisateur d'aller sélectionner un outil de type « pipette » dans une palette extérieure (Beaudouin-Lafon et al., 2001)(Appert, Beaudouin-Lafon, & Mackay, 2005). Cependant, de par la nature de cette méthode, il n'est possible d'interagir qu'avec un seul objet à la fois.

Figure 23. En amenant la toolglass sur un objet, il est possible de prélever sa couleur (a). La toolglass est ensuite amenée à un autre objet (b) auquel on applique la couleur prélevée (c) (Bier et al., 1993, fig. 7).

Les lentilles de prévisualisation permettent d'obtenir un aperçu du changement d'une propriété visuelle (Figure 24). Cela favorise l'expérimentation à court-terme, puisque l'on peut alors comparer deux états d'un objet et évaluer le résultat du changement. L'expérimentation à court-terme est un besoin important pour la conception exploratoire (Terry & Mynatt, 2002a).

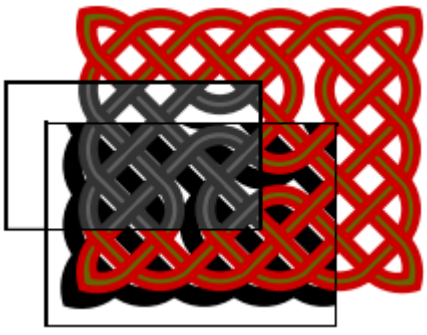


Figure 24. Deux lentilles survolent l'entrelacement dessiné. La première montre une prévisualisation d'une opération d'ajout d'ombre portée, la seconde une prévisualisation d'une coloration en niveau de gris (Bier et al., 1993, fig. 8).

Les toolglass offrent aussi des services facilitant la sélection d'objets superposés, en permettant par exemple de cliquer sur leurs sommets qui deviennent visibles au survol de la lentille, ou en modifiant l'échelle des objets sous celle-ci.

Les « toolglasses » sont donc des outils intéressants surtout parce qu'elles augmentent la visibilité, et parce que l'interaction est directe : on n'utilise pas de commande ou de boîtes de dialogues. Il faut amener l'outil à l'objet, ce qui nous amène cependant aux mêmes limitations données par la manipulation directe en raison de sa courte distance sémantique et articulatoire : cela ne permet pas d'interagir efficacement sur plusieurs objets à la fois (Frohlich, 1993).

d) Graphical Search & Replace

Puisque la structuration préalable peut poser certains problèmes (nécessité de prévoir, travail supplémentaire, expertise requise) freinant la conception exploratoire, des techniques permettent de modifier plusieurs objets en spécifiant des propriétés, comme la fonction « chercher et remplacer » des éditeurs de texte.

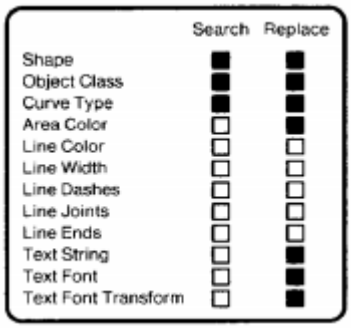


Figure 25. Paramètres sélectionnables que le système peut chercher et remplacer (Kurlander & Bier, 1988, fig. 8).

Le « Graphical Search & Replace », fonctionne similairement au « chercher et remplacer » d'un éditeur de texte (Kurlander & Bier, 1988). Cette technique alternative permet l'édition cohérente d'une illustration sans structuration préalable de celle-ci. L'utilisateur définit un ensemble de formes synthétiques, un ensemble de propriétés de styles (contour, couleur de fond etc.) et un ensemble de paramètres de recherches (tolérance à l'erreur, considération des rotations etc.) (Figure 25). Il définit ensuite son patron de remplacement (avec des ensembles de paramètres similaires) et l'éditeur cherche dans le dessin les objets qui correspondent et effectue les remplacements (Figure 26).

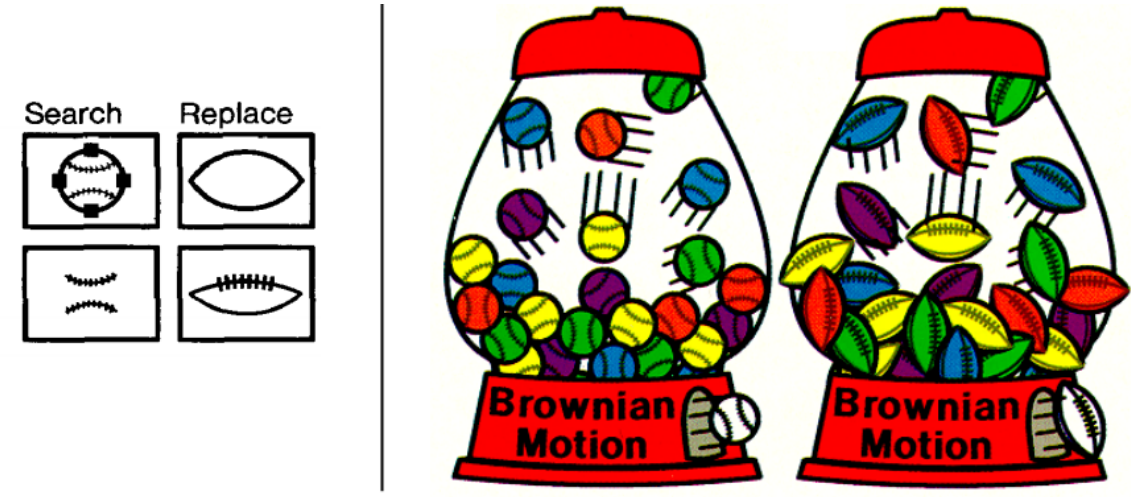


Figure 26. Exemple d'utilisation du Graphical Search & Replace : à gauche, l'utilisateur définit des figures à rechercher et leurs figures de remplacement. A droite, la scène originale et la scène après remplacements (Kurlander & Bier, 1988, fig. 10 et fig. 11).

La recherche graphique a de nombreuses applications : elle peut être mise à profit pour pouvoir justement éditer plusieurs objets en une seule fois, tout en ne freinant pas la conception exploratoire puisqu'il n'est pas nécessaire de créer des abstractions. Cela peut également servir, dans le cas où le patron de remplacement est plus élaboré que celui à remplacer, à faire des changements multiples (en utilisant le patron comme un modèle) pour obtenir une composition plus élaborée (Figure 28). Pour aller plus loin il est même possible de faire de la récursivité si le patron de remplacement contient le patron recherché. La recherche graphique est liée à d'autres sujets, elle peut par exemple être vue comme une interface d'une base de données graphique où l'on doit faire des requêtes sur des objets dont on doit modifier des propriétés visuellement représentées, ou comme une méthode de conception de grammaires générant des figures récursives (Figure 28).

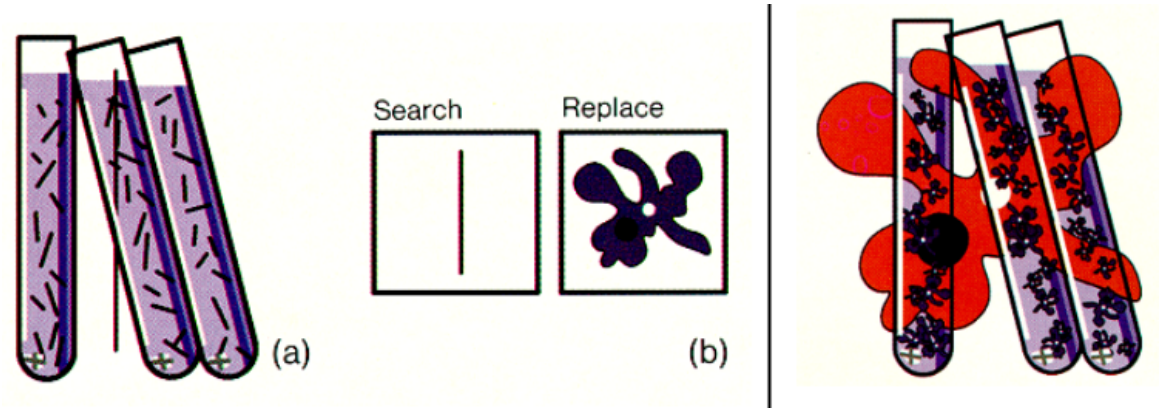


Figure 27. Utilisation de l'outil pour remplacer des lignes d'un dessin (a) en une forme plus élaborée (b) à gauche, résultat à droite (Kurlander & Bier, 1988, fig. 18 et fig. 19).

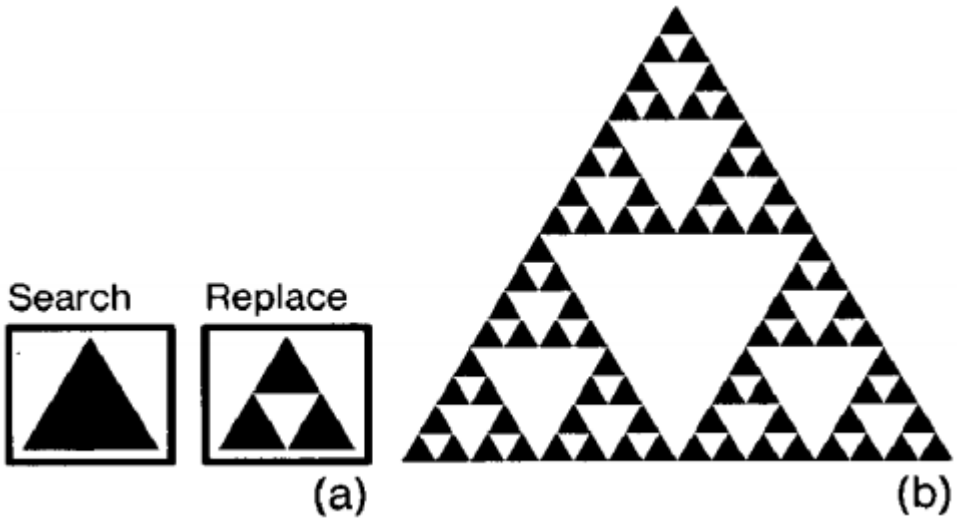


Figure 28. Exemple de récursivité. L'utilisateur demande à remplacer les triangles noirs par trois triangles plus petits (a). Le remplacement est effectué sur un grand triangle quatre fois (b) (Kurlander & Bier, 1988, fig. 17).

Si cette fonctionnalité s'avère puissante en tant qu'outil aidant la modification et l'exploration, son utilisation reste contraignante. En effet, son utilisation repose sur des boîtes de dialogue, elle est donc peu instrumentée et n'offre pas de possibilité de manipulation directe. Son degré d'indirection est donc élevé.

e) Surrogates

(Kwon et al., 2011) étendent les idées de la manipulation directe et de l'interaction instrumentale, en présentant l'interaction avec des « surrogates » ou substitut d'objets. Les surrogates sont des instruments d'interactions spécialisés qui permettent à l'utilisateur d'interagir avec un substitut d'un objet du domaine au lieu de l'objet lui-même. L'intérêt est d'augmenter l'accès aux objets et la capacité à en manipuler plusieurs. Le concept des surrogates est similaire au principe des « poupées vaudous » : on propage le résultat des actions réalisées sur la poupée sur les objets du domaine. Les surrogates sont utiles pour la manipulation d'objets 3D, présents dans les jeux vidéo (Figure 30) (où il est plus simple de modifier le surrogate qu'un avatar en 3D directement) et les éditeurs graphiques sous la forme d'inspecteurs exposant les propriétés accessibles des objets et permettant de les modifier (Figure 29).

La manipulation directe d'objets simples peut se voir entravée si l'espace visuel est limité ou bien le nombre d'attributs associés à l'objet est important. L'avantage d'un surrogate est qu'il se trouve externalisé, dans un espace différent des objets du domaine, et au prix d'une indirection spatiale supplémentaire (Beaudouin-Lafon, 2000), il devient facile d'accéder aux propriétés de l'objet et de les manipuler. Le surrogate devient un méta-objet pouvant disposer de plus d'espace que l'objet du domaine. Un

surrogate peut se voir lié à plusieurs objets. Un changement appliqué au surrogate est alors propagé à tous les objets liés. La manipulation directe sur plusieurs objets étant problématique de par son manque de prédictibilité, l'interaction via des surrogates rend ainsi la manipulation de plusieurs objets plus facile à prévoir.

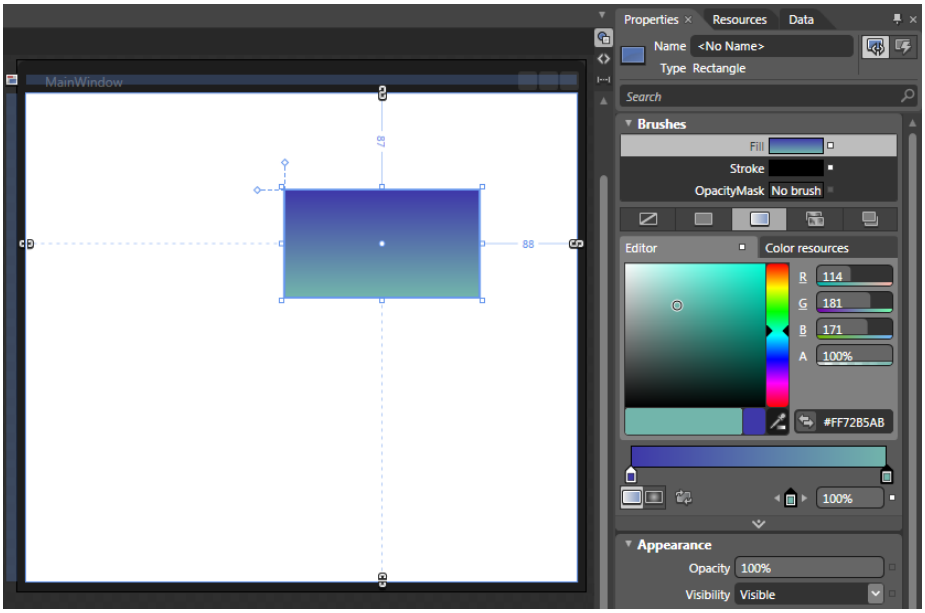


Figure 29. Inspecteur de propriété du logiciel Microsoft Blend. Plutôt que de manipuler l'objet dans la scène directement pour éditer certaines de ses propriétés, comme le gradient de couleur, l'utilisateur interagit avec le surrogate présent dans l'inspecteur.

Figure 30. Exemple d'interaction par les surrogates dans un jeu vidéo (Torchlight II). A gauche, l'avatar 3D, à droite, l'interface contenant le surrogate permettant de modifier l'avatar. L'utilisateur doit glisser et déposer l'équipement de son personnage dans des emplacements prévus à cet effet pour équiper son avatar 3D.

2. Techniques d'interaction améliorant la conception exploratoire

Si les éditeurs graphiques d'aujourd'hui permettent aux utilisateurs d'explorer des solutions, de faire des erreurs, de les corriger, et s'ils permettent le raffinement d'un design, les mécanismes sous-jacents ne sont pas toujours cohérents et l'interaction pour le faire peu éprouvée : les fonctionnalités existent, mais sont détournées de leur fonction première. Par exemple, le mécanisme de « l'undo/redo » qui permet d'annuler ou refaire une action est un mécanisme qui assure à l'utilisateur la possibilité de corriger ses erreurs et ainsi de prendre peu de risques en explorant des solutions. Les actions réversibles permettent aux utilisateurs de ne pas avoir peur de dégâts potentiels (Shneiderman, 1987). Cependant, cette fonctionnalité est aussi utilisée pour faire des comparaisons entre deux états d'une scène : en annulant une action et en la rejouant, cela permet d'observer le résultat de cette action et de l'évaluer (Terry & Mynatt, 2002a). Un autre exemple donné par Terry et Mynatt concerne l'exploration de variation : s'il est possible d'explorer plusieurs versions d'un travail, c'est généralement à la charge de l'utilisateur de gérer lui-même ses différentes versions, en dupliquant des fichiers dans son système d'exploitation. Il s'agit là encore d'un exemple de fonctionnalité nécessaire dénuée de services interactifs qui lui sont propres.

a) Annuler/refaire

Bien que tout éditeur (textuel ou graphique) moderne supporte une notion d'historique permettant aux utilisateurs de corriger une erreur récente ou de comparer deux états de la conception, ils sont limités par une exploration causale et séquentielle de l'historique (Scull et al., 2009). Même si la visibilité des historiques a été améliorée, présents dans certains outils sous forme d'une liste des actions réalisées, leur implémentation reste sous forme d'empilement imposant un déroulement linéaire (Vitter, 1984), l'utilisateur n'ayant que peu de contrôle hormis celui de revenir plus rapidement à un état précédent.

Selon (Scull et al., 2009) le processus créatif est non-linéaire et devrait être supporté par des outils non-linéaires également. Leurs travaux présentent une nouvelle visualisation montrant les actions dans leur contexte et permettant une exploration non-linéaire de l'historique sous la forme de story-boards interactifs (Figure 31). Il s'agit d'un mode nouveau d'interaction avec l'historique d'un document activable à tout moment pendant l'édition de celui-ci. Les actions sont représentées selon leur nature (translation, rotation, changement de couleur etc.) par une petite image à proximité de l'objet concerné. Les actions sont considérées dans un contexte spatial plutôt que dans

une liste où elles sont difficilement connectées aux objets, elles deviennent des instruments ayant un degré d'indirection faible (Beaudouin-Lafon, 2000).

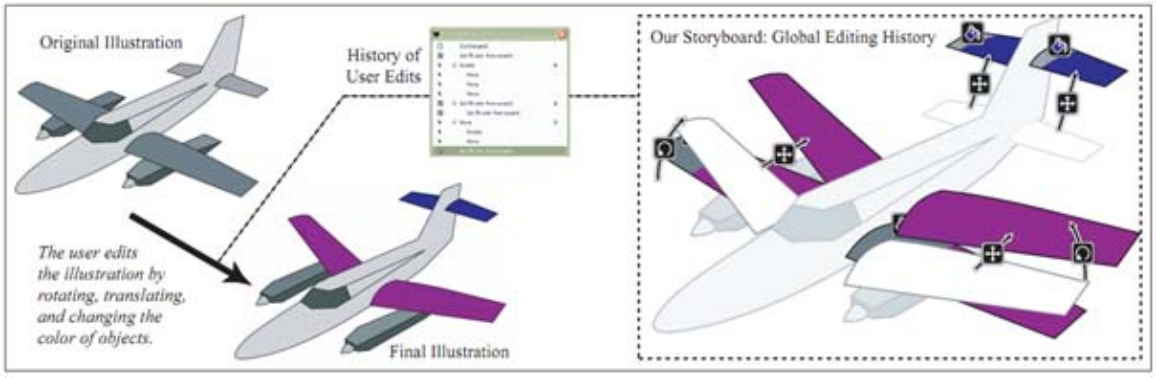


Figure 31. Visualisation des actions locales à chaque objet. La scène courante est à gauche, et à droite se trouve la vue de l'historique. Les flèches et les icônes montrent les opérations de translation, de rotation et de changement de couleur réalisées par l'utilisateur. En cliquant sur un icône, l'utilisateur peut annuler l'action correspondante (Scull et al., 2009, fig. 1).

Les story-boards encouragent la conception exploratoire en permettant à l'utilisateur d'accéder à des actions précédentes dans un contexte spatial et d'annuler sélectivement celles-ci. L'undo devient non-séquentiel, ce qui est moins destructeur dans le sens où l'utilisateur ne perd pas toutes les actions réalisées entre l'état courant et celui qui était du moment de l'action annulée.

Des historiques graphiques interactifs augmentent la visibilité de l'utilisateur sur les actions précédemment réalisés (Kurlander & Feiner, 1988). Même s'il s'agit d'un historique linéaire, cela favorise l'exploration puisque l'on peut avoir un aperçu de l'état d'un document à différents moments et les comparer avec l'état actuel. Qui plus est, l'historique étant interactif, l'utilisateur peut modifier des objets d'une scène passée avec des répercussions sur les objets de la scène actuelle.

Kurlander et Feiner présentent dans leur article l'exemple suivant : l'utilisateur dessine un visage de profil et en crée trois autres copies. Il aimerait ensuite pouvoir colorer l'œil de chaque visage d'une même couleur. Même s'il reste possible de sélectionner chacun des objets un par un pour leur appliquer l'opération (car ils ne sont qu'au nombre de quatre), cela reste une opération qui peut s'avérer coûteuse dans le cas où les objets seraient plus nombreux ou dispersés. Grâce à l'historique éditable, l'utilisateur modifie dans l'historique la couleur du premier visage dessiné avant qu'il ne soit dupliqué, ainsi, les copies ultérieures bénéficient de la même modification (Figure 33).

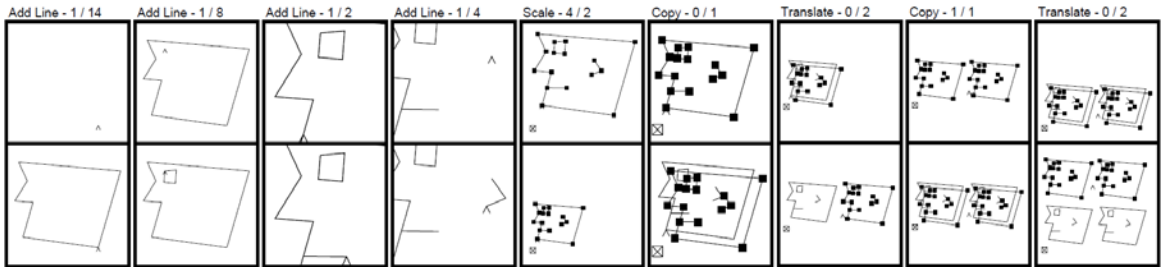


Figure 32. Historique graphique d'une scène dans laquelle quatre visages ont été dessinés (Kurlander & Feiner, 1988, fig. 3).

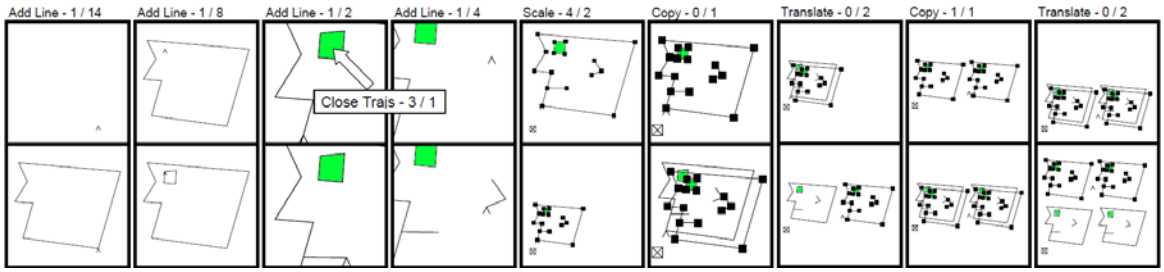


Figure 33. L'utilisateur sélectionne le panel suivant la création de l'œil, le colorie, puis restore les opérations suivantes (Kurlander & Feiner, 1988, fig. 5).

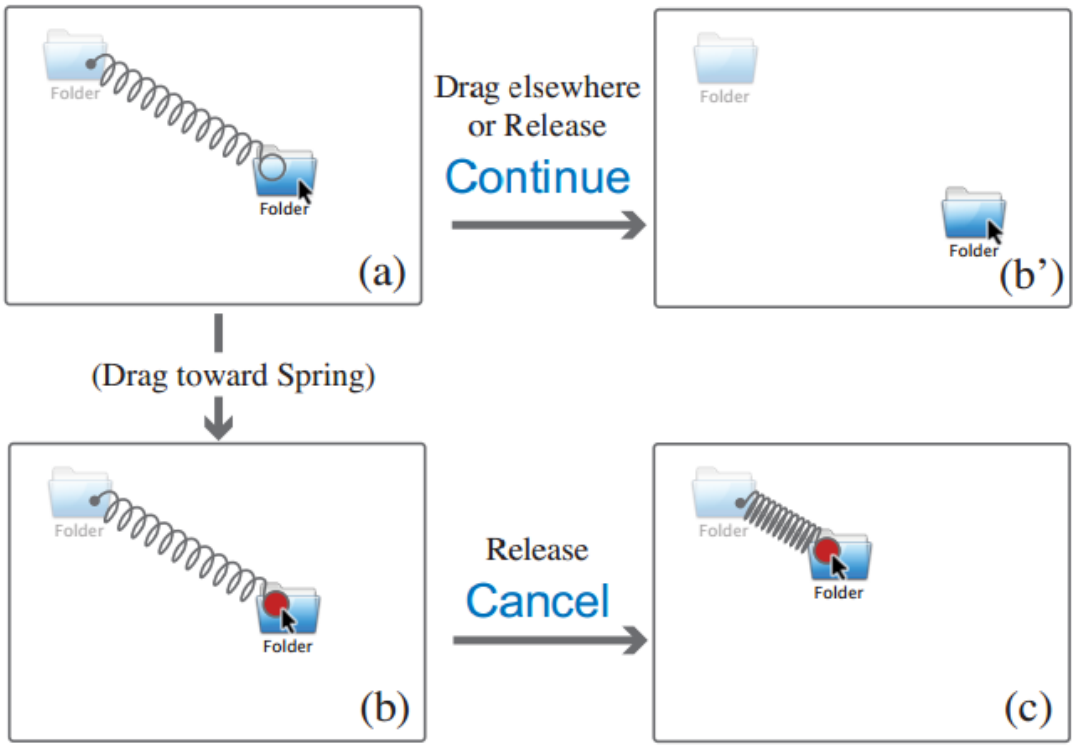


Figure 34. En déplaçant un objet, un ressort apparait et montre la position d'origine de l'objet (a). L'utilisateur peut terminer son opération (b) ou bien saisir une poignée présente à l'extrémité du ressort ce qui annule le déplacement (b') et renvoie l'objet à sa position d'origine (c') (Appert et al., 2012, fig. 2).

Dwell-and-spring (Appert, Chapuis, & Pietriga, 2012) est un exemple de technique d'interaction qui permet de faire de l'undo/redo par manipulation directe. Cette technique permet d'annuler toute conséquence d'une manipulation directe de type glisser-déposer, même quand il s'agit d'opérations non supportées par les historiques comme le déplacement d'une fenêtre, une action sur une barre de défilement, etc. L'historique n'est pas linéaire, toute action peut être annulée indépendamment de la séquence d'actions dans laquelle elle a été effectuée : on annule la dernière action réalisé sur un objet d'intérêt, par manipulation directe (Figure 34). Dwell-and-sping utilise des instruments représentés par la métaphore du ressort.

a) Comparer, explorer

Les modes d'interaction actuels imposent une progression linéaire dans les tâches, ce qui est contradictoire avec la nature ouverte, désordonnée et expérimentale d'un processus créatif (Terry & Mynatt, 2002b). Bien que la prévisualisation de commande puisse permettre à l'utilisateur de voir quels seraient les effets de celle-ci avant de l'appliquer, les systèmes ne permettent de montrer qu'une seule prévisualisation à la fois. Side-Views (Terry & Mynatt, 2002c) est un outil pouvant fournir des prévisualisations persistantes et interactives de commandes. Réaliser des prévisualisations de multiples commandes, ou de plusieurs instances d'une même commande mais avec des paramètres différents, et avoir la possibilité de comparer côte-à-côte ces prévisualisations sont les objectifs de Side-Views pour améliorer la conception exploratoire.

Side-Views emploie la métaphore des infos-bulle pour fournir des prévisualisations dynamiques à la demande : lorsque l'utilisateur attend un court délai sur une commande d'intérêt, une fenêtre « pop-up » (info-bulle) est affichée, contenant la prévisualisation de la commande telle qu'elle serait appliquée au document courant (Figure 35 et Figure 36). S'il existe dans de nombreuses applications des outils permettant la prévisualisation de commandes, il faut noter qu'ils nécessitent généralement l'utilisation de boîte de dialogues modales (Terry & Mynatt, 2002b). Side-Views propose un instrument demandant donc moins de temps et moins d'actions pour accéder à cette fonctionnalité.

Une Side-View se fermera automatiquement après un court-délai, comme une info-bulle classique, mais elle peut aussi être rendue persistante en cliquant sur sa barre de titre. L'utilisateur peut donc instancier autant de vues qu'il le souhaite pour autant de commandes, ce qui lui permet de comparer plusieurs résultats de commandes en même temps. Les vues se mettent automatiquement à jour si le document est modifié

montrant alors le résultat qu'aurait la commande sur la version à jour du document. Lorsque la commande visualisée peut avoir différents paramètres, l'utilisateur peut s'il le désire, obtenir un spectre de paramètres : c'est-à dire une visualisation d'une série de prévisualisations selon la portée des valeurs pour chaque paramètre. Pour finir, il est également possible de faire des compositions de prévisualisations.

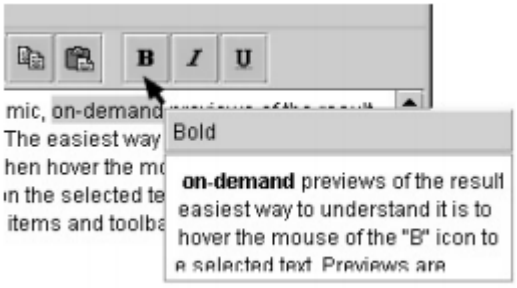


Figure 35. En survolant la commande « bold », une Side-View apparait et montre le résultat de l'application de la commande sur le texte sélectionné (Terry & Mynatt, 2002c, fig. 1).

Les Side-Views sont donc des outils mis au profit du travail créatif et de la conception exploratoire : l'utilisateur n'a pas de prédictions à faire, la persistance des comparaisons original/résultat lui permet d'explorer des alternatives et plusieurs solutions sans même changer la nature de son document. Il s'agit d'un « what-if tool », un outil qui permet d'expérimenter sur les données sans commettre aucun changement (Terry & Mynatt, 2002c).

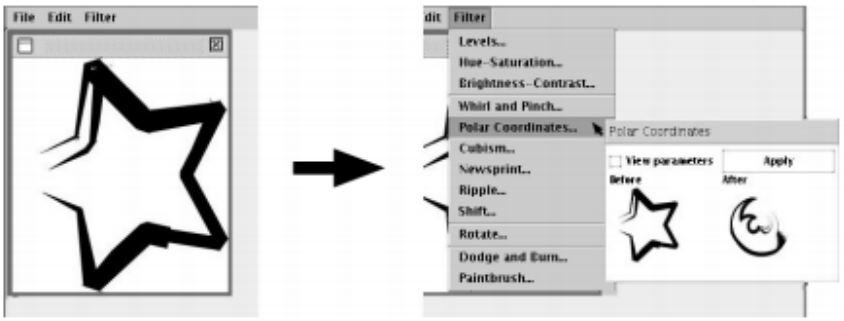


Figure 36. Exemple d'utilisation de Side-View pour la manipulation d'image. A gauche se trouve la fenêtre contenant l'image, à droite une Side-View est apparue en survolant le menu « Polar Coordinates » et montre quel serait le résultat de cette commande sur l'image (Terry & Mynatt, 2002c, fig. 2).

Parallel Paths est un modèle d'interaction qui facilite la génération et la manipulation de solutions alternatives (Terry et al., 2004), l'intérêt étant de pouvoir offrir un support à l'exploration de solutions alternatives, embarquées dans un même espace de travail afin de faciliter les comparaisons côte à côtes. Contrairement à Side-Views dont l'intérêt est de proposer des prévisualisations de résultats de commandes appliquées à

une même solution afin de comparer ces résultats, Parallel Paths à pour objectif de permettre aux utilisateurs d'avoir plusieurs solutions différentes et d'explorer en parallèle ces solutions. Ces solutions pouvant être par ailleurs manipulées individuellement ou collectivement : en considérant l'espace de variations comme une arborescence de créations, l'utilisateur peut interagir sur une solution ou bien sur plusieurs d'entre elles en agissant sur un nœud de cette arborescence. Ceci n'est pas possible en travaillant sur des fichiers dupliqués : on ne peut pas répercuter des modifications sur deux copies d'un même fichier en modifiant celui d'origine par exemple.

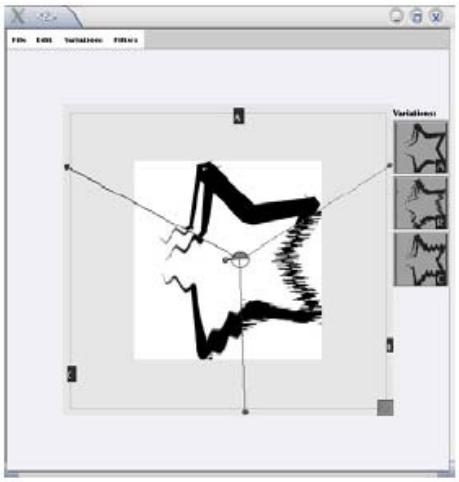


Figure 37. Exemple d'utilisation de Parrallel Pies. L'utilisateur peut explorer trois variations d'une étoile dans un même espace de travail (Terry et al., 2004, fig. 1).

Parallel Pies est un exemple de technique d'interaction utilisant le modèle Parallel Paths : il permet de diviser des zones d'un même espace de travail avec des axes manipulables directement (Figure 37). Chacune des zones dévoilant une variation de la solution selon les commandes qui ont été appliquées.

3. Synthèse sur les techniques d'interactions

Nous avons étudié différentes techniques qui permettent aux utilisateurs de *manipuler plusieurs objets*. La création de groupe est une fonctionnalité répandue dans les éditeurs graphiques : elle permet de considérer un ensemble d'objets comme une entité unique et ainsi d'étendre les modifications réalisées sur le groupe aux objets qui le constituent. Les masters et les styles sont des techniques qui permettent de structurer un document. En agissant sur un style ou un master, ce sont l'ensemble des éléments qui dépendent respectivement du style ou du master qui sont modifiés. Les opérations de *structuration* sur les groupes sont limitées : il est possible de détruire ou

de créer un groupe, mais on ne peut ajouter ou supprimer d'élément sans passer par des étapes intermédiaires de destruction ou création.

Les « toolglasses » (Bier et al., 1993) sont des instruments post-WIMP qui offrent des moyens d'interactions plus directs : elles *réduisent le nombre d'actions* requises pour *désigner* des opérations et des objets. En contrepartie de cette faible indirection spatiale, la portée des interactions est réduite à des objets individuels. Le « Graphical Search & Replace » (Kurlander & Bier, 1988) est une technique similaire aux fonctions « rechercher et remplacer » des éditeurs de texte. Elles offrent un *pouvoir d'action* important en permettant de *rechercher* des propriétés graphiques pour l'ensemble des objets d'une scène et de les remplacer en peu d'actions. Cette technique semble couvrir l'ensemble des besoins identifiés mais repose sur l'utilisation de boîtes de dialogue ce qui augmente l'indirection temporelle et ne permet pas des manipulations rapides à effets immédiats, composante nécessaire à l'*exploration*. Les « surrogates » (Kwon et al., 2011) sont des substituts d'interactions, des instruments qui contrôlent d'autres objets. Ils pallient certaines limites de la manipulation directe (l'accès aux objets, les modifications de plusieurs objets et la modification de propriétés intangibles) en introduisant une indirection spatiale : l'utilisateur n'interagit pas directement avec le ou les objets d'intérêt mais avec le surrogate qui répercute les modifications sur ces objets.

Nous avons également étudié des techniques qui peuvent aider les utilisateurs dans leur activité de conception exploratoire. Une fonctionnalité utilisée couramment pour comparer des solutions est celle de « l'annuler/refaire », bien qu'il s'agisse d'une utilisation détournée de cette fonctionnalité. Les historiques non-linéaires (Scull et al., 2009)(Appert et al., 2012) ou graphiques (Kurlander & Feiner, 1988) encouragent *l'exploration* puisqu'il est possible d'annuler sélectivement les actions précédentes au lieu de suivre l'ordre de leur réalisation. Des outils tels que « Side-Views » (Terry & Mynatt, 2002c) ou « Parallel Pies » (Terry et al., 2004) sont des techniques spécialement conçues pour améliorer la *conception exploratoire* : elles offrent des services permettant de comparer des prévisualisations de résultats de commandes et d'explorer des variations.

D. Évaluation des techniques d'interaction

Afin d'évaluer le pouvoir d'action, il est utile de disposer de méthodes et d'outils existants d'évaluation de l'interaction (tel que GOMS et le Keystroke-Level Model (Card et al., 1980)(John & Kieras, 1996), les modèles de tâches (Diaper & Stanton, 2003) etc.).

Ces outils permettent d'aider les concepteurs à choisir la ou les techniques d'interactions les plus efficaces selon la tâche à réaliser. Ces méthodes reposent sur des dimensions d'analyse dont certaines prennent en compte les besoins identifiés dans les scénarios de travail.

1. Contexte cognitif

De nouvelles techniques d'interaction sont développées, issues de travaux de recherche, augmentant ainsi la métaphore du bureau et les interfaces graphiques de types WIMP. Elles restent cependant peu connues et peu utilisées dans les applications commerciales, car les boîtes à outils classiques ne les proposent pas (Mackay, 2002). Mackay et al. souhaiteraient encourager les concepteurs à utiliser des techniques post-WIMP en les comparant.

Les tâches à réaliser étant affectées par le contexte courant de travail, (Mackay, 2002) s'appuie sur deux des six activités définies par (T. R. G. Green & Petre, 1996) afin de proposer des catégories de tâches : celles dites de « copie » et celles de « modification » ou de résolution de problème. La copie est une tâche mécanique, ne demandant pas de savoir requis, l'utilisateur sait ce qu'il doit obtenir (cela correspond à de l'incrémentation ou bien de la transcription). La modification demande des capacités à résoudre un problème et d'être capable de changer de point de vue à tout moment, impliquant de savoir prendre des décisions aux cours de la progression dans la tâche (*conception exploratoire*). Si l'utilisateur peut utiliser exactement les mêmes commandes dans les deux types de tâches et obtenir un résultat identique, c'est en fait le processus de réflexion qui diffère ainsi que la stratégie optimale d'exécution. Cela induit l'impossibilité de déterminer l'efficacité absolue d'une technique d'interaction donnée pour une commande : il faut plutôt examiner l'ensemble des séquences de commandes réalisées par l'utilisateur sous différentes conditions cognitives pour déterminer la technique la plus effective.

2. Stratégies pour la conception

Les résultats de ces travaux montrent qu'une technique d'interaction n'est pas meilleure qu'une autre dans l'absolu. Elle ne peut l'être que selon la nature de la tâche à accomplir : copie, ou résolution de problème. Plus précisément, la technique d'interaction optimale dépend de la tâche, du contexte cognitif, mais aussi des préférences individuelles.

(Mackay, 2002) présente trois stratégies afin d'aider au choix d'une technique d'interaction, applicable à la conception de tout type d'éditeur graphique. La plus

commune consiste à choisir la technique arbitrairement en premier lieu et de favoriser certains types de tâches par rapport à d'autres. Un argument en faveur de cette stratégie serait la minimisation du nombre de commandes disponibles à apprendre. Fournir plusieurs moyens d'accomplir une même action ne serait pas forcément idéal, les performances pouvant être diminuées si l'utilisateur a une variété de chemins pour atteindre son but. Cette stratégie pourrait expliquer pourquoi certains logiciels conçus pour supporter des tâches de résolution de problèmes sont finalement mieux adaptés à des tâches mécaniques de copie. La deuxième stratégie suggère de créer des agents intelligents, pouvant détecter les intentions de l'utilisateur afin de lui fournir la technique d'interaction la plus appropriée. Le problème, c'est qu'il est difficile de voir comment le système pourrait détecter correctement les différences entre les deux contextes cognitifs puisque les ensembles mesurables d'actions peuvent être identiques. Par ailleurs cela peut freiner l'utilisateur qui pourrait être contraint d'avoir à corriger les erreurs du système. La dernière stratégie préconise d'intégrer toutes les techniques d'interaction dans une même interface. Cela pose des difficultés de conception mais offre une flexibilité certaine, l'utilisateur pouvant décider lui-même de la technique à utiliser selon le contexte.

3. Coût des techniques d'interaction

CIS est un modèle qui aide à décrire une technique d'interaction, à l'analyser, et à prédire son efficacité selon le contexte d'utilisation (Appert et al., 2005). CIS décrit une interface comme un ensemble d'objets que l'utilisateur peut manipuler : les « work objects » (par exemples des formes représentées à l'écran) et les « tool-objects » (items de menu, barre d'outils…), et définit son état par l'ensemble des objets et la valeur de leur attributs. Une manipulation est une création, une modification ou une suppression d'un objet de travail décrit par un couple commandes/attributs. Au final, une technique d'interaction est un ensemble d'étapes d'interaction, c'est-à-dire l'ensemble des séquences d'actions qui réduisent progressivement l'espace d'interaction (c'est-à-dire les manipulations disponibles pour l'utilisateur dans un état donné) à une simple manipulation qui une fois exécutée, mène à un nouvel état et un nouvel espace d'interaction.

CIS définit quatre propriétés des techniques d'interaction : l'ordre et le parallélisme, la persistance, la fusion et le développement. La fusion caractérise la capacité d'une technique d'interaction de modifier plusieurs objets de travail en définissant des manipulations multiples en une seule fois (*pouvoir d'action*), et le développement

correspond à la capacité offerte à l'utilisateur de créer des copies d'outils avec des valeurs d'attributs différentes.

E. Activité de programmation

Les problèmes soulevés concernent également les activités de programmation. Même si l'activité de programmation en elle-même utilise souvent des moyens d'interaction limités (édition et modification de texte), certaines constructions des langages eux-mêmes proposent des moyens pour offrir du pouvoir d'action : par exemple changer le code d'une méthode à une influence sur le comportement de tous les objets concernés. Green et al. (1998) qualifient ce pouvoir d'action des langages comme une « mise à jour de masse ».

L'héritage des langages à objets et la délégation des langages à prototypes sont des concepts de structurations permettant d'abstraire des propriétés communes et de réduire le nombre d'actions requis. Une hiérarchie de classe ou un prototype est donc un moyen d'augmenter le pouvoir d'action : modifier une classe ou un prototype (par exemple une méthode) change le comportement de toutes ses instances ou clones.

De même, les outils de refactorisation que l'on trouve dans les environnements de développement (IDE) sont un exemple de réponse au besoin d'un pouvoir d'action augmenté : si l'utilisateur change le nom d'une méthode, le système applique ce changement à tous les appels de cette méthode, possiblement dans plusieurs classes ou fichiers différents.

1. Principes d'abstraction dans la programmation

Le besoin de structurer et de créer des abstractions est fortement présent dans les langages de programmation orientés objets (POO), avec le concept d'héritage par exemple. L'héritage est un concept ancré dans les langages à objets, augmentant la modélisation conceptuelle et la réutilisabilité du code. Selon le point de vue choisi, il peut s'agir d'un outil permettant de structurer et de raisonner sur le programme, ou bien d'un outil favorisant le partage de code et sa réutilisation (Taivalsaari, 1996).

Taivalsaari recense quatre principes majeurs d'abstractions relatifs à la structuration dans la POO : la classification/instanciation, la généralisation/spécialisation, l'agrégation/décomposition et le groupement/individualisation (Figure 38). Ces principes démontrent l'intérêt d'établir des structures pour faciliter la compréhensibilité du code et sa réutilisabilité, mais l'analogie avec les techniques de structurations dans l'édition graphique peut nous aider à définir une taxonomie des structures.

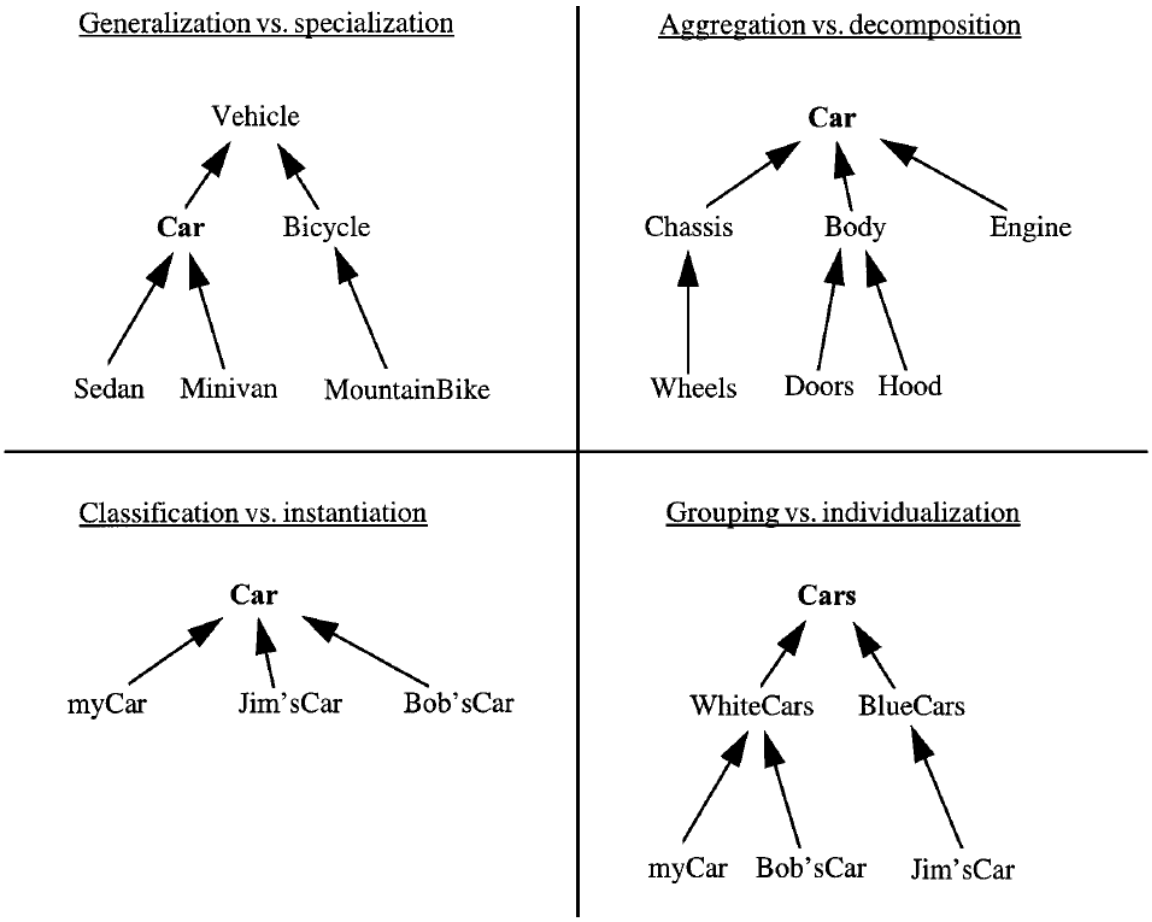


Figure 38. Les quatre principes d'abstraction (Taivalsaari, 1996, fig. 2).

Le principe de classification consiste en la création de classes ou de catégories, une instance de classe partageant des caractéristiques communes avec les autres tout en restant individualisée. Si la plupart des langages orientés objets permettent de créer des classes, ce n'est pas le cas pour les langages à prototypes qui sont dit « sans classes », mais où l'instanciation reste possible via le clonage d'objets. Le concept de classe se rapproche du concept de master dans l'édition graphique : un objet abstrait que l'on instancie pour obtenir une conception graphique structurée. La classification supprime les détails des instances et met en évidence les propriétés des classes.

L'agrégation revient à considérer un ensemble de concepts comme un seul concept de plus haut niveau : un agrégat. Cela correspond aux abstractions telles que l'on peut établir dans une tâche de conception graphique. Par exemple, dans le scénario de travail du clavier virtuel, le groupe définissant une touche du clavier est une agrégation d'une lettre, et de deux rectangles. L'agrégation met en retrait les propriétés de ses éléments constituant pour dévoiler les détails des relations comme un ensemble.

La généralisation est la construction de concepts qui partagent un ensemble de similarités avec des concepts sous-jacents. La généralisation est à la base même du mécanisme d'héritage : sur une base d'une ou plusieurs classes données, la généralisation fournit une classe plus générale, plus abstraite, qui détient les caractéristiques communes et supprime celles qui sont différentes pour chacune d'entre elles. Cela supprime donc les différences entre des éléments et met en valeur leurs propriétés communes.

Le groupement se traduit par la nécessité de grouper des éléments ensemble non pas parce qu'ils ont les mêmes propriétés (classification) mais parce qu'il est important de décrire les propriétés d'un groupe d'objet comme un ensemble. Cela permet de créer des collections non homogènes et supprime les détails des groupes d'objets.

2. La délégation des langages à prototypes

Les langages à prototypes offrent une alternative aux langages basés sur des classes dans la POO (Lieberman, 1986)(Myers, Giuse, & Zanden, 1992). Ils offrent un modèle de création flexible qui permet le partage de propriétés et de comportement. De tels mécanismes permettent aux utilisateurs de structurer une hiérarchie de prototypes et d'agir sur un ou plusieurs clones en manipulant un prototype dans la hiérarchie de délégation avec plus de flexibilité qu'avec un modèle de hiérarchies de classes.

Dans la POO classique, l'héritage catégorise les objets en classes qui décrivent le comportement de leurs instances. Le concept de classe devient obsolète dès lors que l'on raisonne en prototypes. Un prototype représente le comportement par défaut d'un concept. D'autres nouveaux objets peuvent alors réutiliser tout ou partie des caractéristiques du premier prototype et se démarquer en spécifiant quelles caractéristiques diffèrent entre eux. La délégation est le mécanisme permettant d'implémenter le concept de prototypes dans la POO (Lieberman, 1986).

L'inconvénient des classes est qu'elles doivent toutes être crées obligatoirement avant les instances que l'on souhaite utiliser et le comportement de ces instances ne peut être associé qu'avec des classes. Parce que n'importe quel objet peut être utilisé en tant que prototype et que l'on peut déléguer et partager des propriétés entre les objets à tout moment, la délégation s'avère donc bien plus flexible et permissive. Cela laisserait à penser que les techniques de structurations explicites par le biais d'objets abstraits telle que les classes (par exemple les masters) seraient donc moins efficaces pour la création et la manipulation d'objets graphiques que des techniques basées sur la délégation.

Pour mieux comprendre le concept de délégation et la différence avec la classification, Lieberman présente l'exemple de Clyde, un éléphant. En raisonnant en « classes », l'on peut se dire que pour que l'existence d'un éléphant Clyde ainsi que celle d'un éléphant Fred soit possible, il faut que l'on accède à une classe, probablement « éléphant », qui détienne l'ensemble des attributs minimalistes que doivent avoir tous les éléphants. Clyde et Fred sont alors chacun une instance de la classe éléphant. Pour raisonner en « prototypes », l'on peut considérer que l'on a un éléphant : Clyde. Il existe, c'est un prototype. Si l'on veut pouvoir définir une autre entité (l'éléphant Fred) ayant des caractéristiques communes avec Clyde, il suffit de demander à Clyde de déléguer tout ou partie de ses propriétés à Fred, le nouveau prototype, qui peut se contenter d'exprimer quelles sont les différences qui le démarquent de Clyde. La Figure 39 illustre cet exemple.

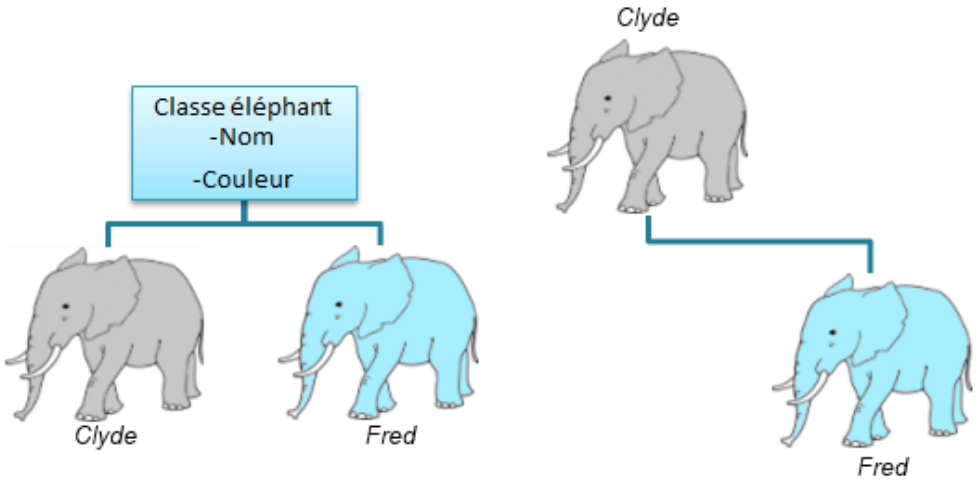


Figure 39. A gauche, Clyde et Fred sont deux instances de la classe « éléphant », et ont des valeurs de propriétés qui leur sont propres. A droite, Clyde est un prototype d'éléphant qui délègue ses propriétés à Fred. Fred est alors comme Clyde, à la différence près qu'il est bleu.

La délégation supprime la distinction entre classes et instances, tous les objets peuvent servir de prototypes. La délégation est alors très avantageuse pour le développement interactif et incrémental, puisqu'il n'est pas nécessaire de réfléchir à une structure au préalable. Cela encourage l'exploration et diminue la barrière d'abstraction ainsi que l'engagement prématuré.

Les avantages des langages à prototypes face aux langages à base de classes ont encouragé des chercheurs à proposer des systèmes et environnements permettant la création et la manipulation de prototypes. Self est un langage de programmation orienté-objet basé sur les prototypes (Ungar & Smith, 1987).

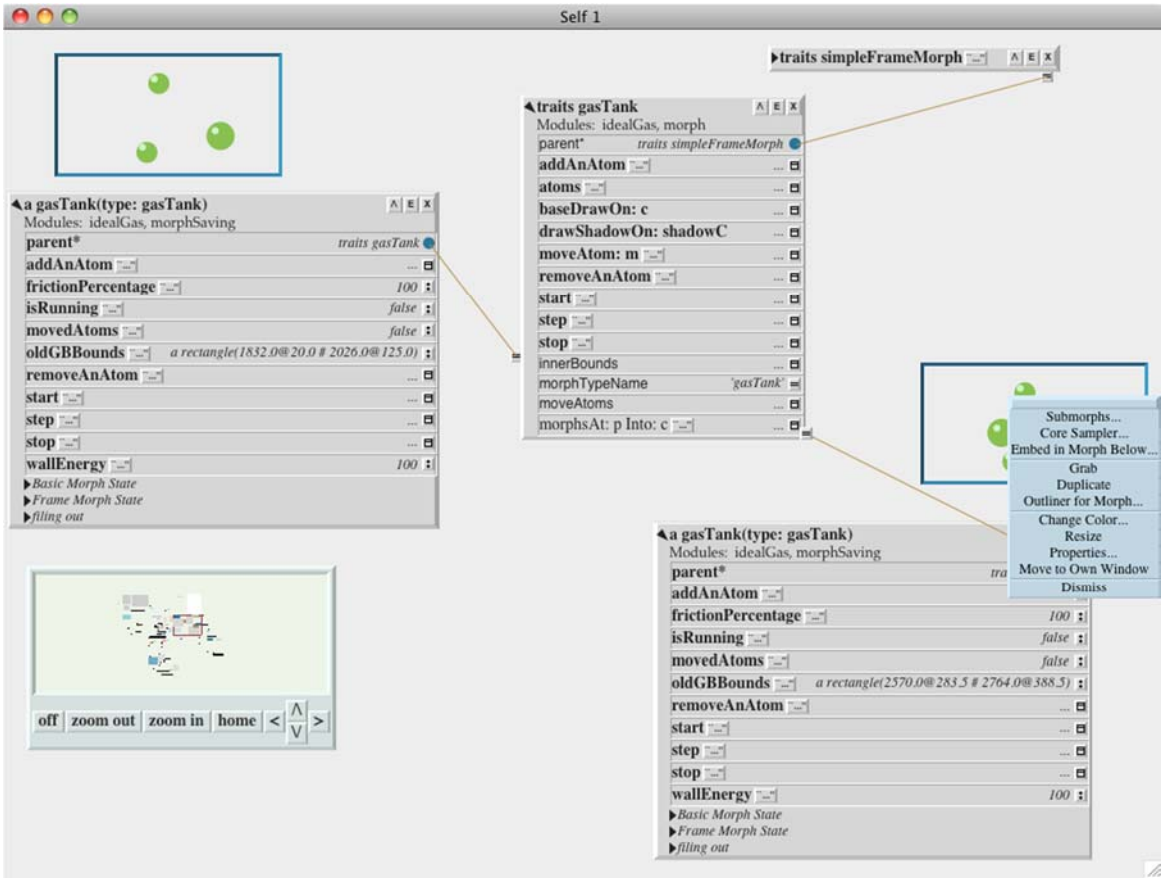


Figure 40. Présentation de Self 4.4[5] et de son interface utilisateur, Morphic. Les objets graphiques sont des morphes qui dépendent de prototypes dont les liens de parenté sont représentés par les traits oranges.

Morphic (Figure 40) est un kit de construction d'interfaces utilisateurs basé sur le langage Self qui réifie des prototypes et des clones en objets graphiques et permet leur construction et leur édition via manipulation directe (Maloney & Smith, 1995). L'un des buts avoués de Morphic étant de faciliter la construction ainsi que l'édition d'objets graphiques interactifs, ils constituent alors l'abstraction centrale de Morphic, et sont appelés « morphes » (du grec pour forme). Les morphes peuvent être composés d'autres morphes, des « submorphs », décrivant ainsi une agrégation de morphes analogue à une hiérarchie de prototypes et détenant les avantages de ces dernières : une grande flexibilité et un pouvoir d'action augmenté pour la manipulation d'éléments graphiques.

(Myers et al., 1992) présentent le système Garnet, un environnement offrant aux programmeurs la capacité d'écrire du code avec un langage à prototypes. Garnet repose sur KR (Knowledge Representation), un « moteur » logiciel modélisant des

[5] Self 4.4 est la dernière version du langage. Documentation et téléchargement disponibles sur http://selflanguage.org/.

connaissances liées par des contraintes. Dans Garnet, les objets sont persistants et reçoivent les entrées des utilisateurs, ce qui en fait un système efficace pour le développement d'interfaces graphiques. Sa force vient donc de celle des langages à prototypes : le partage de propriétés entre un prototype et ses instances, la rapidité de mise en œuvre, et les modifications dynamiques. Le style de programmation dans Garnet se veut donc différent des autres systèmes des langages à objets. Le programmeur crée ses objets graphiques avec des outils interactifs, écrit les contraintes pour définir des relations et ensuite associe les instances à des interacteurs prédéfinis.

3. La restructuration automatique

Que l'on définisse une hiérarchie de classe ou une arborescence de prototypes, le programmeur doit réfléchir à sa structure en tout premier lieu (très minutieusement pour une hiérarchie de classes et sans doute un peu moins pour une hiérarchie de prototypes qui se veut plus flexible). La volonté derrière un mécanisme de restructuration automatique est celle de permettre la création de structures a posteriori. Concrètement, cela permet à l'utilisateur d'explorer sa solution sans être contraint de définir des structures au préalable, puisqu'elles pourraient être générées automatiquement : la barrière d'abstraction est donc minimale. Dans la programmation orientée-objet, la création de structures est un concept très présent par la notion d'héritage, et le développeur doit penser très tôt dans le processus de développement à la hiérarchie qu'il doit construire car il sera très fastidieux de réorganiser par la suite cette hiérarchie. Et si la structure préalablement établie n'est pas parfaite le code devient plus difficile à comprendre ainsi qu'à maintenir. Restructurer un programme peut donc s'avérer très difficile, d'autant plus si le système est complexe et important où s'il a été conçu par plusieurs programmeurs (Moore, 1996).

Des outils ont été conçus afin d'aider à la conception de structures hiérarchiques de prototypes. Par exemple, Guru est un algorithme qui crée automatiquement un graphe organisé de prototypes, en factorisant des propriétés partagées par tous les prototypes dans des nouveaux, construisant ainsi une nouvelle hiérarchie optimisée, c'est-à-dire sans doublon de propriété dans des prototypes différents (Moore, 1995) (Figure 41 et Figure 42).

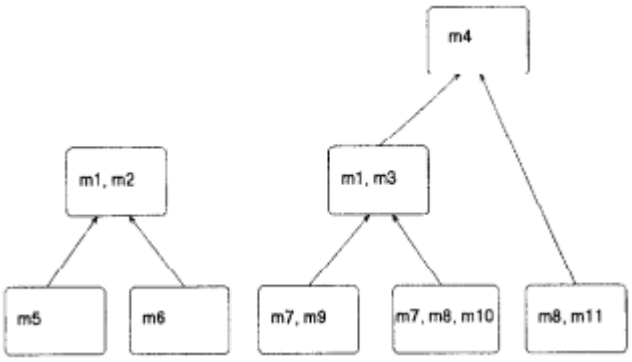


Figure 41. Exemple de hiérarchie d'héritage avant restructuration en haut (Moore, 1996, fig. 1).

Figure 42. Hiérarchie d'héritage après restructuration. (Moore, 1996, fig. 3).

4. Programmation par l'exemple

L'outil informatique est certes adapté à la réalisation de tâches répétitives (par l'utilisation de programmes, scripts, macros…), pour autant, les utilisateurs se voient trop souvent contraints de les réaliser eux-mêmes (Kurlander, 1993). Une solution serait peut-être de permettre à l'utilisateur de créer ses propres commandes. Ce concept est celui de « l'end-user programming » (Cypher & Halbert, 1993) : il s'agit de rendre possible pour les utilisateurs finaux, la construction ou la modification des outils mis à dispositions afin de pouvoir agir plus efficacement sur les objets d'intérêts.

Les macros sont un exemple d'abstractions que peuvent créer les utilisateurs qui permettent d'automatiser des tâches répétitives. Les macros sont des outils qui encapsulent plusieurs opérations et qui permettent ainsi à l'utilisateur de les réaliser en une seule action. Il peut donc s'agir d'outils efficaces pour réduire la viscosité d'une tâche ou pour réaliser des tâches non prévues par l'application à l'origine. Par ailleurs, ces outils peuvent profiter à toute une communauté d'utilisateurs dès lors qu'ils peuvent être partagés (Kurlander & Feiner, 1992). Cependant, comme toute abstraction, s'il est à la charge de l'utilisateur de les définir cela peut s'avérer coûteux en temps d'exécution et d'apprentissage. Qui plus est, les utilisateurs non experts ont

moins tendance à utiliser ces outils puisque cela demande une certaine familiarité avec le langage d'extension de l'application (Kurlander & Feiner, 1992).

La Programmation par l'Exemple (que nous simplifierons par PbE pour « Programming by Example ») est une approche qui se veut plus proche de la façon de penser d'un utilisateur dès lors qu'il s'agit de programmer une série d'instructions (Lieberman, 2001). Plutôt que de devoir écrire du code, l'utilisateur procède en réalisant un exemple d'une tâche à répéter que le système va apprendre, lui permettant de réappliquer sur d'autres objets une même séquence d'opérations. L'idée des chercheurs repose sur le fait que lorsque l'on désire montrer à quelqu'un comment réaliser une tâche, on l'enseigne, en montrant comment le faire étape par étape. Cette approche est à l'opposé de la programmation d'outils informatiques telle que nous la connaissons, où il est nécessaire d'écrire au préalable quelles actions l'outil va réaliser, dans quel ordre et en devant respecter une syntaxe précise. La PbE serait donc une approche plus « naturelle et efficace » de programmation pour un utilisateur (Lieberman, 2001). La PbE pourrait donc être un moyen de créer des abstractions (les programmes issus des exemples réalisés par l'utilisateur) en limitant l'effort requis, améliorant ainsi la conception exploratoire.

a) Problème de l'interprétation du système

Un système de PbE enregistre les actions réalisées par l'utilisateur dans l'interface pour produire un programme généralisé que l'utilisateur pourra utiliser plus tard dans des exemples analogues. Le problème majeur de la PbE reste celui de l'ambiguïté : le système ne peut pas toujours savoir pour quelle raison un utilisateur a réalisé une action particulière (Myers & Buxton, 1986). Par exemple, si l'utilisateur sélectionne un objet, le système doit-il comprendre que l'utilisateur fait référence à cet objet en particulier ou bien à un objet ayant un nom similaire ? A moins qu'il ne s'agisse d'un objet ayant une position particulière à l'écran ou d'un objet du même type ou alors un objet ayant d'autres propriétés ? Par ailleurs, la plupart des systèmes de PbE décrivent ces objets selon les propriétés des données de l'application que l'objet visuel représente et non par les propriétés visuelles elles-mêmes (St. Amant, Lieberman, Potter, & Zettlemoyer, 2000). Par exemple, si un utilisateur sélectionne un champ de texte dans une application agenda, le système va probablement représenter cette action comme la sélection d'un évènement particulier et non comme la sélection d'un champ de texte. Les propriétés visuelles étant peu considérées par ces systèmes cela limite les possibilités de l'utilisateur. Et si des outils permettent de réaliser des « macros par l'exemple » facilement, il y a un manque de représentation visuelle de ces macros,

ce qui rend impossible de voir les opérations qui les composent (Kurlander & Feiner, 1992).

b) Techniques de programmation par l'exemple

Les outils tels que les tableurs sont de très bons exemples d'outils où « l'end-user programming » est rendu possible via la construction de macros et l'application de la PbE. Par exemple, dans Excel, si l'utilisateur souhaite écrire une liste de valeurs qui s'incrémentent d'un même pas, il n'est pas obligé de calculer ces valeurs une par une et les entrer à la main, ce qui serait fastidieux, ou bien d'écrire du code qui incrémente des valeurs dans une boucle. Il lui suffit d'entrer deux ou trois de ces valeurs, en guise « d'exemple », puis de « demander » au système qu'il remplisse les champs de valeurs suivants à l'aide d'une manipulation directe (Figure 43).

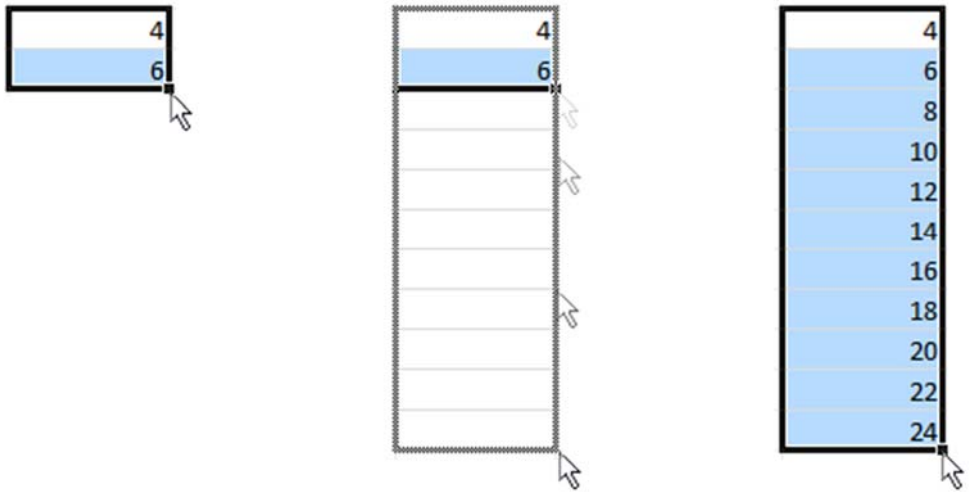


Figure 43. Le système comprend qu'il doit incrémenter une suite arithmétique de raison égale à la différence des valeurs données par l'utilisateur.

Des macros visuellement représentées

Afin de pallier le problème de l'absence de représentation visuelle des macros, (Kurlander & Feiner, 1992) proposent une technique de création de macro par l'exemple. A l'aide d'historiques graphiques qui fournissent des enregistrements visuels de commandes exécutées (Kurlander & Feiner, 1988), l'utilisateur peut construire des macros par l'exemple, en sélectionnant dans l'historique graphique une succession d'opérations réalisées sur des éléments afin de pouvoir les réutiliser sur d'autres (Figure 44 et Figure 45).

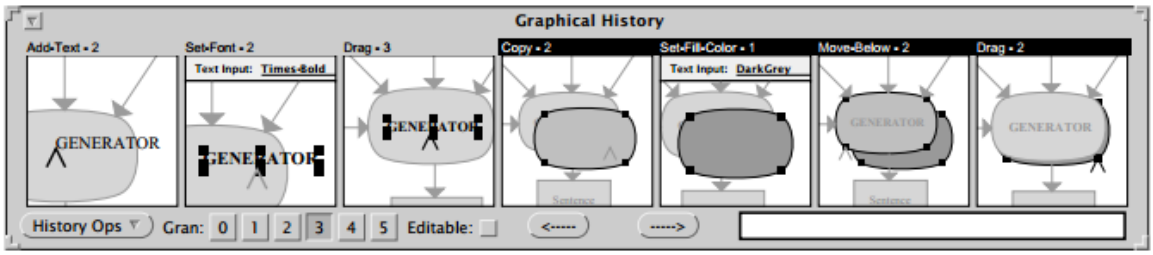


Figure 44. Un historique graphique contient les étapes qui ont permis d'ajouter une ombre portée à un objet. L'utilisateur a sélectionné les quatre panels dont le titre est en blanc sur fond noir dans le but d'en extraire une macro (Kurlander & Feiner, 1992, fig. 1).

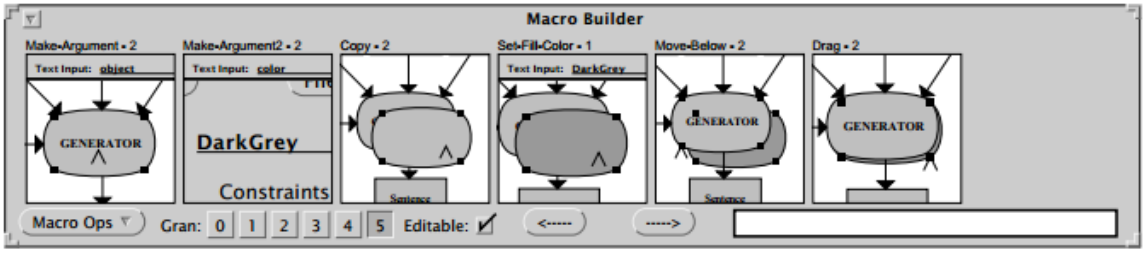


Figure 45. Fenêtre de construction de macros qui contient les quatre opérations sélectionnées sur la figure précédente. L'utilisateur peut ainsi créer une macro permettant d'ajouter des ombres portées à des objets (Kurlander & Feiner, 1992, fig. 3).

Peridot

Les constructeurs d'interface permettent de les concevoir par manipulation directe plutôt qu'en écrivant leur code à la main. Cependant, ils ne permettent généralement de créer que des aperçus statiques puisque la réalisation de l'interaction reste à la charge du développeur. (Myers & Buxton, 1986) présentent Peridot, un système qui permet de concevoir des interfaces utilisateurs et de générer le code décrivant son comportement, par démonstration. L'intérêt est qu'il n'y ait pas de programmation requise pour construire une interface fonctionnelle, une grande part pouvant être réalisé par manipulation directe : le concepteur montre au système comment l'interface doit fonctionner (Myers, 1992). A chaque action de l'utilisateur, le système essaie d'inférer les relations entre l'action et les éléments de l'interface pour proposer une séquence d'instructions généralisée. Un tel système peut permettre de créer des prototypes extrêmement rapidement, sans avoir à écrire de code, et peut rendre accessible à des non programmeurs la réalisation d'interfaces interactives.

Généralisation visuelle

La généralisation visuelle est un concept de la PbE où l'on s'intéresse plus aux propriétés visuelles d'un objet qu'à son sens (St. Amant et al., 2000). En règle générale, le système effectue une correspondance entre un objet sélectionné et le modèle de données de l'application pour permettre une généralisation sur les propriétés de la donnée. Mais parfois, la description intuitive d'un utilisateur peut dépendre des

propriétés visuelles des éléments de l'écran eux-mêmes sans regard sur leur représentation dans l'application. L'idée est donc d'utiliser les propriétés visuelles pour permettre aux systèmes de PbE de faire de la généralisation visuelle. L'intérêt réside dans le fait que les systèmes orientés-visuels correspondent plus à ce que l'utilisateur peut réellement voir et seraient donc plus transparents, surtout pour les non-experts.

5. Synthèse sur la programmation

Le besoin de structurer identifié dans le contexte de l'interaction graphique est également présent dans le domaine de la programmation. L'héritage ou la délégation sont des exemples de méthodes de structuration qui peuvent offrir un plus grand pouvoir d'action sur l'ensemble des éléments de la structure : agir sur des classes afin de modifier ses instances, ou agir sur des propriétés déléguées afin de propager ces modifications. Le langage Self et son interface Morphic, démontrent que les frontières entre interaction et programmation peuvent se rejoindre, en permettant de « programmer » de façon interactive des comportements d'objets graphiques et de structurer leurs propriétés selon le principe de la délégation des langages à prototypes.

Nous pouvons relever néanmoins que la création de ces structures à un stage précoce de développement peut s'avérer problématique, dans le sens où de telles structures peuvent être amenées à être modifiées plus tard. Obliger le développeur à réfléchir en amont à sa structure n'encourage pas l'exploration. C'est pourquoi, des méthodes de restructuration automatique telles que proposées par l'outil Guru peuvent répondre au besoin de la conception exploratoire.

La programmation par l'exemple est une approche qui souhaite concilier la puissance d'action de la programmation et les interactions physiques. Le système infère les intentions de l'utilisateur afin de proposer des instructions à exécuter. Ces instructions ne sont pas issues de l'écriture de code mais d'interactions qui servent d'exemple.

F. Synthèse de l'état de l'art

1. Réduire la viscosité, utiliser des abstractions

Les travaux de Kurlander (1993) et les observations issues des enquêtes contextuelles montrent que les utilisateurs sont souvent contraints de répéter les mêmes actions lorsqu'ils souhaitent modifier plusieurs objets. Ce fait corrobore notre problématique. Ce phénomène est par ailleurs caractérisé par l'une des dimensions cognitive : la viscosité. La viscosité est un véritable frein à la conception exploratoire : dans une situation où le résultat d'une conception graphique n'est pas connu à l'avance, le

besoin d'essayer et d'explorer des solutions (Terry & Mynatt, 2002a) n'est pas assuré à partir du moment où apporter des modifications engendre une importante quantité d'actions à reproduire. Les dimensions cognitives apportent des éléments de réponse permettant de réduire la viscosité : une solution consiste à introduire des abstractions, une autre dimension cognitive.

L'utilisation d'abstractions offre une portée d'action supérieure à l'objet individuel. Elles permettent de définir des groupements d'éléments que l'on peut considérer comme une entité individuelle afin de réduire la viscosité, mais servent aussi à conceptualiser une scène (T. R. G. Green, 1989). Les abstractions restent cependant bloquantes pour la conception exploratoire : leur construction peut être coûteuse en temps et en effort (parfois plus que la répétition d'une courte séquence d'actions) et doivent être maintenues. La barrière d'abstraction définit le nombre d'abstractions à réaliser avant de pouvoir utiliser efficacement un système : nous estimons qu'elle doit être minimale.

2. Techniques de structuration

Les techniques d'interaction que nous avons étudiées (notamment les groupes, les styles et les masters) et la notion d'abstraction ont un point commun : il s'agit de concepts reposant sur des *structures*. Une structure décrit la « manière dont les parties d'un tout sont arrangées entre elles » [6]. Ainsi, un groupe est une structure dans le sens où il résulte du réarrangement de plusieurs éléments entre eux. Les styles servent par essence à établir la structure graphique d'un document, ils définissent une organisation cohérente d'éléments graphiques. C'est également le cas des masters puisqu'ils servent à construire des relations de dépendances, sous la forme d'arrangements hiérarchiques (un master étant « au-dessus » des objets dépendants).

Ces techniques de structuration présentent cependant certains désavantages. Si l'on peut effectivement sélectionner des groupes d'objets pour les modifier comme s'il ne s'agissait que d'un seul élément, et si la création de masters et leur instanciation permettent de modifier toutes les instances en une seule fois, cela demande une structuration spéciale et préalable de la scène qui facilitera les changements répétitifs. Si l'utilisateur ne peut prédire, lorsqu'il crée sa solution, quels seront ces changements répétitifs qui surviendront plus tard, ou s'il est trop contraignant d'établir proprement la structure, ces techniques ne seront pas d'une grande aide (Kurlander, 1993). On se retrouve alors face à une barrière d'abstraction forte ou un besoin d'anticipation élevé de la part de l'utilisateur (T. R. G. Green, 1989).

[6] Définition issue du dictionnaire Larousse.

Par ailleurs, si l'interaction sur le contenu d'une structure est plutôt bien éprouvée (la modification synchrone fonctionne pour les composants individuels de la structure), l'interaction sur la structure elle-même n'est pas toujours utilisable (difficulté pour interagir avec la structure, exemple des groupes hétérogènes ne pouvant pas appliquer certaines opérations), révélant un manque de cohérence entre l'interaction sur une structure et celle sur ses constituants.

3. Apport de la programmation pour l'interaction avec les structures

Nous avons vu que la manipulation directe est efficace pour interagir avec des objets individuels : elle réduit l'effort requis pour interagir avec les objets d'intérêt en permettant des actions physiques rapides, incrémentales et réversibles (Shneiderman, 1987). Cependant, les limites de ce paradigme présentées dans les travaux de (Frohlich, 1993) et (Kwon et al., 2011) révèlent que la manipulation directe ne couvre pas l'ensemble des besoins que nous avons identifiés : notamment l'effort de recherche et de désignation d'objets non-visibles, trop petits, superposés, ou trop nombreux, ainsi que les modifications d'objets multiples de façon synchrone. L'exemple donné par Hutchins (1989) dans lequel il est demandé de trouver tous les mots commençant par la lettre 's' afin de leur appliquer une modification illustre bien ces lacunes : la manipulation directe ne permet pas de désigner tous ces mots en peu d'actions, il faut les sélectionner et les modifier un par un, au risque d'en oublier.

La notion de pouvoir d'action est présente dans le domaine de la programmation par le concept de mise à jour de masse (Thomas R. G. Green et al., 1998). (Taivalsaari, 1996) présente différents principes d'abstraction qui permettent d'étendre la portée des modifications de code tels que le modèle de classes et d'instances, ou le modèle de délégation. Cependant, les moyens d'interaction avec un programme sont limités, puisqu'il s'agit essentiellement de saisie de texte. Ces interactions ont une forte indirection spatiale et temporelle : modifier une ligne de code peut avoir des répercussions à d'autres niveaux du programme (voire dans d'autres fichiers), et les effets ne sont visibles qu'après compilation et exécution du programme.

Self (Ungar & Smith, 1987) (et son interface Morphic (Maloney & Smith, 1995)) est un exemple d'environnement où les frontières entre programmation et interaction se rejoignent : la puissance de la délégation du langage Self étant octroyée aux développeurs qui peuvent « écrire du code » non pas en saisissant du texte, mais en utilisant des moyens d'interaction tel que la manipulation directe, pour établir des relations de dépendances entre prototypes par exemple. Les techniques

de programmation par l'exemple illustrent également la volonté d'augmenter le pouvoir d'action des utilisateurs d'outils interactifs au niveau de la programmation. Elles permettent de créer des instructions en réalisant des interactions servant d'exemples plutôt qu'en écrivant du code.

Le langage Self démontre que ce qui peut être fait en langage à base de classes peut aussi bien être fait dans un langage sans classes. (Ungar, Chambers, Chang, & Hölzle, 1991) pensent que les classes ne sont pas nécessaires pour structurer un programme puisque les objets peuvent très bien le faire d'eux-mêmes : le partage de comportement est facilité entre les instances, et le raffinement ainsi que l'encapsulation sont tout à fait possibles sans avoir à décrire explicitement des types ou des classes. Ce point est intéressant pour nos travaux : il nous permet d'envisager des solutions où les objets que l'on donne à manipuler ne sont pas nécessairement de statuts différents. Il serait donc possible de définir des structures qui peuvent s'affranchir d'objets abstraits ou de meta-objets comme les surrogates.

Les solutions à la problématique de recherche que nous proposons reposent à la fois sur des principes d'interaction et de programmation. Nous souhaitons bénéficier du pouvoir d'action des langages de programmation grâce à des interactions entièrement instrumentées, pour construire, modifier et interagir avec des structures qui permettent d'étendre ces interactions à des objets multiples.

PRINCIPES DE STRUCTURATION

L'intérêt principal de la structuration dans l'édition graphique est d'être un remède à la viscosité : la structuration peut permettre aux utilisateurs d'agir sur un grand nombre d'éléments en peu d'actions. Dans ce chapitre, nous définissons différents principes de structuration illustrés par des outils interactifs que nous avons conçus afin de répondre à la problématique de l'édition synchrone d'objets (Hoarau & Conversy, 2011). Nous opposons la structuration explicite (l'utilisateur construit lui-même ses structures) à la structuration implicite (la structure est révélée d'elle-même par les actions de l'utilisateur) (Hoarau & Conversy, 2012), et présentons également les principes de structuration automatique (il s'agit de structures explicites créées automatique par le système). Pour pouvoir explorer ces concepts, nous avons conçu un éditeur graphique incluant divers outils de structuration à partir des besoins identifiés lors des enquêtes contextuelles et de séances de conception participatives.

A. Exigences de conception

A partir du résultat des enquêtes contextuelles et de l'analyse de l'état de l'art, nous avons synthétisé un ensemble d'exigences concernant la manipulation d'objet via des techniques de structuration (Tableau 2). Celles-ci sont notamment corrélées à l'ensemble des tâches identifiées par Frohlich (1993) qui sont reconnues comme étant difficiles à réaliser avec des techniques de manipulation directe. Les exigences sont définies selon trois ensembles d'exigences : gérer les ensembles d'objets (R1), gérer les actions (R2) et favoriser la conception exploratoire (R3).

Tableau 2. Exigences

Gérer les ensembles d'objets (R1)	Rechercher (R1.1)
	Désigner (R1.2)
	Modifier (R1.3)
	Identifier les ensembles (R1.4)
Gérer les actions (R2)	Spécifier leur nature (R2.1)
	Spécifier leurs paramètres (R2.2)
	Spécifier leur portée (R2.3)
	Percevoir leurs conséquences (R2.4)
Favoriser la conception exploratoire (R3)	Essayer (R3.1)
	Évaluer (R3.2)
	Explorer à court-terme (R3.3)
	Comparer les versions (R3.4)
	Structurer a posteriori (R3.5)

1. Gérer les ensembles

Gérer les ensembles comprend la nécessité de pouvoir *rechercher* (R1.1) et *désigner* (R1.2) des objets appartenant à des ensembles. L'exigence (R1.1) *rechercher* fait écho au besoin de *recherche* identifié dans les scénarios de travail (lorsqu'Élodie devait par exemple identifier les touches de clavier double afin de modifier la taille des caractères ou bien lorsque Sébastien devait repérer les diapositive contenant le plan de son cours). Les structures existantes telles que les groupes n'offrent pas à l'utilisateur la possibilité de rechercher des objets en leur sein (il n'était pas possible pour Élodie de trouver immédiatement les touches doubles dans la sélection contenant l'ensemble des touches). Une technique telle que le « Graphical Search & Replace » (Kurlander & Bier, 1988) permet quant à elle de rechercher des objets selon un ensemble de critères et de diminuer ainsi l'effort requis pour les modifier.

L'exigence (R1.2) d*ésigner* fait généralement suite à une étape de recherche, et correspond à la capacité qu'ont les utilisateurs de désigner des objets d'intérêt parmi

des ensembles (par exemple par une sélection) ou encore de désigner des propriétés de ces objets ainsi que de désigner des actions à réaliser sur ces objets (en utilisant par exemple un instrument). Par exemple, dans le scénario de travail du clavier virtuel, Élodie désignait des labels à modifier, elle désignait leur propriété de taille, et l'action de modification de la taille.

Nous avions identifié le besoin de *structuration* dans les scénarios de travail et la capacité des structures à augmenter le *pouvoir d'action* des utilisateurs sur leurs éléments dans l'état de l'art. L'exigence (R1.3) *modifier* décrit le fait qu'il est indispensable de pouvoir modifier ces ensembles (en ajoutant, supprimant des éléments). Par exemple, nous avons vu qu'une structure telle qu'un groupe ne supporte généralement pas ces opérations, obligeant l'utilisateur à détruire le groupe pour le reconstruire avec les éléments à ajouter (ou sans ceux à supprimer).

Enfin, les utilisateurs doivent pouvoir *identifier* (R1.4) quels objets appartiennent à un ensemble particulier, mais aussi inversement, pouvoir identifier à quels ensembles appartient un objet donné. Ceci afin de répondre aux besoins de *structuration* et de *recherche*. Par exemple, quels sont les objets qui sont dérivés d'un objet « master » particulier, ou quels sont les groupes dans lesquels un objet sélectionné est inclus.

2. Gérer les actions

Gérer les actions correspond à la capacité des utilisateurs à spécifier interactivement la nature d'une action (R2.1), ses paramètres (R2.2), et sa portée (R2.3). Ces exigences sont liées au besoin d'un *pouvoir d'action* augmenté afin d'interagir avec plusieurs objets (la portée) de façon synchrone. Par exemple, l'utilisateur peut définir la nature d'une opération d'alignement en cliquant sur l'icône correspondant ou en sélectionnant la commande dans un menu. Il peut ensuite décider d'un alignement « vertical » ou « horizontal », paramètre de l'action, et l'appliquer à tous les objets d'une sélection, la portée. Percevoir les conséquences (R2.4) grâce à un retour d'information approprié permet à l'utilisateur de réaliser les effets de l'action immédiatement après l'avoir accomplie (Shneiderman, 1987), voire même avant s'il est possible de donner un aperçu du résultat avant validation (Vermeulen, Luyten, van den Hoven, & Coninx, 2013).

3. Favoriser la conception exploratoire

Le cinquième besoin des utilisateurs que nous avions identifié dans les scénarios de travail était celui d'*explorer* des solutions. Les travaux de (Terry & Mynatt, 2002a) ont révélé trois besoins propres à l'activité de conception exploratoire. Pour pouvoir

favoriser la conception exploratoire, il est important de fournir aux utilisateurs des outils qui leur permettent d'essayer (R3.1) et d'évaluer des solutions (R3.2) lors d'une exploration à court-terme (R3.3), et de comparer différentes versions de ces solutions lors d'une exploration à moyen-terme (R3.4). S'il est satisfait des résultats obtenus, l'utilisateur doit pouvoir étendre les modifications réalisées à d'autres objets. Si le système ne le permet pas efficacement, l'utilisateur va être contraint de répéter les mêmes actions manuellement afin de propager les changements, il s'agit alors d'un problème de viscosité (T. R. G. Green, 1989). Pour finir, si la structuration est une solution à la viscosité, cela devient un frein à la conception exploratoire s'il est nécessaire de le faire en amont d'une conception (cf. les compromis entre dimensions cognitives). C'est pourquoi la structuration devrait pouvoir être réalisée a posteriori (R3.5), c'est-à-dire après que les actions aient été réalisées.

B. Conception d'outils interactifs

Les enquêtes contextuelles se sont avérées utiles pour étudier un problème propre à l'édition graphique concernant la répétitivité des tâches et pour révéler le besoin d'interagir avec de multiples objets. Nous avons souhaité continuer à impliquer les utilisateurs que nous avions interviewés dans le processus de conception de solutions en leur proposant de participer à des séances de génération d'idées et de prototypage vidéo.

1. Ateliers de conception

Un premier atelier de conception a été profitable à la génération d'idées dont certaines ont été à l'origine des solutions développées présentées plus loin. Nous en présentons quatre d'entre elles.

a) Manipulation de propriétés

Dans un contexte d'édition d'agenda numérique, un premier prototype présente l'abstracteur de propriétés (Figure 46). Cet outil recense l'ensemble des propriétés contenues dans les objets graphiques sélectionnés et permet d'effectuer des modifications simultanées. Deux manipulations sont proposées pour réaliser ces modifications : la première consiste à déposer une valeur dans la partie supérieure de l'abstracteur délimitée par des pointillées afin de donner cette valeur à tous les objets de la sélection, la seconde manipulation est de déposer une valeur sur une autre afin de la remplacer. Il permet aussi de générer des nouveaux objets dont les propriétés sont égales aux propriétés communes aux autres objets, et de créer des clones. L'intérêt de ce prototype est de permettre de modifier efficacement des propriétés

dans des ensembles d'objets (ici des évènements d'un calendrier). Il a inspiré la conception de ManySpector présenté en page 91.

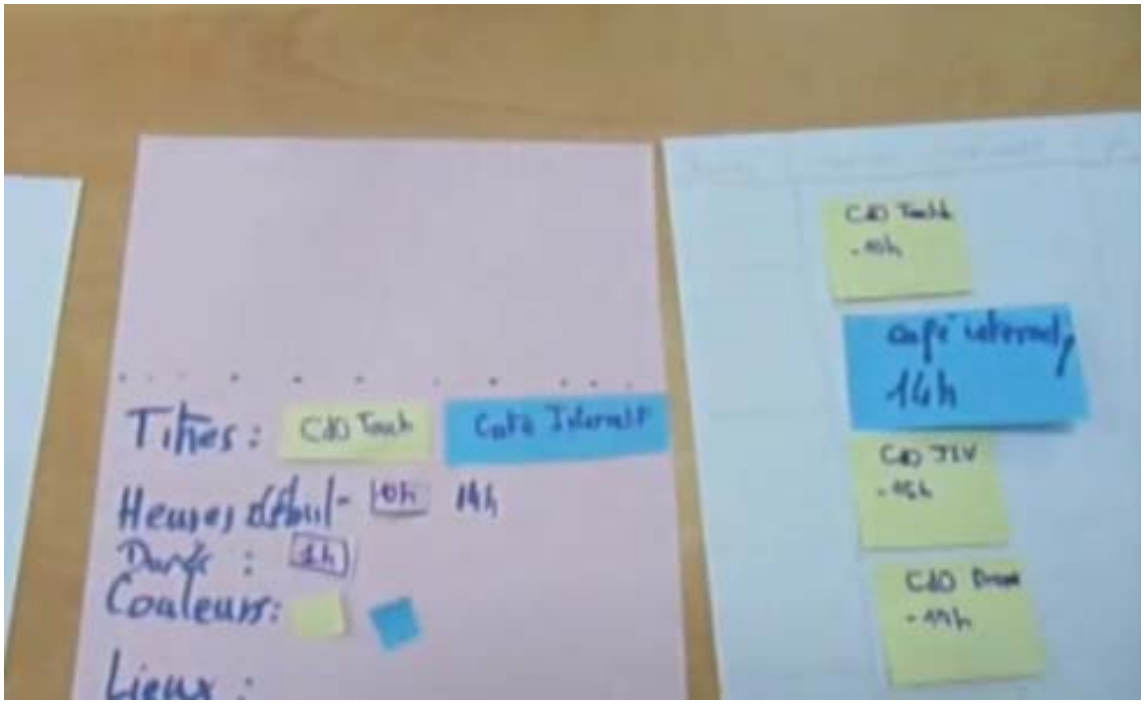


Figure 46. L'abstracteur de propriétés. A droite un éditeur d'agenda, à gauche l'abstracteur. L'abstracteur contient des propriétés trouvées dans des objets de la vue agenda, et permet de modifier ces propriétés ou de générer d'autres objets ayant ces propriétés.

Le prototype suivant présente un moyen interactif permettant d'accéder aux propriétés d'un objet (Figure 47). En cliquant sur un objet, ses propriétés sont affichées autour de lui en formant une spirale. Cela permet non seulement d'éditer ces propriétés, mais aussi de sélectionner l'ensemble des objets ayant des propriétés identiques ou sensiblement identiques (selon une précision définissable à l'aide d'un instrument). Ce principe de sélection étendue à été implémenté dans ManySpector, sous la forme de méta-instruments qui permettent de rafiner une sélection selon des valeurs de propriétés. L'affichage des propriétés en spirale permet également de faire apparaitre des liens de dépendances entre propriétés d'objets différents, ce qui a été source d'inspiration pour l'outil permettant de créer des liens de délégation présenté ultérieurement.

a) Dépendances a posteriori

Une autre idée présente un moyen interactif permettant d'établir des dépendances entre objets. A partir d'une conception graphique préalablement réalisée, il est possible de basculer dans une nouvelle vue recensant les différentes propriétés des éléments de la scène (Figure 48), une vue qui contient également des histogrammes donnant le nombre d'occurrences d'éléments pour une taille donnée, une position, une couleur

etc. Le regroupement des éléments dans cette vue permet d'identifier des clusters et l'utilisateur peut alors définir manuellement des dépendances au sein des groupes.

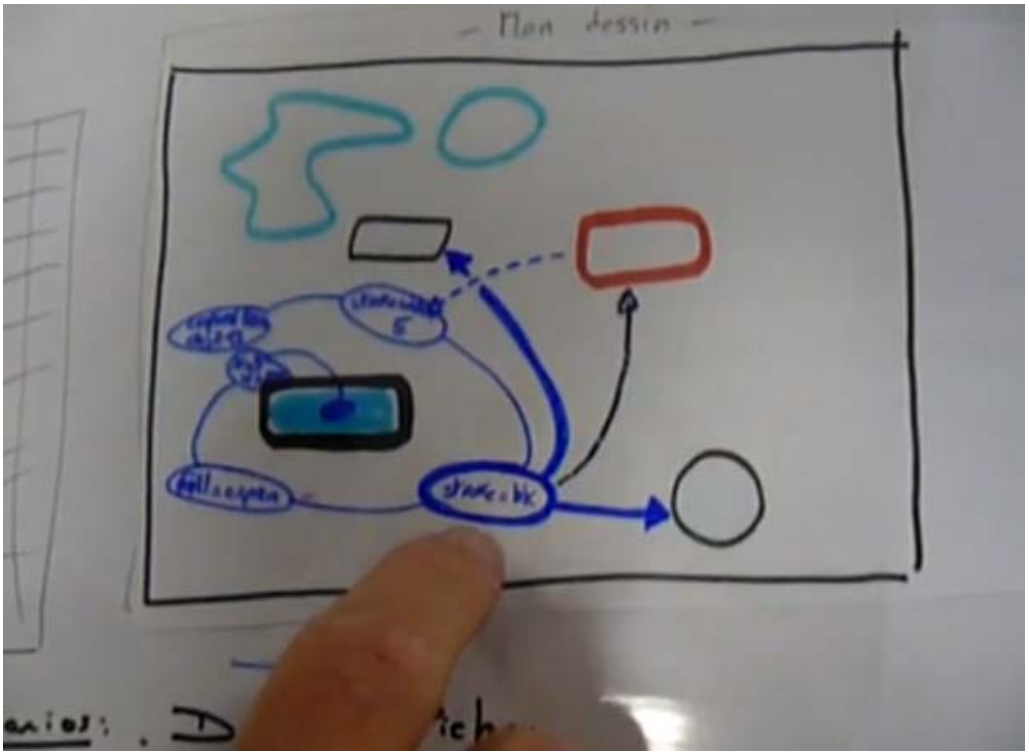


Figure 47. Les propriétés du rectangle bleu sont affichées en spirale autour de lui. Des flèches indiquent des objets ayant des valeurs de propriété identiques.

Une autre idée favorisant le concept de structuration a postériori est celle dite de la « carte à liens » (Figure 49). En prenant l'exemple d'un compte rendu de réunion contenu dans une document textuel, la carte à lien correspond à une vue où tous les éléments textuels sont réorganisés selon leur proximité spatiale et sémantique, afin de faire émerger de l'information et faciliter la création de liens entre ces éléments. Ces fonctionnalités facilitent alors la mise en forme et la correction du document.

2. Réalisation d'un éditeur graphique

Les solutions ont toutes été implémentées dans un éditeur graphique, entièrement développé pour les besoins de la thèse. L'éditeur permet de dessiner des objets graphiques dans une scène et d'interagir avec le contenu de cette scène par manipulation directe. Il existe divers types d'objets graphiques manipulables : des formes primaires (rectangles, ellipses, lignes), du texte, mais aussi des évènements de calendrier et des trajectoires 4D. L'intérêt de cette diversité est de montrer que les concepts des solutions proposées peuvent être généralisés à tout type d'application graphique et non exclusivement à des outils de dessin, tout comme la problématique touche l'édition graphique dans son ensemble.

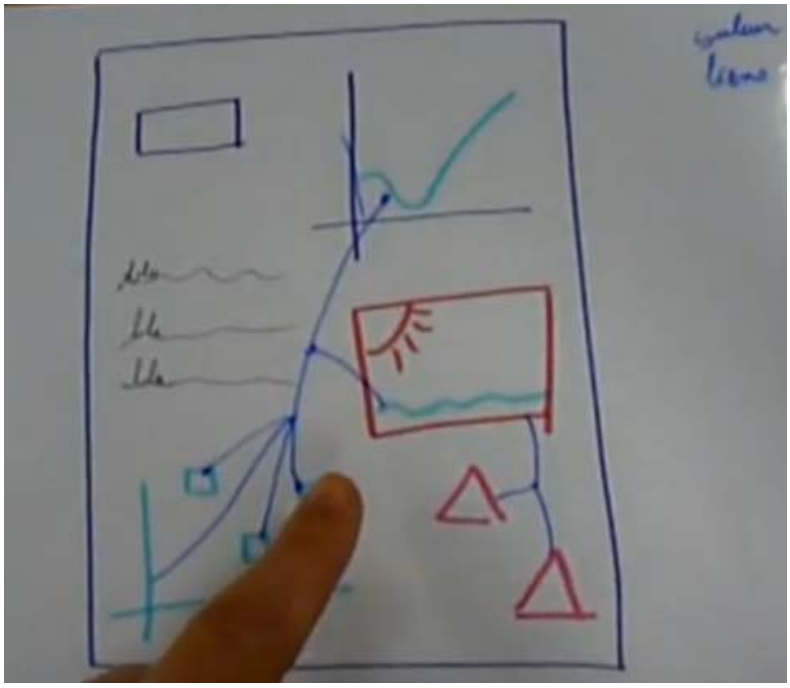


Figure 48. Cette vue réorganise les propriétés graphiques d'une scène selon leur occurrences, afin de faire émerger des clusters.

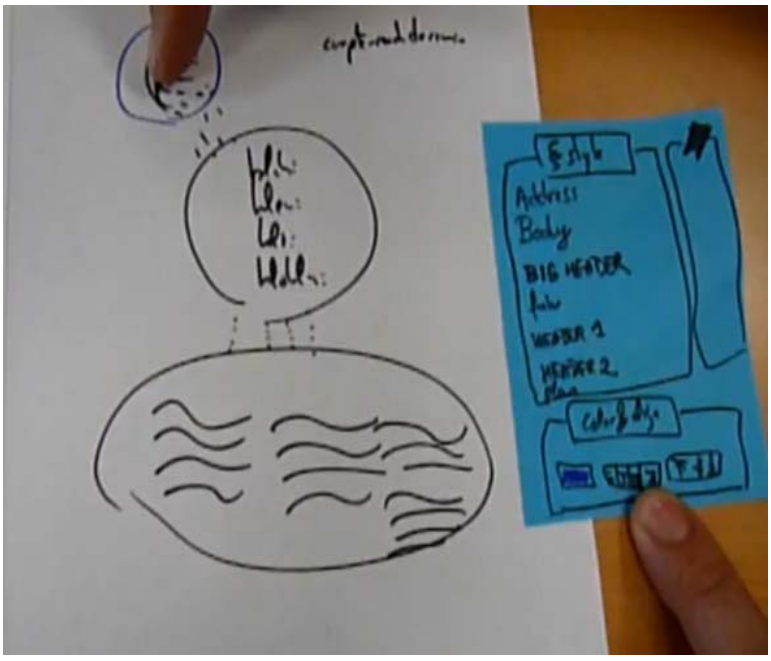


Figure 49. La carte à liens est une idée d'outil réorganisant les éléments d'un document textuel selon leur proximité sémantique (regroupement des titres de niveau 1, regroupement des mots soulignés etc.) dans une autre vue afin de faciliter des opérations sur ses éléments.

a) Vue d'ensemble de l'application

L'éditeur est composé d'une fenêtre principale contenant l'espace de travail, et de plusieurs fenêtres flottantes affichables à souhait et déplaçables telles que la palette

d'outil ou l'inspecteur de propriété (Figure 50). Nous présentons certaines d'entre elles dans cette section.

L'espace de travail

L'espace de travail est la vue principale de l'application. Il présente l'ensemble des objets créés par l'utilisateur, et permet de les modifier. Il existe deux façons d'ajouter des objets dans l'espace de travail : la première, classique, consiste à dessiner par manipulation directe la boîte englobante de l'objet à ajouter après avoir sélectionné un type d'objet dans la palette d'outils. Un feedback de couleur turquoise est présenté à l'utilisateur jusqu'à ce qu'il valide l'opération. La seconde consiste en une opération de glisser-déposer d'un échantillon de forme dans la scène ce qui a pour effet de créer un objet dont les paramètres par défaut ont été spécifiés par l'utilisateur (en sélectionnant des valeurs par défaut dans la palette d'échantillons de valeurs).

La sélection d'objet peut être réalisée en cliquant directement sur les objets, ou bien en dessinant un rectangle de sélection englobant plusieurs objets, tel que dans les éditeurs graphiques traditionnels (R2.3 *portée des actions*). Un objet sélectionné se voit englobé d'une bordure dorée et de poignées rectangulaires qui permettent leur redimensionnement.

Les panneaux d'échantillons

Les panneaux d'échantillons contiennent un ensemble de valeurs de propriétés et de formes. Il existe des échantillons de couleur de fond (représentés par une tâche colorée), des échantillons de couleur de contour (représentés par des tâches dont seul le contour est coloré), et des échantillons d'épaisseur de contour (représentés par des cercles noirs de différentes épaisseurs). Pour modifier une propriété d'un objet de la scène, l'utilisateur peut glisser et déposer un échantillon directement sur l'objet en question. Un feedback est dévoilé dès que l'échantillon survole l'objet, ce qui permet à l'utilisateur de comprendre l'action et d'évaluer les changements avant de les appliquer réellement en relâchant le bouton de la souris (Figure 51). Il devient donc possible d'annuler l'action en relâchant le bouton à l'extérieur de n'importe quel objet (R3.1 *essayer,* R3.2 *évaluer,* R3.3 *explorer à court-terme,* R3.4 *comparer,* R2.4 *percevoir les conséquences*). Un feedback donné avant la résolution de l'action est appelé feedforward (Djajadiningrat, Overbeeke, & Wensveen, 2002)(Vermeulen et al., 2013).

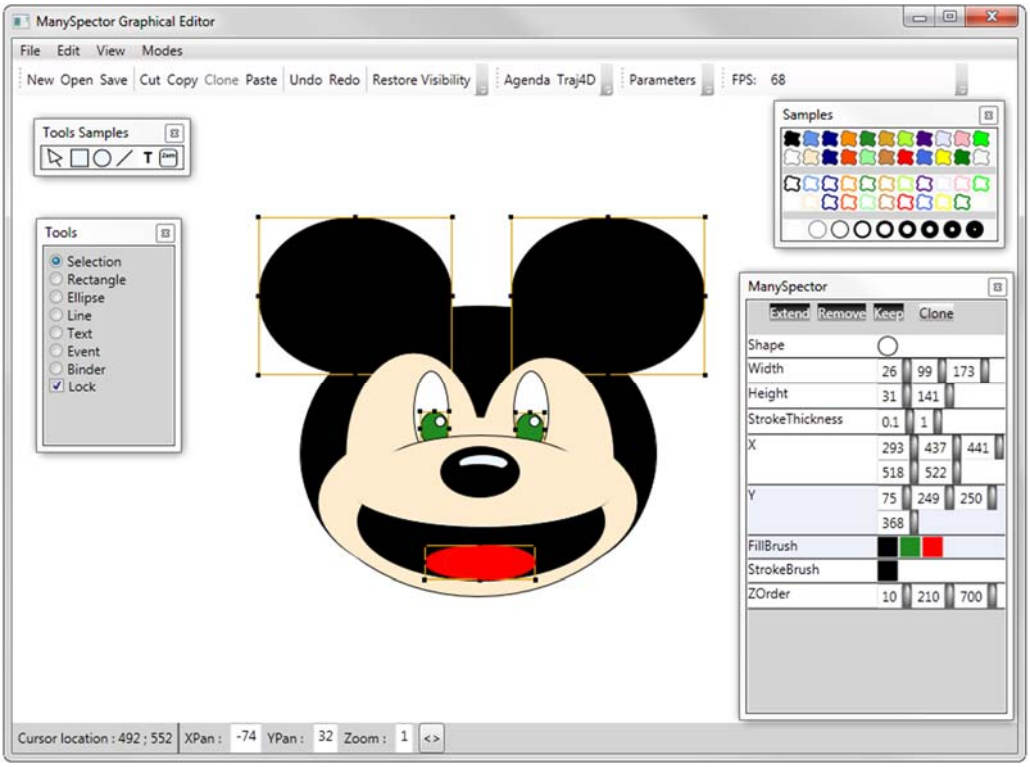


Figure 50. Fenêtre de l'éditeur, avec en haut à gauche des échantillons de formes, au milieu à gauche la palette d'outils, en haut à droite le panel d'échantillons de valeurs, et en bas à droite l'inspecteur de propriétés.

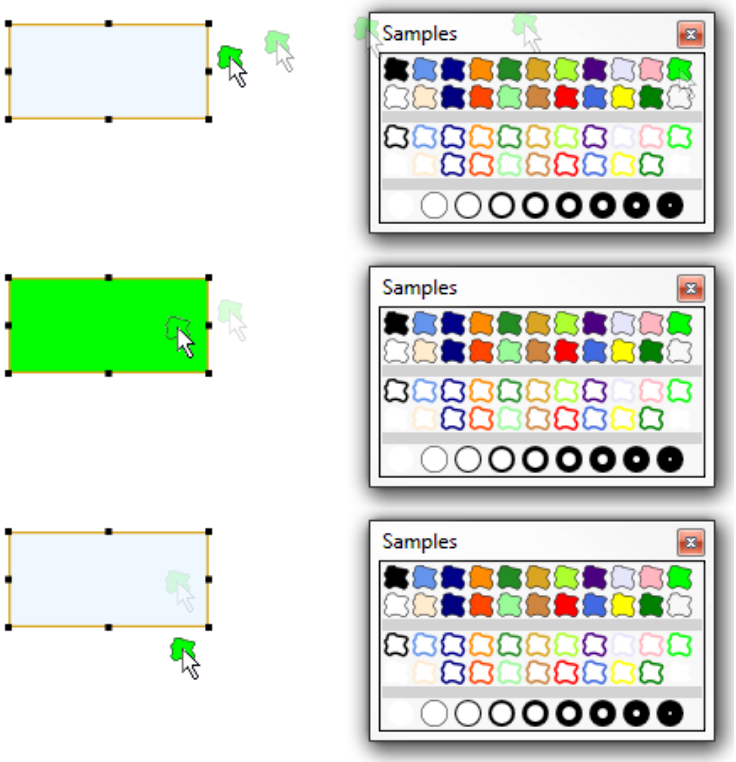


Figure 51. L'utilisateur saisit un échantillon de valeur du panel d'échantillons. Il survole ensuite un rectangle de la scène qui donne un feedback montrant quel serait l'état du rectangle si l'utilisateur validait son opération en déposant l'échantillon. L'utilisateur quitte ensuite le rectangle sans relâcher le bouton de la souris, qui reprend sa couleur d'origine.

Les opérations de glisser-déposer, ou « drag and drop », couvrent une part importante des interactions de l'application. Elles sont caractérisées comme étant une composition d'une source, d'un verbe, et d'une destination. Dans le cas d'utilisation des échantillons de valeurs, la source est une valeur de propriété, par exemple un fond vert (R2.2 *paramètre de l'action*), le verbe est « modifier » (la propriété couleur de fond, R2.1 *nature de l'action*), et la source est un ou plusieurs objets graphiques, par exemple un rectangle (Figure 51) (R2.3 *portée de l'action*).

Puisque l'on ne peut déposer un échantillon de valeur que sur un seul objet à la fois, pour éviter à l'utilisateur d'avoir à répéter une même opération de modification sur plusieurs objets différents, il est également possible de déposer un échantillon sur une sélection d'objets afin de tous les modifier (R2.3 *portée de l'action*) (Figure 52).

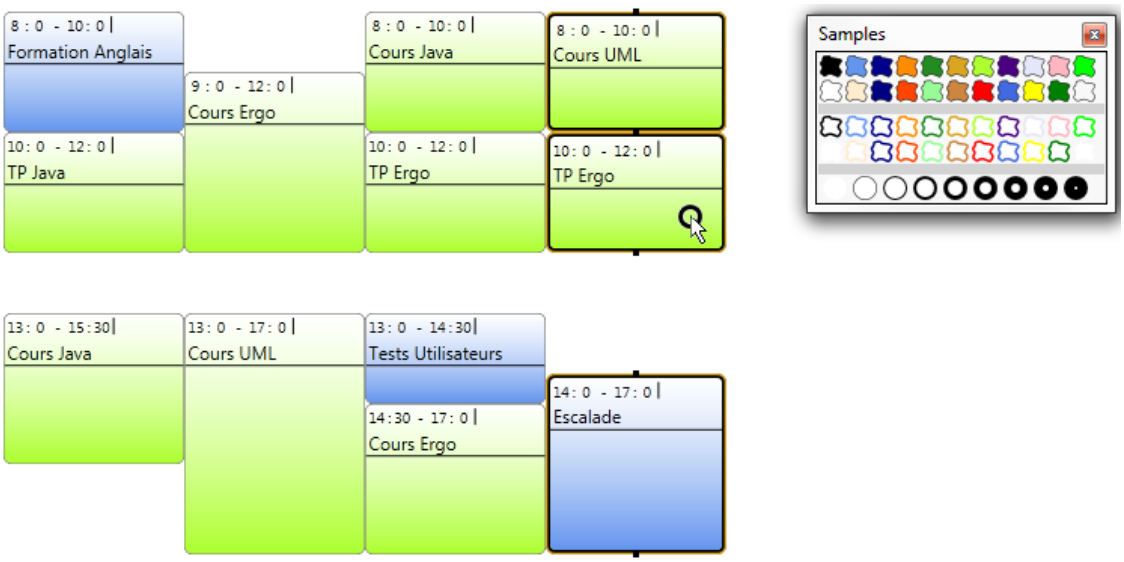


Figure 52. Déposer ou survoler un objet sélectionné avec un échantillon l'applique à tous les objets de la sélection.

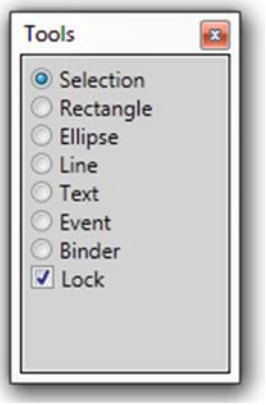


Figure 53. La palette d'outils.

Outils

La palette d'outils permet de sélectionner l'opération courante qui sera interprétée dans l'espace de travail à l'utilisation de la souris (Figure 53). Par défaut, cette opération est la *sélection* : l'utilisateur peut cliquer sur un objet pour le sélectionner, maintenir la touche « shift » pour ajouter ou enlever d'autres objets, ou bien tracer un rectangle de sélection pour sélectionner plusieurs objets. Les outils *Rectangle*, *Ellipse* et *Line* permettent de créer par manipulation directe les formes éponymes. L'outil *Text* permet d'ajouter des objets contenant du texte dans la scène, et l'outil *Event* permet d'ajouter des évènements d'agenda. Pour finir, le *Binder* est un outil qui permet de créer des liens de dépendances entre objets. La case à cocher *Lock* (cochée par défaut), permet de verrouiller un outil : si l'outil n'est pas verrouillé, le système bascule automatiquement en mode sélection après son utilisation.

L'inspecteur de propriétés

L'inspecteur de propriétés, affiche les valeurs de propriétés de l'objet sélectionné et permet de les modifier (Figure 54). Nous présentons plus en détail ce nouveau type d'inspecteur, appelé ManySpector page 91.

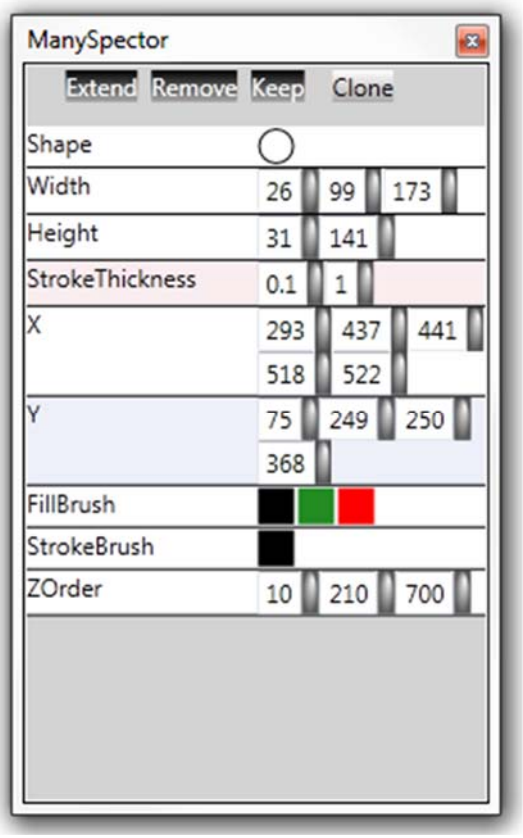


Figure 54. ManySpector, l'inspecteur de propriétés.

b) Édition d'agenda et de trajectoires4D

En plus des fonctions de dessins de formes basiques, l'application dispose d'un éditeur d'agenda et de trajectoires. Les utilisateurs peuvent manipuler les évènements d'un agenda dans une vue semaine, représentés par des rectangles contenant diverses informations telles que le nom de l'évènement, son horaire, et d'autres informations optionnelles (participants, lieu etc.). Les évènements sont organisés horizontalement selon le jour de la semaine et verticalement selon les heures de début et de fin. L'écran est divisé en sept colonnes, une par jour de la semaine (Figure 55). Les évènements peuvent être déplacés par manipulation directe, et son guidés magnétiquement sur les colonnes des jours et les lignes des heures par tranche de quart d'heures.

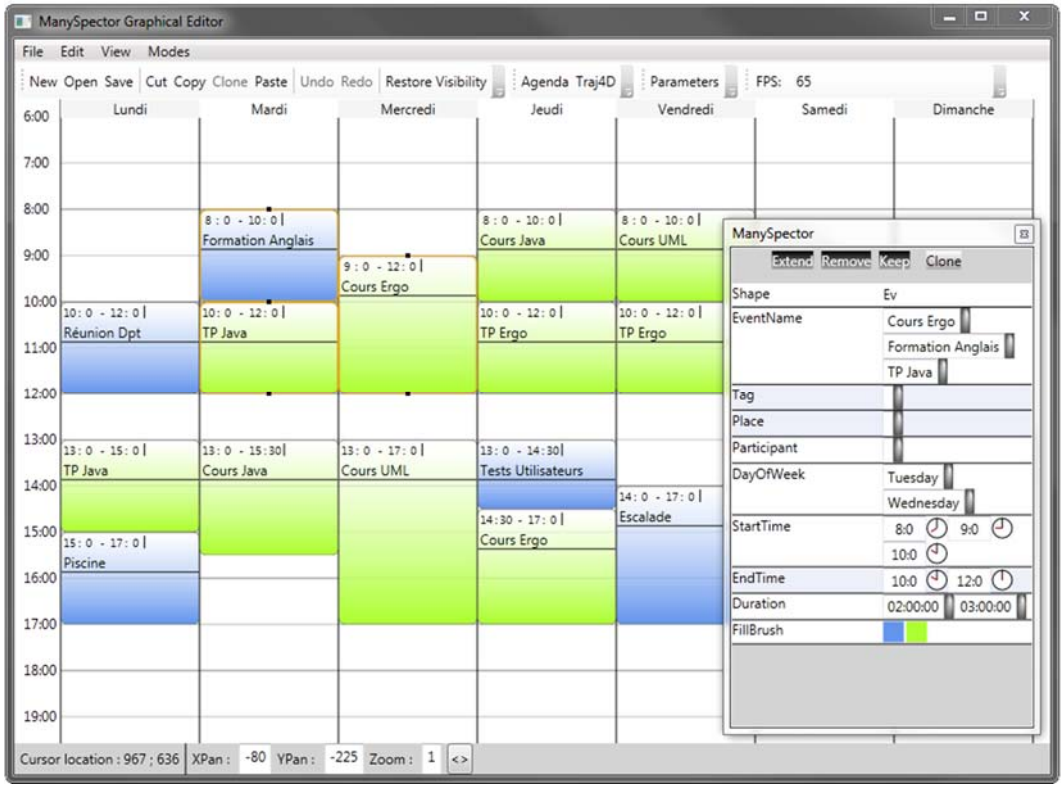


Figure 55. Vue calendrier de l'application.

Il existe également un mode d'édition de trajectoires. Cela permet d'éprouver nos techniques dans un contexte où l'information est véritablement dense et complexe (Figure 56). Les trajectoires sont des trajectoires 4D, définies par une position spatiale en latitude, longitude et altitude et par une donnée temporelle. Chaque point d'une trajectoire est un objet graphique manipulable : il peut être sélectionné, modifié et déplacé. Une trajectoire est aussi un objet en soi qui peut être manipulé de la même façon. Ce mode d'édition a été conçu afin d'évaluer nos outils dans des situations plus concrètes (comme pour l'édition d'agenda), mais où l'information est plus dense et dispersée.

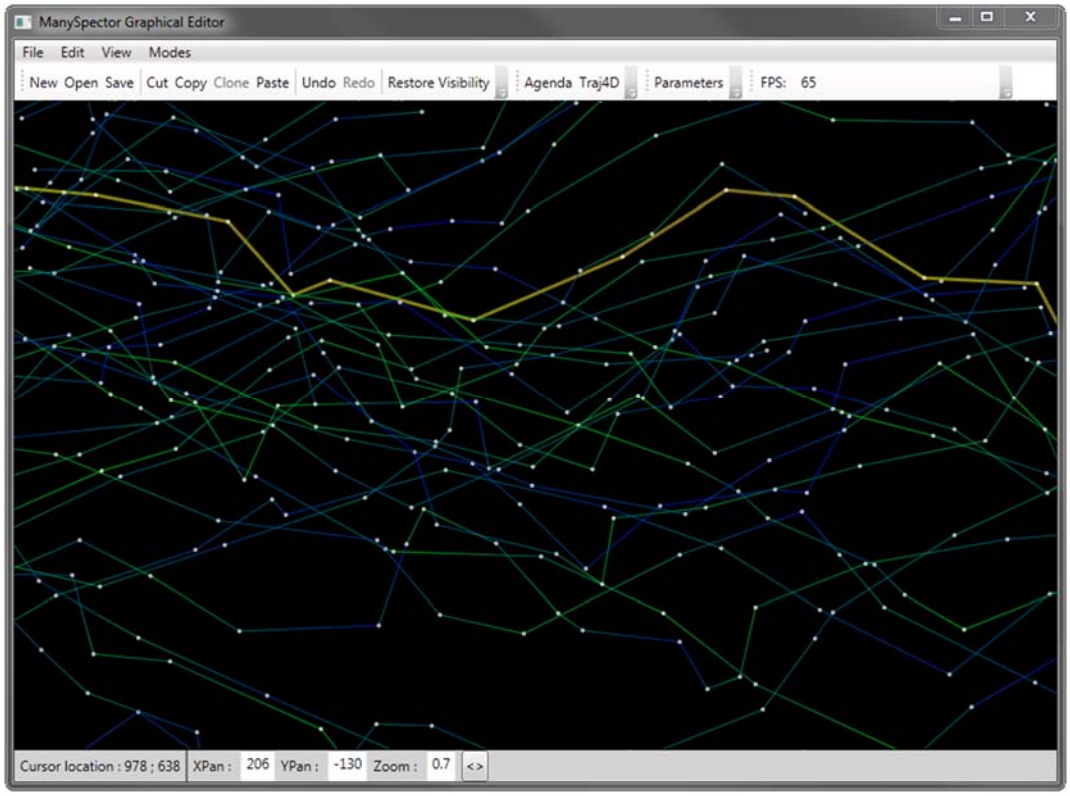


Figure 56. Vue d'édition de trajectoires. Chaque point est un objet graphique manipulable.

c) Implémentation

L'éditeur a été entièrement développé en C# avec le framework WPF. Le langage C# est un langage orienté-objet à classes qui a permis de concevoir l'application selon une architecture de type MVC (Modèle-Vue-Contrôleur). Le bénéfice attendu de WPF est sa capacité à permettre la conception d'application interactives rapidement grâce à son langage de haut niveau (les interfaces se décrivant selon le langage xaml, dérivé du xml).

L'éditeur devant permettre la manipulation d'objets graphiques, il était donc nécessaire de se poser la question de ce qui définit un objet graphique. Puisque nous détenions différents types d'objets (des formes, des évènements) le choix d'implémenter une classe C# par type d'objet est un choix pertinent qui présente des avantages mais aussi des inconvénients. Le type devient intrinsèque à la classe de l'objet et n'est pas défini comme une propriété, ce qui limite les manipulations sur cette propriété mais permet de définir un comportement propre à chaque type (par exemple, aimanter les évènements aux lignes qui définissent les jours de la semaine). Par ailleurs, vouloir ajouter un nouveau type d'objet a un coût puisque cela demande de créer des nouvelles classes et représentations. Une alternative possible aurait été de définir le

type comme propriété d'une classe unique « objet graphique », en différenciant « classe » et « forme » (classiquement en POO on ne différencie pas les deux).

La propriété commune à tout objet graphique est celle de la position spatiale. S'ajoutent les propriétés de couleurs de fond et de contour, et l'épaisseur de contour. Les formes primaires (rectangles, ellipses) ont une hauteur et une largeur. L'objet texte détient une taille de fonte. Les évènements sont caractérisés par des propriétés non-visuelles, comme le jour de la semaine, les heures de début, de fin, la durée, le nom de l'évènement etc. Les trajectoires sont constituées d'un ensemble de points définis par une position spatiale et temporelle. Les outils que nous avons conçus ont pour cible la manipulation de ces propriétés.

C. La structuration explicite

Dans les dimensions cognitives, les abstractions sont présentées comme un remède à la viscosité (T. R. G. Green, 1989). En effet, leur utilisation permet d'agir sur plusieurs éléments en agissant simplement sur l'abstraction. En implémentant un système qui utilise des abstractions, il faut être attentif à ce qu'elles soient utilisables : elles ne doivent pas demander un apprentissage coûteux en temps et en effort, et il ne faut pas qu'elles soient indispensables aux étapes précoces d'une conception pour éviter d'entraver l'exploration (afin de limiter l'engagement prématuré).

Les abstractions sont présentes dans de nombreux outils sous la forme de structures. Ces outils permettent aux utilisateurs de construire leurs propres structures, que ce soit en créant des groupes d'objets, des masters ou autres. Il devient alors possible, pour les utilisateurs, d'interagir sur de multiples objets simplement en agissant sur la structure (*cf. état de l'art page 24)*.

1. La délégation de propriétés

Nous avons exploré des techniques d'interaction qui permettent aux utilisateurs de structurer leur scène explicitement, en s'inspirant du principe de délégation des langages à prototypes (Lieberman, 1986). Les utilisateurs peuvent spécifier qu'une propriété d'un objet (dit clone) dépend de celle d'un autre objet (dit prototype). Un prototype est une entité similaire au master de Sketchpad (Sutherland, 1964) : si l'utilisateur modifie une propriété d'un objet, cela propage le changement à tous les objets clones qui en dépendent (R1.3 *modifier les ensembles,* R2.3 *portée des actions*). La délégation dans Self (Ungar & Smith, 1987) repose sur ce type de fonctionnement (Figure 57). Si l'on modifie un prototype qui délègue des propriétés à d'autres objets, ceux-ci peuvent refléter les mêmes changements. Cependant les changements ne sont

pas visibles immédiatement car les objets ne sont pas notifiés explicitement, il faut cliquer sur chaque objet pour le mettre à jour ou bien rafraichir la fenêtre (Figure 58 à Figure 59) (R2.4 *percevoir les conséquences*).

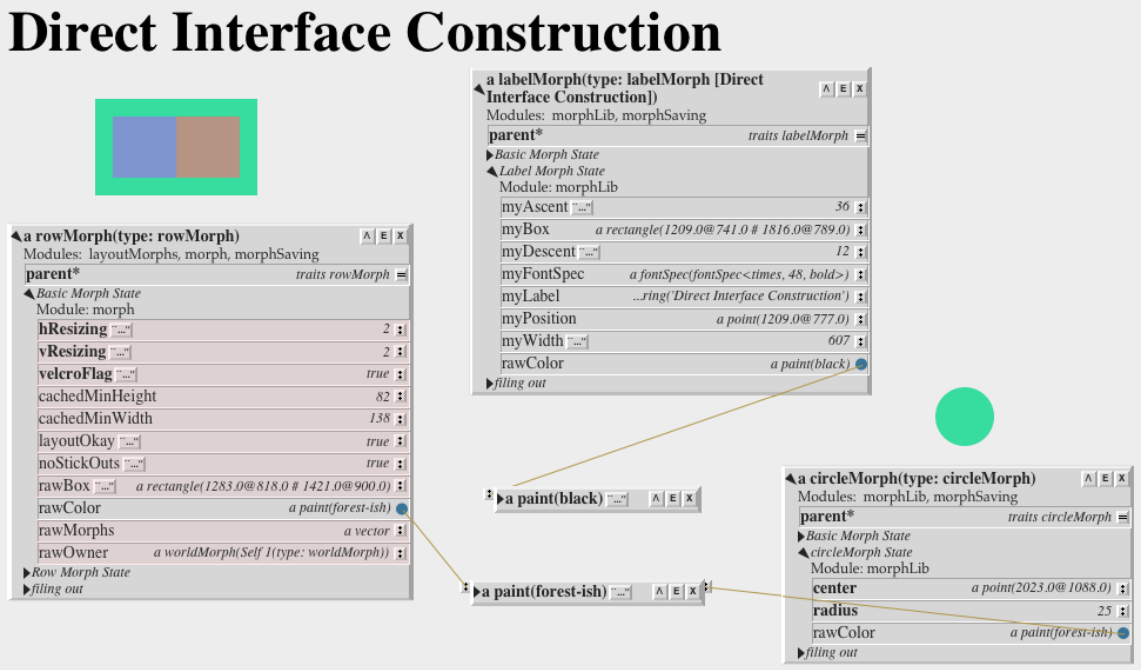


Figure 57. Scène graphique dans Self contenant trois morphes (objets graphiques) et leurs prototypes : un label, un cercle, et un agrégat de morphes (un rectangle contenant deux carrés).

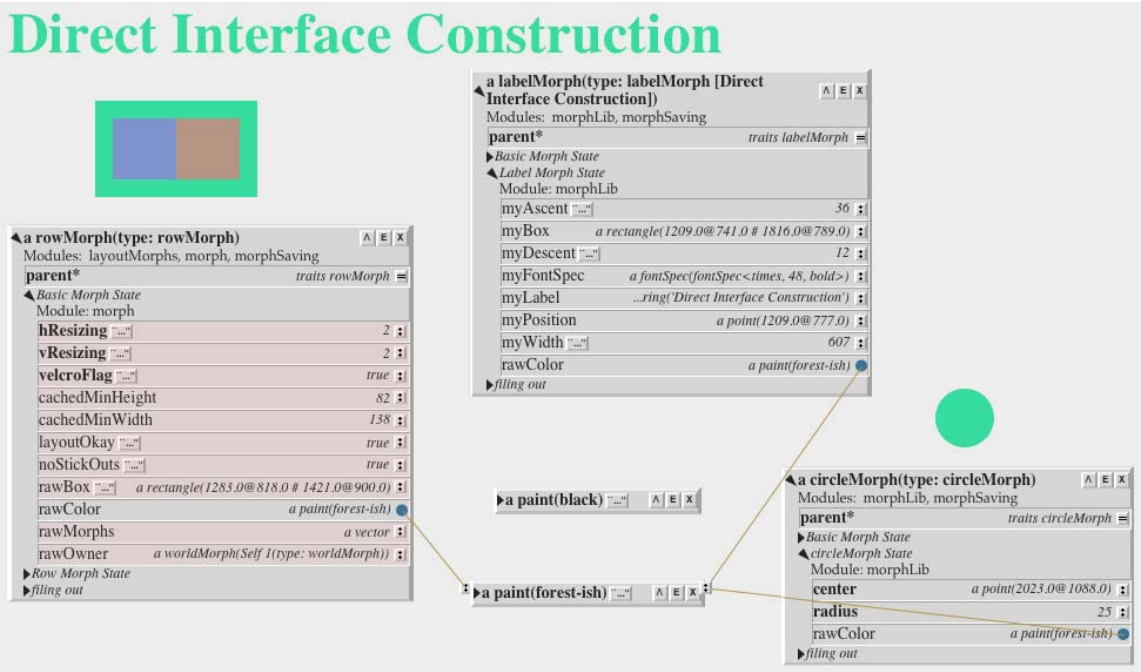


Figure 58. La couleur du label était déléguée par un prototype « peinture » détenant la propriété couleur noire. Le lien de délégation a été déplacé par manipulation directe sur le prototype codant la couleur verte déléguée également aux deux autres objets.

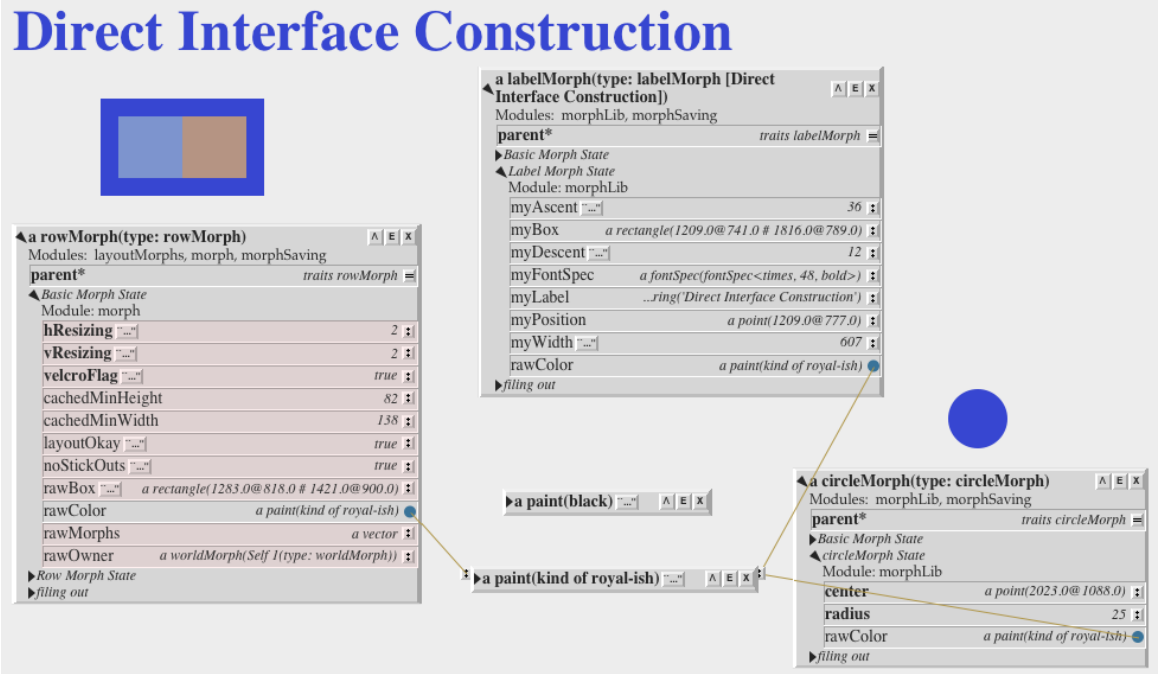


Figure 59. Le prototype « peinture » qui codait la propriété « couleur verte » à été modifié pour coder la couleur bleue. Les trois objets qui dépendaient de ce prototype sont modifiés.

a) Interactions

Nous souhaitions permettre aux utilisateurs de créer leurs structures avec facilité, en favorisant des outils conçus pour être utilisés avec de la manipulation directe, plutôt que via des boîtes de dialogues ou des commandes à saisir. De fait, cliquer sur un objet de la scène en utilisant l'outil « binder » de la palette d'outils dévoile ses propriétés autour de lui (Figure 60).

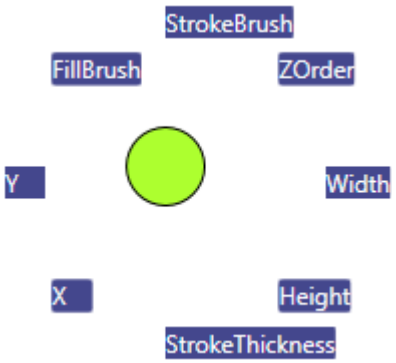


Figure 60. Les propriétés du cercle sont présentées en cercle autour de l'objet.

L'utilisateur peut alors cliquer sur l'une de ces propriétés et tracer un lien élastique qu'il va pouvoir déposer sur un autre objet, de la même manière qu'il déposerait un échantillon de valeur sur un objet (Figure 61). Cet objet devient alors un clone du premier, puisque la valeur de sa propriété devient dépendante de celle de l'objet source, le prototype. Si l'utilisateur modifie ultérieurement la propriété du prototype

qui est déléguée, la modification se propage et affecte le clone. L'apparence de l'objet clone reflète immédiatement la dépendance dès lors que la propriété concernée est représentée visuellement (R2.4 *percevoir les conséquences*).

Les liens de dépendance sont visibles, pour tous les objets, en partant de la propriété vers l'objet. Un lien vert indique que la propriété est déléguée à l'objet et un lien violet qu'elle est dépendante de l'objet (Figure 62). Il est possible de supprimer des liens en cliquant dans un espace vide de la scène et en traçant un trait coupant les liens à supprimer (Figure 63). Les liens en intersection avec la ligne de suppression sont grisés (*feedforward* (Vermeulen et al., 2013)) indiquant à l'utilisateur quels seront les liens supprimés lors de la validation de l'action.

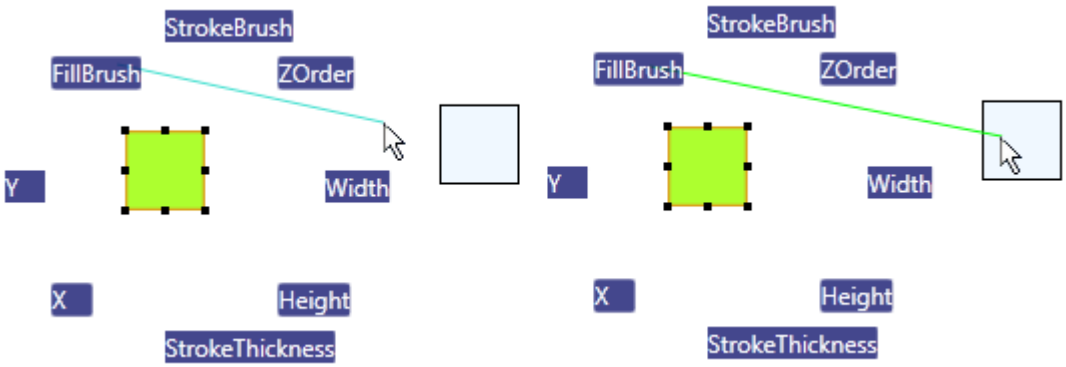


Figure 61. L'utilisateur trace un trait de la propriété du premier rectangle vers le second. Celui-ci devient vert afin d'indiquer à l'utilisateur que l'opération est possible.

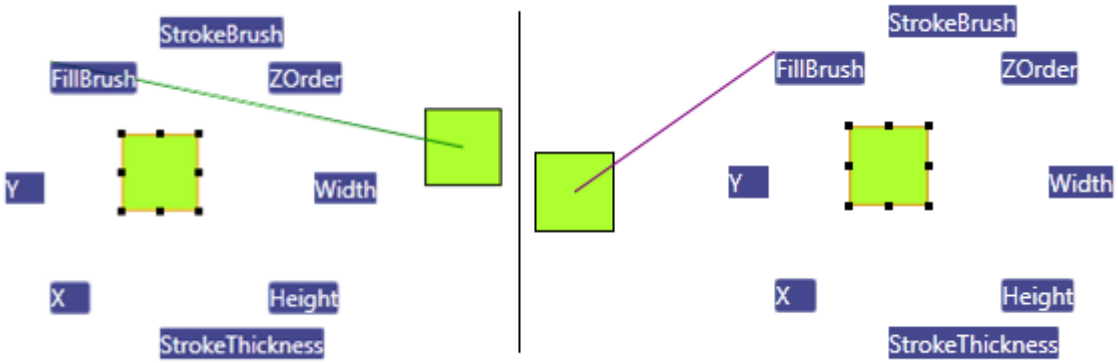


Figure 62. A gauche, le lien vert signifie que la propriété est déléguée à l'objet, tandis qu'à droite, le lien est de couleur violette pour indiquer que la propriété est déléguée par l'objet.

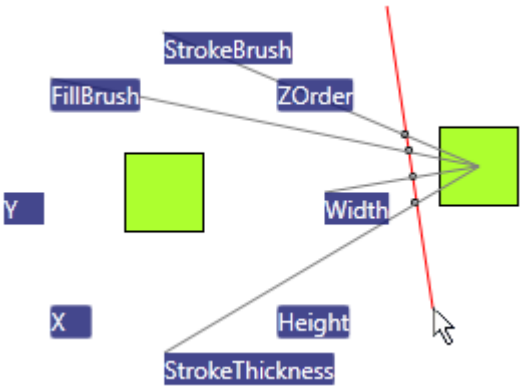


Figure 63. L'utilisateur peut tracer un trait en cliquant dans le vide, et couper des liens qu'il désire supprimer. Tous les liens en intersection avec le trait tracé sont grisés afin d'indiquer qu'ils seront supprimés au relâchement du bouton de la souris.

b) Clonage

Le système propose deux façons de créer de nouveaux objets à partir d'objets existants : soit en copiant l'objet, soit en le clonant. Copier un objet consiste en l'opération classique de copie des éditeurs graphiques : les propriétés de l'objet copié sont totalement indépendantes de l'objet source. Le clonage a pour effet de créer un clone dont presque toutes les propriétés sont déléguées par l'objet source (entre autre : couleur, forme, taille ainsi que titre, heures et jour de la semaine pour un évènement d'agenda). Seules les propriétés définies explicitement ne sont pas déléguées, telles que la position du nouvel objet qui est généralement différente du prototype. Créer un clone minimise le nombre d'actions requises pour spécifier une seule différence entre un clone et le prototype : s'il était copié au lieu d'être cloné, cela demanderait d'établir des liens un par un pour chacune des propriétés.

Par ailleurs, le système mémorise quels sont les objets « sources » dont est issue une copie simple. L'utilisateur peut alors décider ultérieurement de transformer l'objet copié en objet cloné, en déposant un instrument « clone » sur celui-ci (Figure 64). L'intérêt réside dans le fait que cela permet de créer des liens a posteriori (R3.5 *structurer a posteriori)* : l'utilisateur n'est pas obligé de prendre prématurément la décision de créer un objet délégué ou indépendant, ce qui limite l'engagement prématuré (T. R. G. Green, 1989). Il peut choisir d'effectuer une simple copie et de prendre cette décision plus tard. Cette opération de transformation de copie en clone n'affecte que les propriétés qui n'ont pas été déléguée explicitement par l'utilisateur.

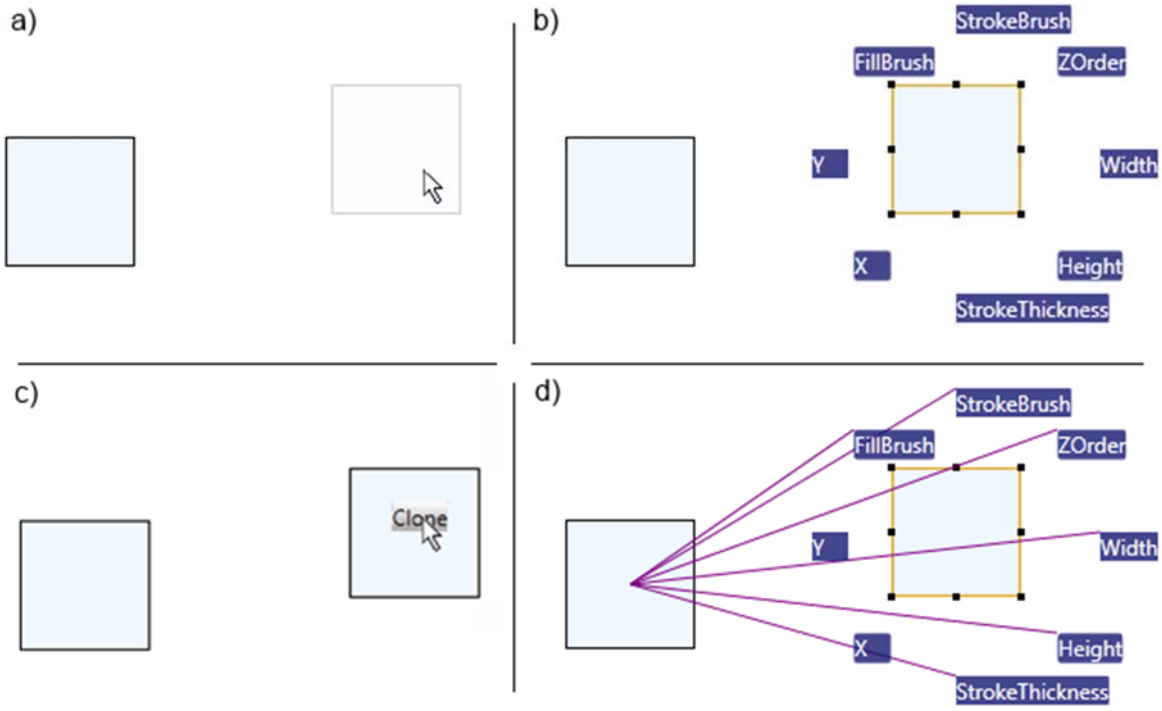


Figure 64. L'utilisateur copie un objet (a) la copie n'a aucun lien de dépendance avec l'original (b). L'utilisateur dépose l'instrument « clone » sur la copie (c), celle-ci est transformée en clone de l'objet original (d).

2. Les tags, aide à la structuration et outil de développement

Les tags, tels que démontrés dans X11, Tcl/Tk (Ousterhout, 1994) et Swingstates (Appert & Beaudouin-Lafon, 2006) sont des étiquettes que l'on « appose » à des objets afin de construire des ensembles hétérogènes par extension. Grâce aux mécanismes des types délégués du langage C# qui permettent d'encapsuler des méthodes, nous pouvons définir des collections de données qui peuvent appliquer automatiquement des traitements aux données lorsqu'elles sont ajoutées ou supprimées de la collection, et de sauvegarder n'importe quelle propriété si nécessaire. Par exemple, un tag « SelectedObject » peut, lorsque l'on décide de l'attribuer à un objet, modifier la propriété « IsSelected » de l'objet à *Vrai* et changer la couleur du contour de l'objet par la couleur verte (s'il s'agit de l'attribut caractérisant un objet sélectionné) après avoir sauvegardé la couleur d'origine qui peut être restituée au retrait du tag.

Dans notre application, nous utilisons les tags pour les liens à supprimer par exemple (Figure 63). Tous les liens en intersection avec la ligne de suppression sont tagués « à supprimer », modifiant leur couleur en gris. La couleur d'origine d'un lien lui est ainsi restituée au retrait du tag, lorsque l'utilisateur déplace la ligne de suppression de telle sorte qu'elle ne soit plus en intersection avec le lien.

Nous avons exploré des idées d'outils pouvant permettre à l'utilisateur de créer ses propres tags et de définir des opérations applicables aux objets lors de l'ajout ou du retrait du tag (Figure 65). Il ne s'agit que d'une ébauche, les recherches concernant la structuration par les tags étant envisagée dans des travaux futurs.

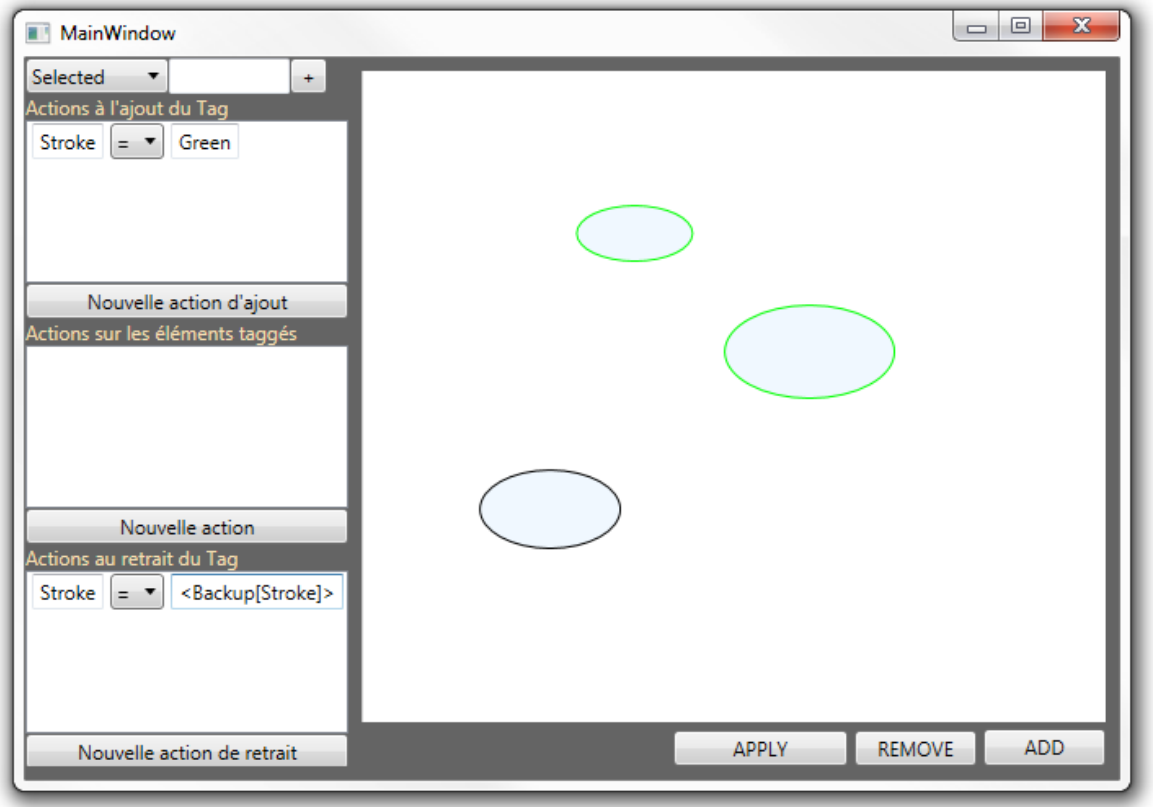


Figure 65. Prototype d'une application permettant de créer ses propres tags pouvant encapsuler des opérations de modification de propriétés.

3. Apports de la structuration explicite

La structuration explicite est supposée pouvoir apporter plus de pouvoir d'action en agissant sur des nœuds de la structure, ce qui engendre une propagation aux objets qui en dépendent. En contrepartie, les utilisateurs se voient contraints de gérer une structure, entravant alors la conception exploratoire. Nous avons donc souhaité réduire cet inconvénient en permettant de structurer a posteriori grâce au mécanisme de copie et clonage réversible. De plus, nous souhaitions offrir des moyens d'interactions avantageux aux utilisateurs pour créer leurs structures tels que la manipulation directe, au lieu de techniques utilisant des boîtes de dialogues modales ou des menus comme c'est le cas dans des applications existantes, afin de limiter l'indirection spatiale et temporelle (Beaudouin-Lafon, 2000).

L'interaction avec plusieurs objets, dans le but de leur faire dépendre d'un prototype donné peut poser problème, compte tenu des propriétés de la manipulation directe (Frohlich, 1993). Une solution visqueuse serait d'interagir avec chacun des objets individuellement afin d'établir des liens entre leur propriétés et le prototype en question. Cette solution coûteuse reflète les limites de la manipulation directe décrites par Frohlich concernant la réalisation de tâche répétitives : l'outil réduit le nombre d'actions à réaliser sur un objet simple et permet d'observer immédiatement les effets de la création de liens. En revanche lorsque l'utilisateur doit créer un nombre de liens important pour plusieurs objets cette technique est limitée. Ce problème est le même si l'on souhaite interagir avec des objets similaires dans le but de leur faire dépendre d'un unique prototype, comme s'il s'agissait d'un master. Par exemple, nous avons présenté dans l'un des scénarios de travail une tâche où l'utilisateur souhaitait modifier des diapositives contenant le plan d'un cours. Ce plan était presque identique à chaque occurrence : le contenu et la forme sont identiques, sauf la police du titre de la partie abordée après une occurrence du plan qui était « en gras » et non « normale ». L'ordre des parties évoluant souvent (activité de conception exploratoire) l'utilisateur devait modifier chaque occurrence du plan individuellement à chaque changement.

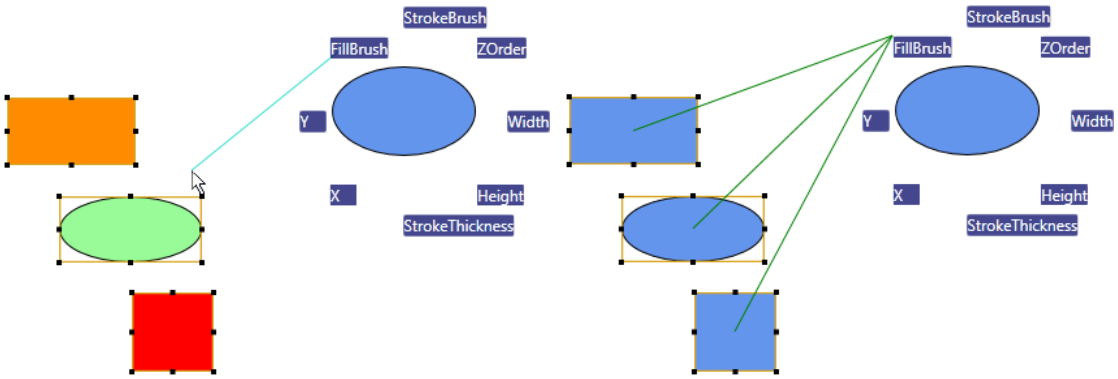


Figure 66. Créer un lien de délégation vers un objet sélectionné, crée également des liens vers tous les autres objets de la sélection.

C'est pourquoi il pourrait s'avérer intéressant de déléguer le contenu d'un plan à tous les autres, afin de n'avoir que celui-ci à modifier. Mais il serait tout de même coûteux de créer répétitivement des liens de délégation sur le contenu d'un plan à tous les autres. Une solution plus efficace que nous avons implémentée pour permettre la délégation multiple consiste alors en la sélection de tous les objets dont une même propriété doit être déléguée par un prototype, puis de déposer cette propriété du prototype sur n'importe lequel des objets sélectionnés (R2.3 *portée de l'action*) (Figure 66).

Le graphe de délégation de propriété est une extension de l'arbre de délégation des langages à base de prototypes. Cependant, dans un arbre, les objets ne peuvent pas avoir plusieurs parents. Par exemple, l'arbre de scène disponible dans Illustrator peut s'avérer utile pour conceptualiser une scène, mais ne permet pas de spécifier des relations croisées entre objets. Contrairement à un arbre, un nœud de notre graphe peut avoir plusieurs parents. Cela permet aux utilisateurs d'être plus spécifiques concernant les liens de parenté de leurs propriétés : un objet peut avoir sa propriété de couleur de fond déléguée d'un prototype 'A' et sa largeur de contour d'un objet 'B'.

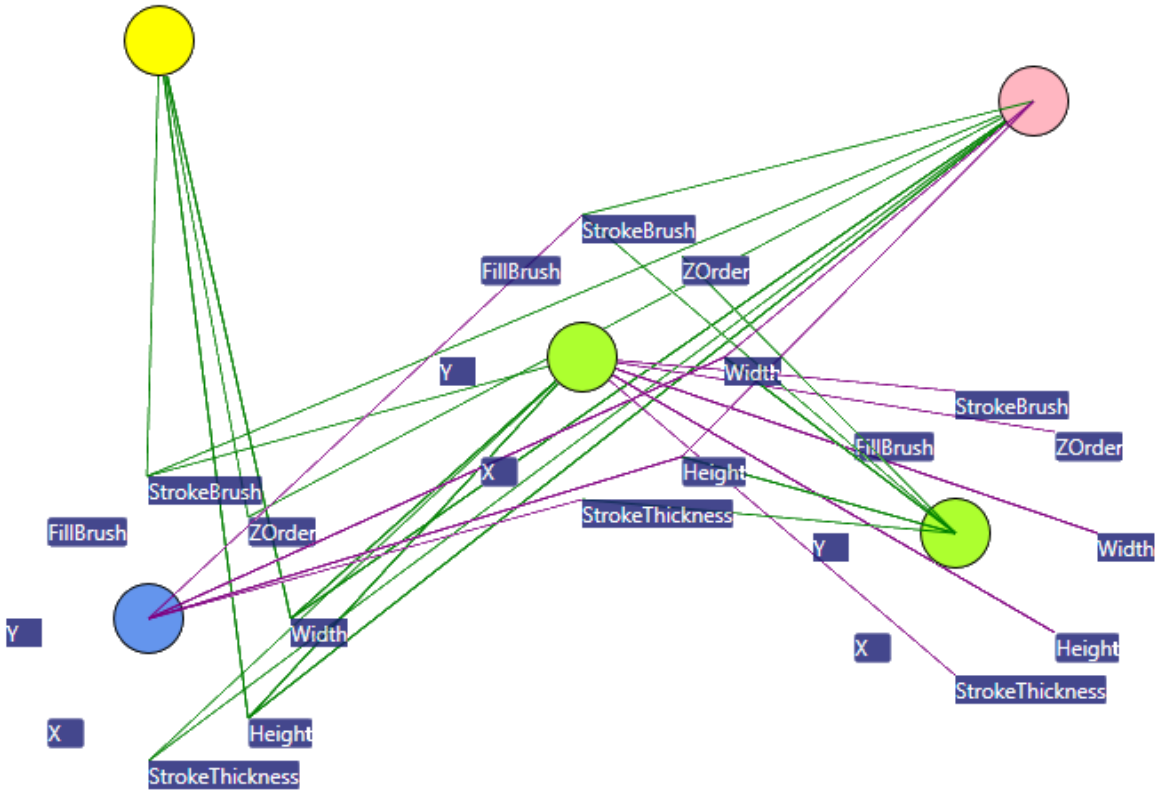


Figure 67. Illustration de la complexité de la visualisation lorsque le nombre de liens augmente.

La visualisation des liens a été définie arbitrairement (lignes droites, couleurs verte ou violette selon la parenté), mais nous pensons que cette technique ne passe pas à l'échelle. Si le nombre de liens crées devient trop important, il peut être difficile de comprendre les relations entre propriétés et objets (Figure 67). La représentation des liens pourrait bénéficier d'améliorations significatives en s'inspirant des travaux du domaine de la visualisation d'information, par exemple grâce au « bundling » (Hurter, Ersoy, & Telea, 2013).

D. La structuration implicite

Créer ses propres structures apporte donc un avantage indéniable, mais au coût d'un effort cognitif supplémentaire. Il faut réfléchir aux structures à créer, et réaliser les interactions le permettant. Nous présentons un autre concept, celui des structures implicites. Une structure implicite n'est pas expressément définie par l'utilisateur, ce sont l'ensemble de ses décisions qui dévoilent via les outils fournis des structures, que l'utilisateur va pouvoir exploiter pour interagir avec un grand nombre d'objets efficacement. Afin d'illustrer ce concept, nous avons conçu ManySpector, un nouveau type d'inspecteur de propriétés, qui permet de définir des structures implicites et qui offre de nouvelles interactions.

1. ManySpector

Un inspecteur est une fenêtre présentant verticalement un ensemble de noms de propriétés et leurs valeurs (Johnson et al., 1989). Par exemple, la Figure 68 illustre la représentation d'un rectangle vert par l'inspecteur de Visual Studio 2010. Un inspecteur offre deux services à l'utilisateur : visualiser des valeurs, et modifier ces valeurs. Pour modifier une valeur, il faut généralement saisir la nouvelle valeur désirée dans le champ de texte correspondant.

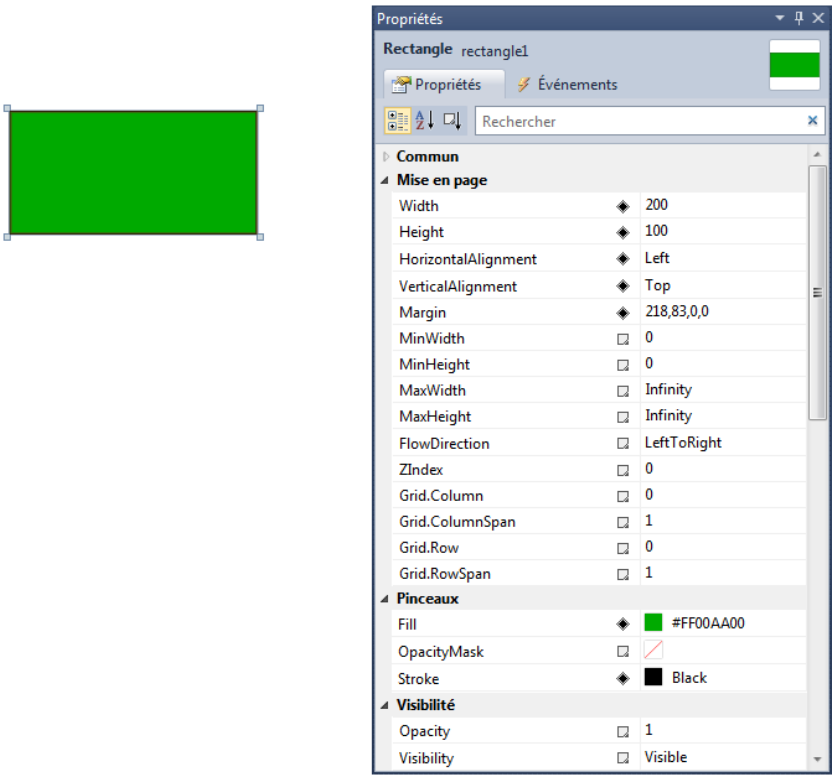


Figure 68. Inspecteur de Visual Studio. A gauche l'objet sélectionné, à droite l'inspecteur présentant les propriétés de l'objet et des champs de textes permettant de les modifier.

Lorsque plusieurs objets de différents types sont sélectionnés, un inspecteur classique ne présente que les propriétés qui sont communes à tous (Figure 69, à gauche). L'utilisateur peut modifier une telle valeur, le système propageant alors le changement à tous les objets de la sélection.

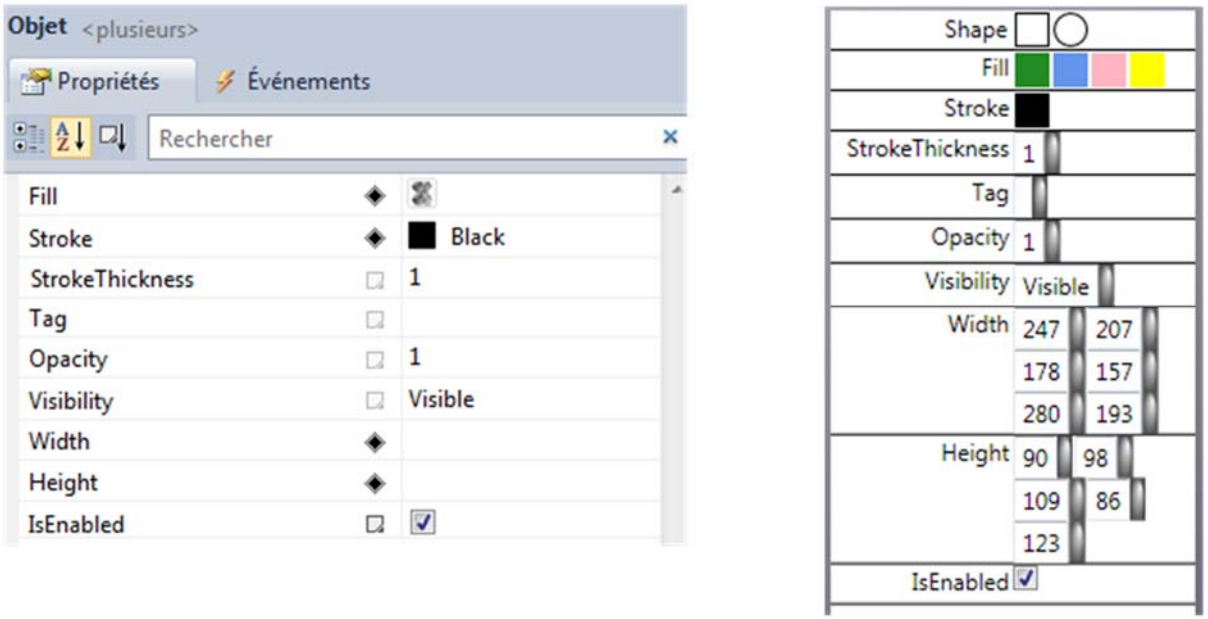


Figure 69. La sélection contient plusieurs objets de tailles et couleurs différentes, seul leur contour est identique (noir et d'épaisseur égale à un). Un inspecteur classique, comme celui de Visual Studio (à gauche) n'affiche que des champs blancs pour les propriétés multi-valuées, tandis que ManySpector (à droite) présente les différentes valeurs existantes pour tous les objets de la sélection.

Il faut noter qu'un inspecteur ne présente aucune information si les valeurs d'une propriété donnée sont différentes pour les objets de la sélection. Cela limite donc non seulement la visibilité des propriétés puisque l'utilisateur n'est pas informé de leurs valeurs, mais aussi la capacité à les modifier. Généralement la seule opération possible revient à donner une même valeur à tous les objets.

Notre volonté en concevant ManySpector était de dépasser ces limites. ManySpector est un inspecteur ayant la particularité d'afficher toutes les valeurs d'une propriété donnée, partagées par un ensemble d'objets pouvant être de type différent (Figure 69, à droite). Par exemple, l'on peut voir sur l'illustration que la propriété couleur de remplissage est valuée avec quatre valeurs différentes dans la sélection. Ce sont les valeurs « partagées ». Les valeurs partagées révèlent une structuration implicite de la scène : l'ensemble des objets qui partagent une valeur pour une propriété donnée. Ces ensembles ne sont pas définis pas l'utilisateur, mais sont le résultat de ses manipulations. Nous pensons qu'ils peuvent s'avérer utiles puisque les utilisateurs peuvent être amenés à penser à des objets selon un prédicat graphique (par exemple,

« tous les objets rouges »). Nous avons réifié ces valeurs partagées dans ManySpector. La réification des valeurs partagées sert de base à l'élaboration d'ensembles de techniques d'interaction, qui offrent des nouveaux services pour la conception exploratoire et l'interaction basée sur les structures : des requêtes de sélection d'objets avec des exemples graphiques, le raffinement d'une sélection et la modification de propriétés sur des objets multiples.

a) Réification d'ensembles

La représentation d'une valeur partagée dans ManySpector réifie (Beaudouin-Lafon & Mackay, 2000) à la fois une valeur en soi, mais également l'ensemble des objets sélectionnés qui possèdent cette valeur de propriété. En tant que valeur, de façon similaire à l'interaction du panel d'échantillons, l'utilisateur peut glisser une valeur partagée de ManySpector et la déposer sur un objet ou un ensemble d'objets sélectionnés dans la scène pour leur donner cette nouvelle valeur de propriété. La source de cette opération est une valeur de propriété, le verbe est « modifier », la destination est un objet graphique. Si la valeur est numérique, l'utilisateur peut utiliser la molette pour incrémenter ou décrémenter la valeur (R2.3 *portée de l'action*). L'ensemble de ces fonctionnalités couplé à un feedback immédiat, permet à la fois l'exploration et l'ajustement précis des propriétés, réduisant ainsi le décalage temporel entre l'action et la visualisation du résultat (Beaudouin-Lafon, 2000).

ManySpector limite l'affichage de valeurs partagées à trois lignes de valeurs pour ne pas surcharger l'espace visuel (Figure 70). S'il existe plus de valeurs que le nombre par défaut, ManySpector permet une révélation progressive de celles-ci en survolant la propriété avec la souris. Son contenu s'élargit alors grâce à une animation qui révèle progressivement les valeurs suivantes (Figure 71).

Puisqu'une valeur partagée réifie également un ensemble d'objets, le fait d'en survoler une avec le curseur de la souris met en évidence dans la scène tous les objets concernés (c'est-à-dire tous les objets de la sélection ayant cette valeur de propriété) en estompant tous les autres objets progressivement avec une courte animation (Figure 72). Cela permet de comprendre et de visualiser les objets sont dont constitués les ensembles (R1.4 *identifier les ensembles*). Ce mécanisme peut également aider à la détection d'anomalies et à leur correction (Figure 73).

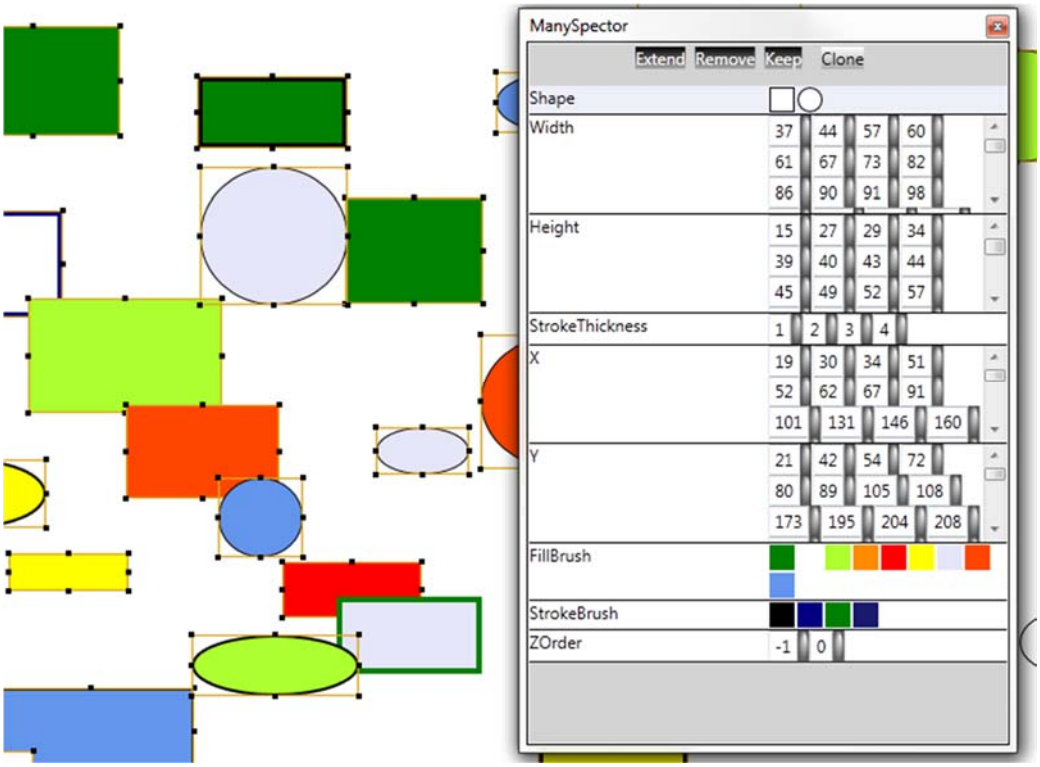


Figure 70. La scène contient plus de 50 objets sélectionnés. Des barres de défilement indiquent la présence d'autres valeurs partagées.

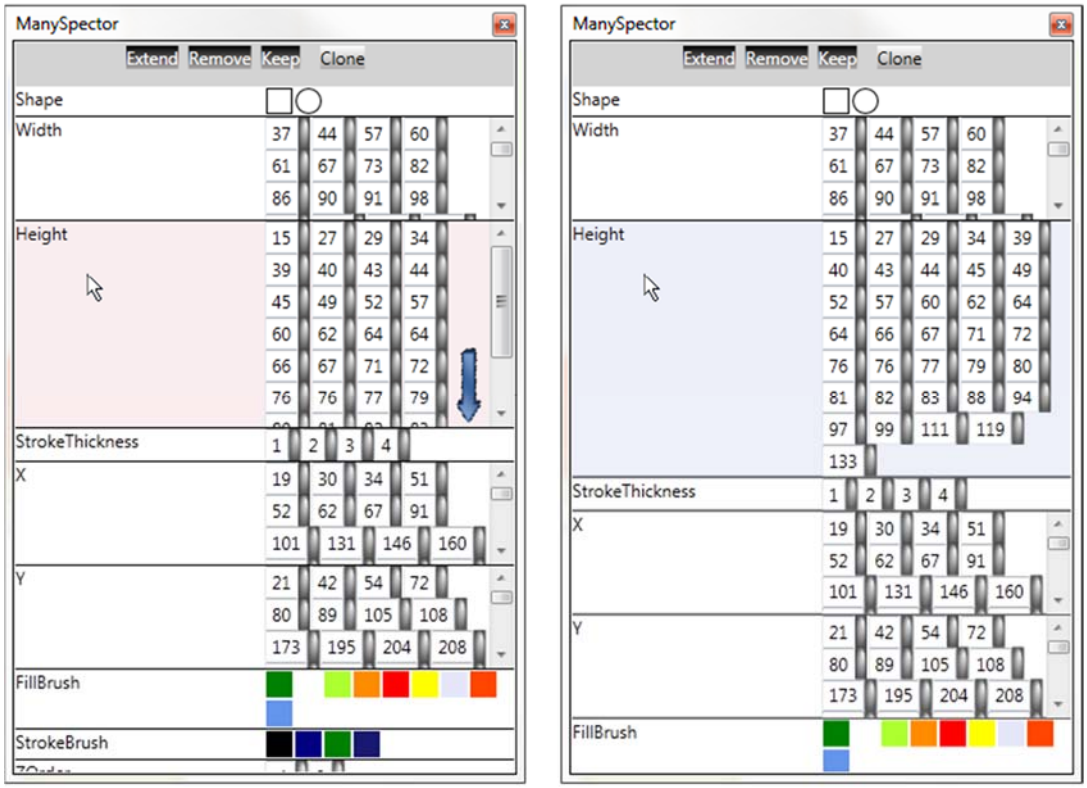


Figure 71. En survolant une propriété dont toutes les valeurs ne peuvent être affichées, une animation étend la hauteur de la zone correspondant à la propriété afin de les rendre toutes visibles et accessibles. En quittant la zone, celle-ci reprend sa taille initiale progressivement passé un court délai.

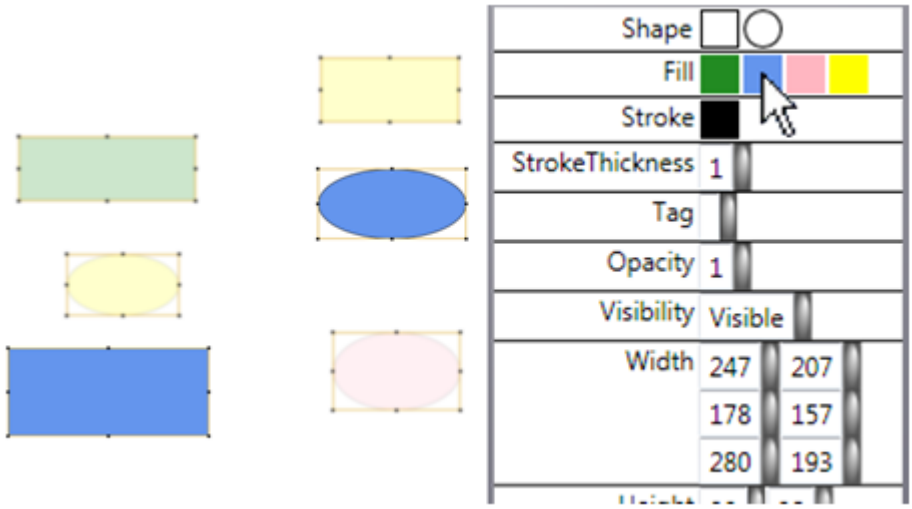


Figure 72. En survolant la valeur partagée « couleur de fond bleue », tous les objets bleus sont mis en évidence : les autre objets sont mis en retrait en réduisant progressivement leur opacité et en leur appliquant un effet de flou.

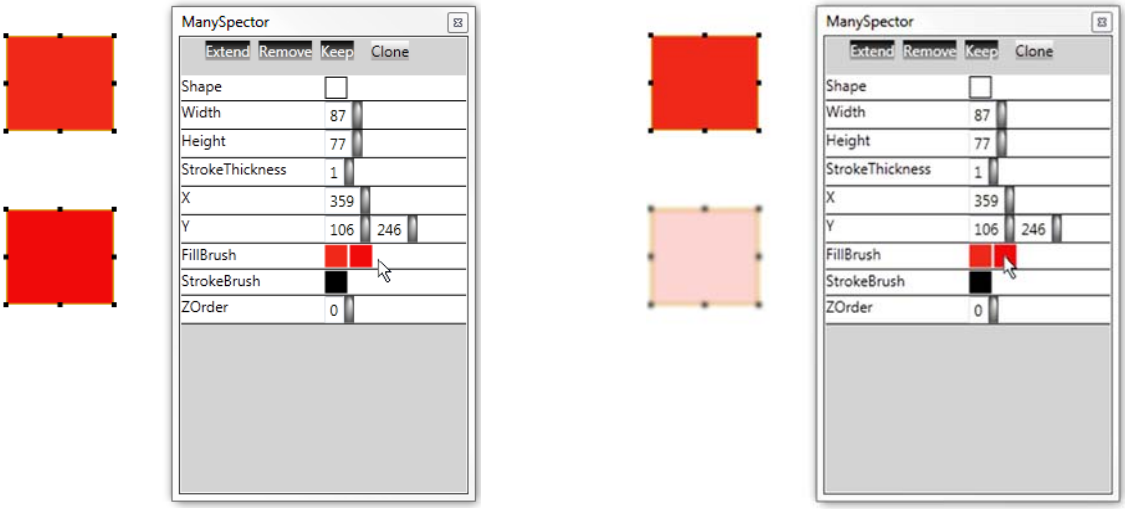


Figure 73. L'utilisateur a sélectionné deux objets rouges qui lui semblaient identiques, mais ManySpector indique qu'il existe deux valeurs partagées différentes. En survolant chacune de ces valeurs, l'utilisateur peut voir quel objet correspond. Il peut alors corriger son erreur en donnant la même nuance de rouge pour tous les objets.

L'utilisateur peut déposer un échantillon de valeur depuis le panneau d'échantillons sur une valeur partagée (considérée ici comme un ensemble), ce qui a pour effet de donner cette valeur à tous les objets de cet ensemble, donc de modifier en une seule fois des objets multiples (R2.3 *portée de l'action*) (Figure 74). Pour cette opération, la source est une valeur de propriété (épaisseur de contour à 6), le verbe est « modifier » (la propriété épaisseur de contour) et la destination est un ensemble d'objet (les objets à fond jaunes). Il est également possible d'effectuer la même opération en utilisant une valeur partagée comme un échantillon de valeur : l'utilisateur

peut saisir une valeur partagée (considérée comme une valeur) pour la déposer sur une autre, attribuant alors à tous les objets de cet ensemble la valeur saisie (Figure 75).

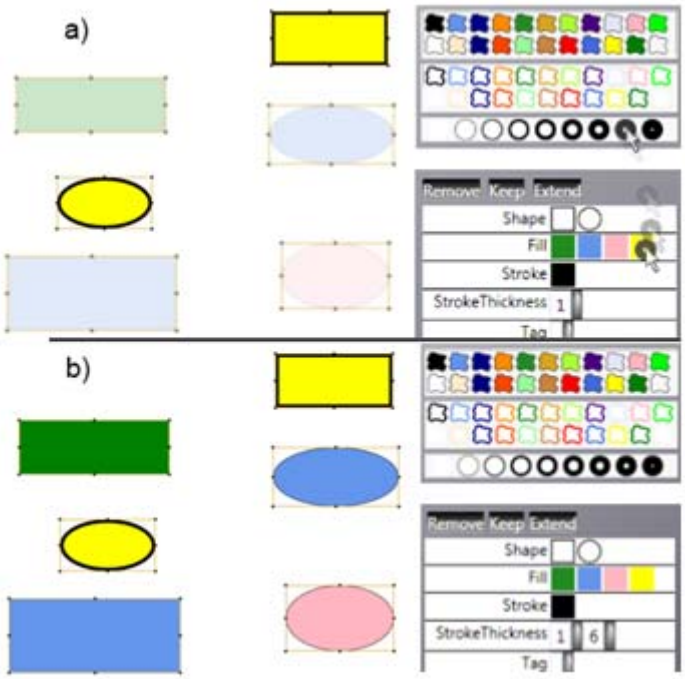


Figure 74. L'utilisateur saisit un échantillon de valeur (contour égal à 6) du panel d'échantillons et survole la valeur partagée fond jaune (a). Tous les objets jaunes montrent alors un feedback en ayant leur épaisseur de contour modifiée à 6 points. L'utilisateur valide l'opération en déposant l'échantillon sur la valeur partagée (b).

b) Requêtes de sélection

Pour sélectionner des objets, l'utilisateur peut cliquer directement sur eux en maintenant la touche « shift » appuyée, ou bien tracer un rectangle de sélection. Dans l'optique de raffiner cette sélection, il peut utiliser trois méta-instruments (c'est-à-dire des instruments qui contrôlent d'autres instruments, ici il s'agit de la sélection) : *Remover*, *Keeper*, et *Extender*. L'interaction consiste en un glisser-déposer de la représentation de l'instrument sur une valeur partagée. *Remover* retire de la sélection tous les objets qui détiennent la valeur partagée (Figure 76). *Keeper* ne garde dans la sélection que les objets ayant la valeur, et retire tous les autres. *Extender* quant à lui ajoute à la sélection tous les objets qui ne sont pas actuellement sélectionnés mais qui détiennent la valeur. Les instruments peuvent également être déposés directement sur un objet de la scène afin de l'ajouter ou l'enlever de la sélection. Ces interactions étendent l'ensemble des requêtes basées sur l'exemple définit précédemment (R1.3 *modifier les ensembles*).

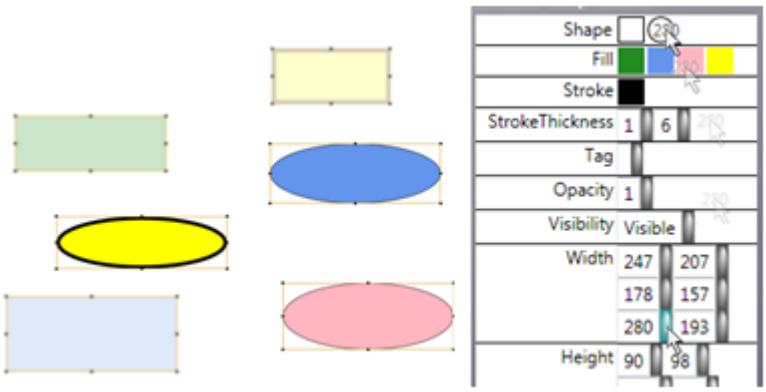


Figure 75. L'utilisateur peut saisi une valeur partagée et l'utiliser comme un échantillon de valeur. Ici il saisit la valeur « largeur : 280 », et la dépose sur la valeur partagée « forme : cercle ». Tous les cercles de la sélection ont alors leur largeur égale à 280.

Figure 76.Exemple d'utilisation des méta-instruments de sélection. L'utilisateur dépose l'instrument « Remove » sur la valeur partagée « fond : bleu ». Tous les objets bleus sont retirés de la sélection.

2. Utiliser les ensembles implicites pour structurer explicitement

Puisqu'une valeur partagée réifie un ensemble, il est possible d'utiliser les outils de la structuration explicite afin de créer des liens de délégation entre un prototype et un ensemble d'objets ayant une valeur de propriété donnée (*réutilisation,* (Beaudouin-Lafon & Mackay, 2000)). L'utilisateur peut donc créer un lien de délégation vers une valeur partagée de ManySpector, ce qui crée un lien du prototype vers tous les objets ayant cette valeur (Figure 77 et Figure 78) (R1.3 *modifier les ensembles*, R2.3 *portée de l'action*). La source de l'opération est une propriété d'un objet graphique, le verbe est « créer un lien de délégation », la destination un ensemble d'objets. Les deux outils (liens et ManySpector) qui sont indépendants peuvent donc communiquer entre eux, ce qui améliore l'homogénéité de l'application et favorise la réutilisation de

ManySpector. Ceci permet d'appliquer des requêtes diverses (autres que la modification de propriétés) à des ensembles.

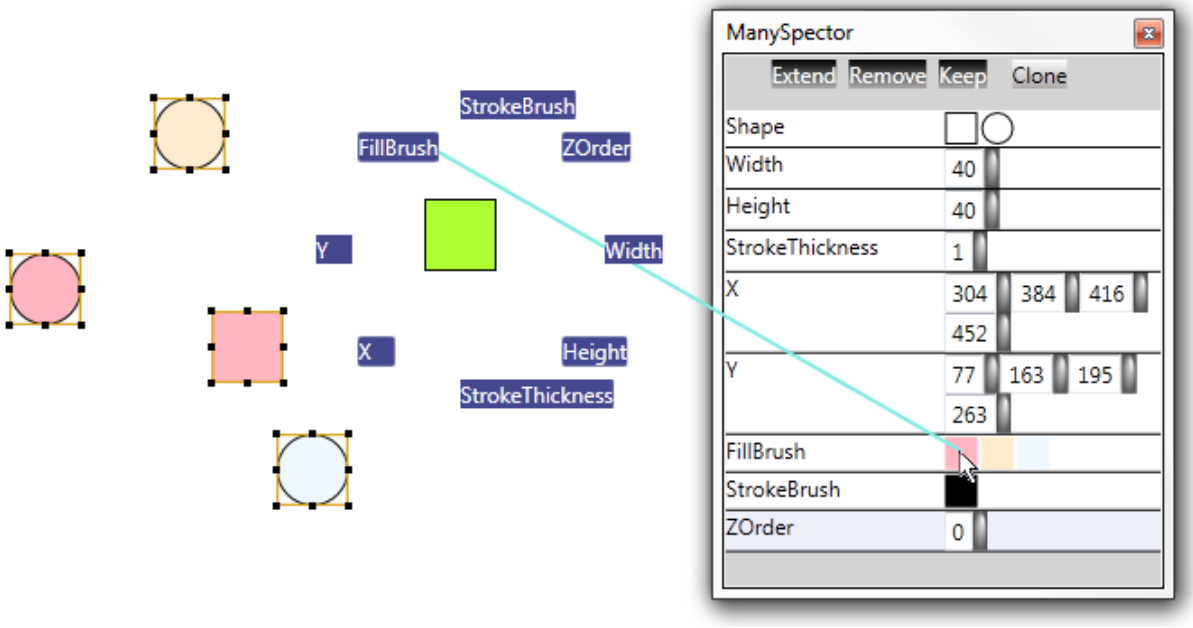


Figure 77. L'utilisateur créé un lien de délégation vers une valeur partagée de ManySpector.

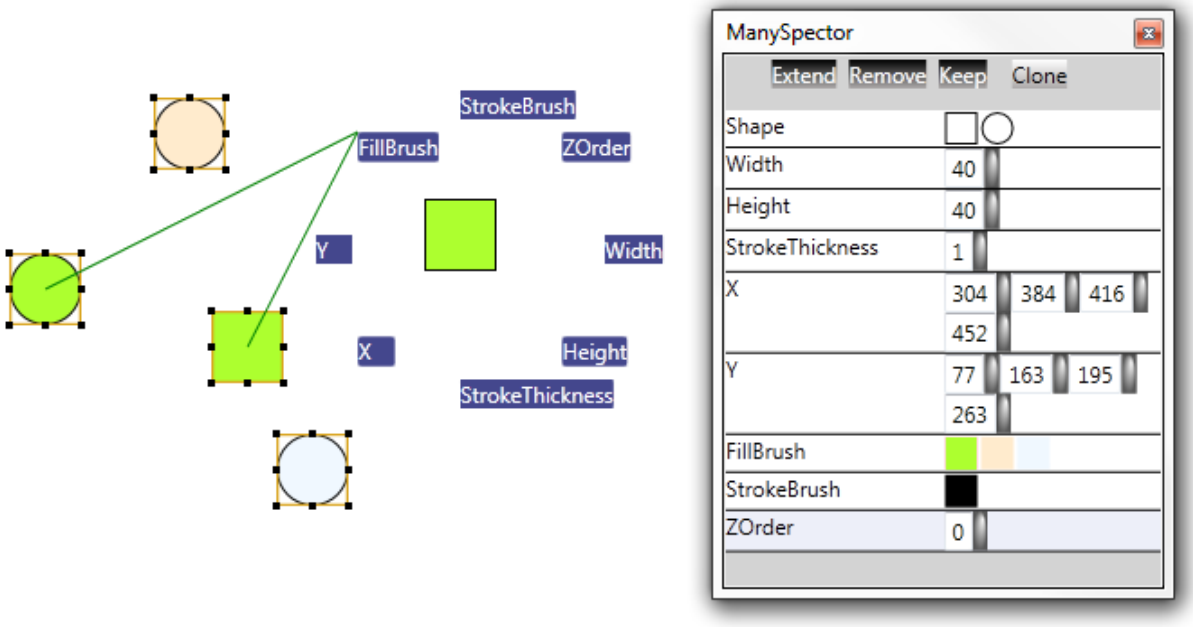


Figure 78. Tous les objets de la sélection ayant la valeur partagée désignée lors de la création du lien de délégation ont leur propriété déléguée par l'objet source.

3. Conclusion sur la structuration implicite

Une structure implicite s'oppose à une structure explicite de par la façon dont elles sont conçues. Nous faisons l'analogie avec les tags intentionnels et les tags extensionnels de SwingStates (Appert & Beaudouin-Lafon, 2006) : les tags

extensionnels sont attribués explicitement à des objets, c'est une décision prise par l'utilisateur qui réalise une ou plusieurs actions pour y parvenir, tandis que les tags intentionnels sont attribués à tous les objets qui satisfont une propriété donnée, sans intervention de l'utilisateur (avec un prédicat, par exemple « tous les objets bleus ») (Appert & Beaudouin-Lafon, 2006). Alors qu'une structure explicite résulte d'une décision et d'un ensemble d'actions de la part de l'utilisateur (dans un but conceptuel ou bien pour réduire la viscosité), une structure implicite ne se révèle que comme une conséquence des actions réalisées par l'utilisateur dans le but d'enrichir sa conception, et non comme une intention de créer une structure. Cela sous-entend que les structures implicites ne sont pas conçues volontairement, mais révélées par le système. Dans notre application elles sont révélées par la sélection.

ManySpector est un exemple d'outil qui exploite le concept de structuration implicite. Ce sont les opérations de sélection qui révèlent, dans ManySpector, une structure implicite de la scène selon les valeurs de propriétés des éléments sélectionnés. Cette structuration implicite offre les avantages attendus d'une structure : permettre la manipulation de plusieurs éléments en une seule fois, en offrant de plus une plus grande granularité concernant les ensembles à modifier (la sélection étant un ensemble et les ensembles définis par les valeurs partagées des sous-ensembles de cet ensemble). Qui plus est, le coût d'abstraction est alors nul, puisqu'aucun effort de structuration n'est demandé à l'utilisateur. Les requêtes de raffinement permettent d'avoir un contrôle explicite sur la structure implicite : l'utilisateur peut modifier explicitement la structure implicite et ainsi exposer une structure optimale en un effort minimal. Il faut noter qu'une structure implicite ainsi obtenue est cependant éphémère, contrairement à une structure définie explicitement comme un graphe de délégation de propriétés. Ce constat est important car les structures sont aussi des moyens qui permettent de conceptualiser une scène. Par exemple, inclure deux rectangles et un label dans un groupe, que l'on nomme « touche » (comme cela a été fait dans le scénario de travail du clavier virtuel), est un moyen de permettre de limiter le nombre d'actions requise sur cet ensemble d'objets, mais aussi de nommer cet ensemble, et ainsi de pouvoir référer à un concept : celui d'une touche. La nature éphémère d'une structure implicite implique qu'elles ne peuvent pas être utilisées à des fins conceptuelles persistantes : l'utilisateur peut se référer par exemple au concept de « tous les rectangles rouges », mais ce concept est perdu dès que la sélection est modifiée.

E. Création automatique de structures explicites

Nous avons vu que les structures sont des techniques efficaces pour interagir avec un nombre important d'éléments en un faible nombre d'actions, puisqu'elles réduisent le nombre d'actions requises pour interagir avec ces éléments, mais au détriment d'un effort cognitif et d'actions supplémentaires. Nous avons identifiés deux inconvénients concernant la technique de structuration grâce aux liens de délégation : la visibilité des liens, et l'effort de création. L'un des principaux inconvénients de la création de structures explicites est justement qu'elles demandent du temps et de l'effort aux utilisateurs (T. R. G. Green, 1989) à tel point qu'ils peuvent préférer ne pas les utiliser (Kurlander, 1993). De plus, devoir maintenir ces structures (on parle « d'abstraction management ») entrave la conception exploratoire. L'utilisateur doit donc au préalable réfléchir à leur construction, s'assurer qu'elles conviennent toujours au cours de son activité de conception, et les modifier dans le cas contraire. C'est pourquoi, il peut s'avérer intéressant de trouver des solutions qui peuvent assister l'utilisateur dans la création de structures explicites, notamment si cela peut se faire a posteriori afin de ne pas limiter la conception exploratoire.

Les principes de « refactoring » ou de programmation par l'exemple vont dans ce sens : laisser l'utilisateur libre d'agir au moment opportun, sans se soucier d'établir une structuration préalable de son travail, en lui fournissant des outils qui lui permettent de faire de la structuration a posteriori. Guru est un outil qui permet de restructurer automatiquement une hiérarchie d'héritage dans les langages à prototypes (Moore, 1995). Afin de pallier le problème de l'effort demandé pour créer une hiérarchie de délégation, nous nous sommes inspirés des travaux de Moore pour factoriser toutes les propriétés communes à un ensemble d'objets dans des prototypes de plus haut niveau.

1. Présentation de l'algorithme de restructuration

L'algorithme (Moore & Clement, 1996) fonctionne de la manière suivante : dans un premier temps, toute hiérarchie existante est « mise à plat ». Plus précisément, les propriétés contenues dans des prototypes de haut niveau se voient attribuées à tous les fils, jusqu'à ce qu'il n'y ait plus que des prototypes de niveau égal. Ensuite, l'algorithme crée de nouveaux prototypes abstraits qui contiennent toute les propriétés partagées par plusieurs éléments, jusqu'à obtenir une nouvelle hiérarchie optimisée, sans doublon de propriétés (Figure 79 et Figure 80).

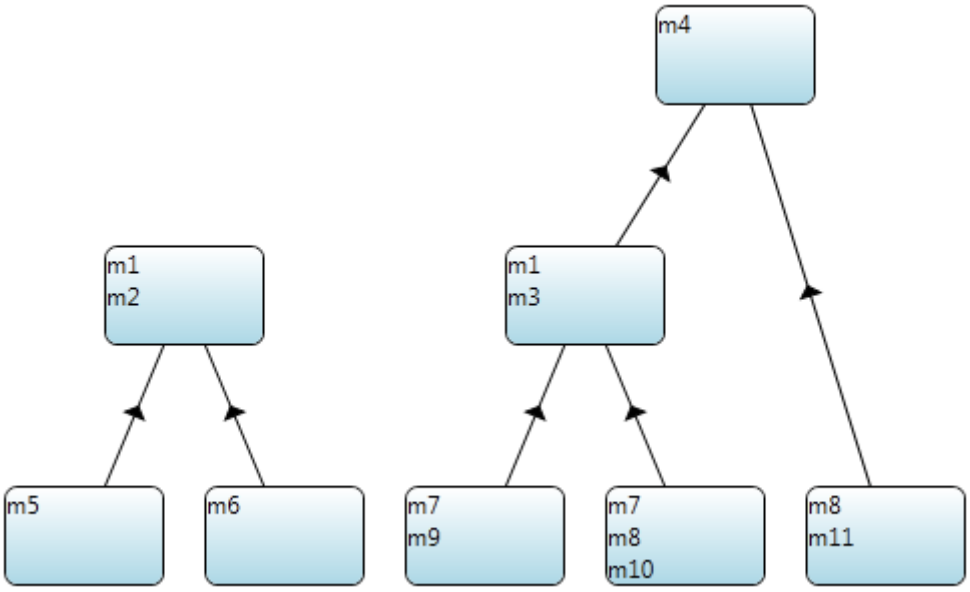


Figure 79.Nous avons implémenté l'algorithme de restructuration de Moore dans notre éditeur. Les cases représentent un prototype, et 'm' un ensemble de propriétés et de valeurs. Cette arborescence contient des doublons de propriétés.

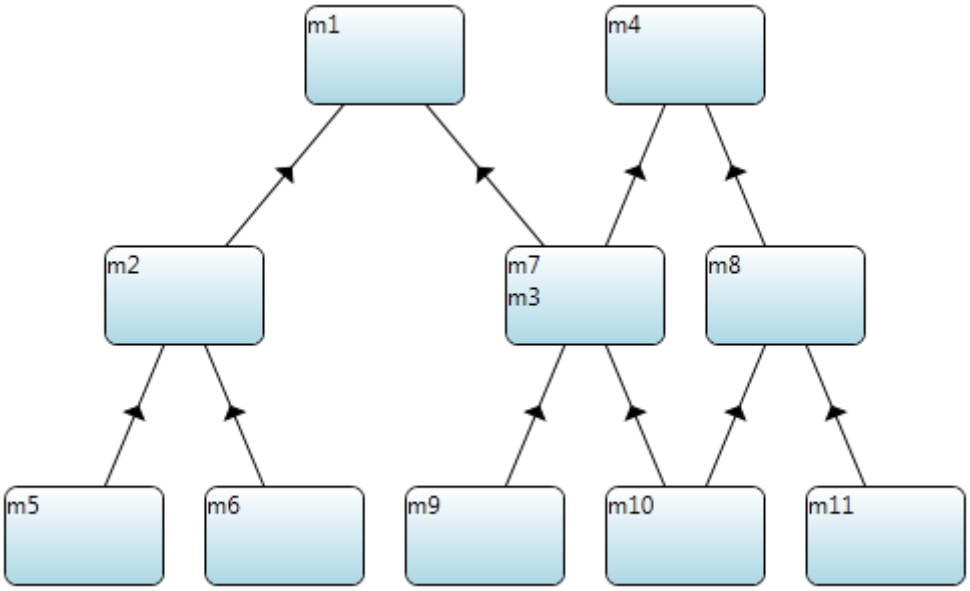


Figure 80. L'arborescence de la Figure 79 après restructuration.

2. Le graphe

Dans l'éditeur, l'outil (que l'on nomme IHR pour « Inheritance Hierarchy Restructuring ») se présente sous la forme d'une fenêtre externalisée contenant l'arborescence de prototypes. A l'origine, celle-ci est vide. A chaque fois que l'utilisateur crée un nouvel objet dans la scène, un nouveau prototype est ajoutée au graphe, sans

aucun lien de parenté avec les autres prototypes (Figure 81). Sélectionner un objet de la scène sélectionne également son prototype et inversement (Figure 82).

L'utilisateur peut choisir de créer une structure automatique à partir d'une sélection d'objets de la scène. Pour ce faire, il sélectionne les objets en question et applique ensuite une commande de restructuration. L'algorithme explore alors les propriétés des objets observés, et construit une hiérarchie où les propriétés communes aux objets sont factorisées dans des prototypes abstraits (c'est-à-dire des prototypes qui ne sont pas représentés dans la scène) (Figure 83). Le graphe ainsi obtenu est un *graphe orienté acyclique*, il ne s'agit ni d'un *arbre* (car un nœud peut avoir plusieurs parents), ni d'un *multitree* (puisque plusieurs parents d'un nœud peuvent avoir un même parent) (Furnas & Zacks, 1994). Nos préoccupations étaient de trouver comment lier la scène au graphe afin que l'utilisateur puisse comprendre la correspondance entre ses objets et le graphe (R1.4 *identifier les ensembles*), et de définir les interactions.

a) Modifications

L'utilisateur peut modifier une propriété déléguée pour modifier tous les objets concernés en une seule fois. Il peut, pour cela, choisir d'agir directement sur l'arbre, en déposant une valeur de propriété du panel d'échantillons sur un des nœuds (Figure 84) ou bien modifier directement n'importe quel objet de la scène en utilisant conjointement une commande « set for all » (Figure 85), ce qui a pour effet de modifier tous les autres objets ayant la propriété déléguée (R2.3 *portée de l'action*). La commande « set for all » s'active en cochant la case à cocher correspondante dans l'interface. L'intérêt de cette opération est de permettre d'interagir uniquement sur les objets de la scène et de rendre la visualisation de l'arbre optionnelle. Après refactorisation, toutes les propriétés identiques partagées par plusieurs objets peuvent donc être changées en ne modifiant qu'une seule occurrence d'un de ces objets dans la scène. L'utilisation de la commande « set for all » reste optionnelle : si l'utilisateur le désire, il peut préférer modifier individuellement un objet sans que ce changement ne se propage à d'autres. Ainsi, la valeur est « surchargée » : elle dépend toujours du même prototype, mais sa valeur est différente.

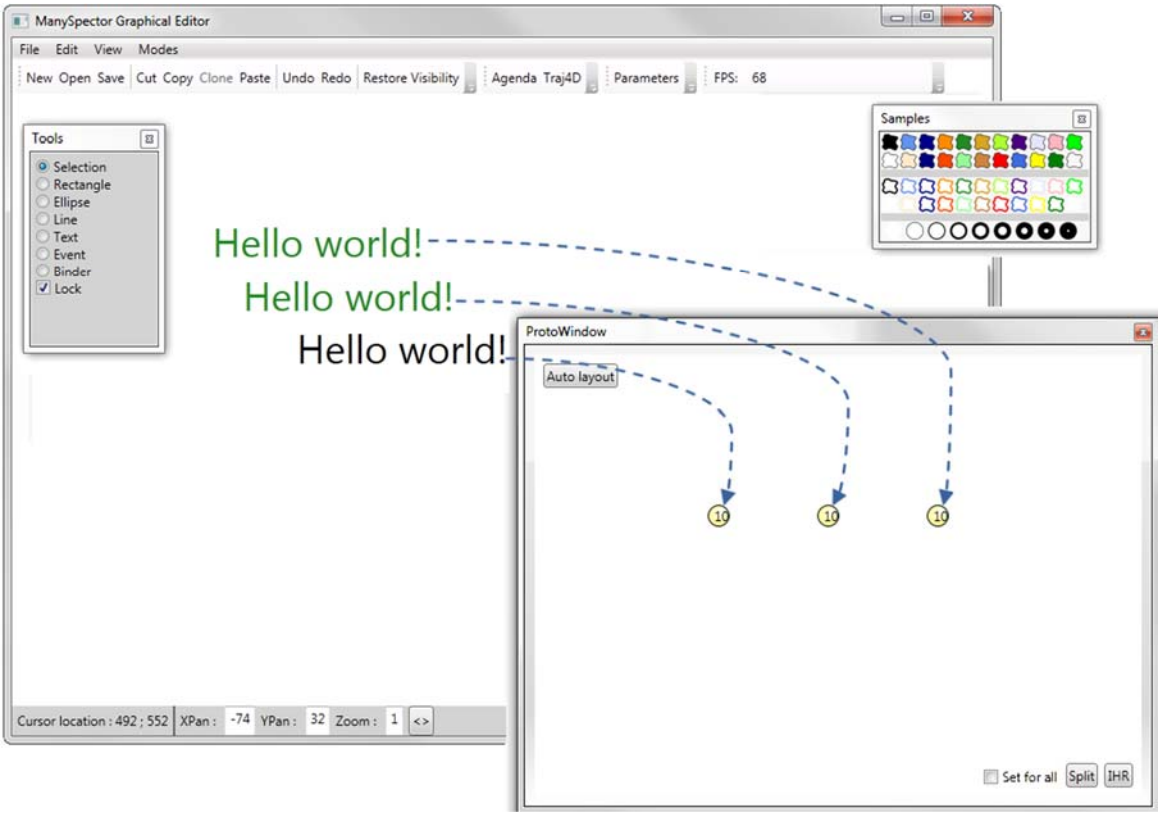


Figure 81. Trois objets de type texte ont été ajouté dans la scène. La fenêtre à droite contient trois cercles qui représentent les prototypes correspondant à chacun des objets.

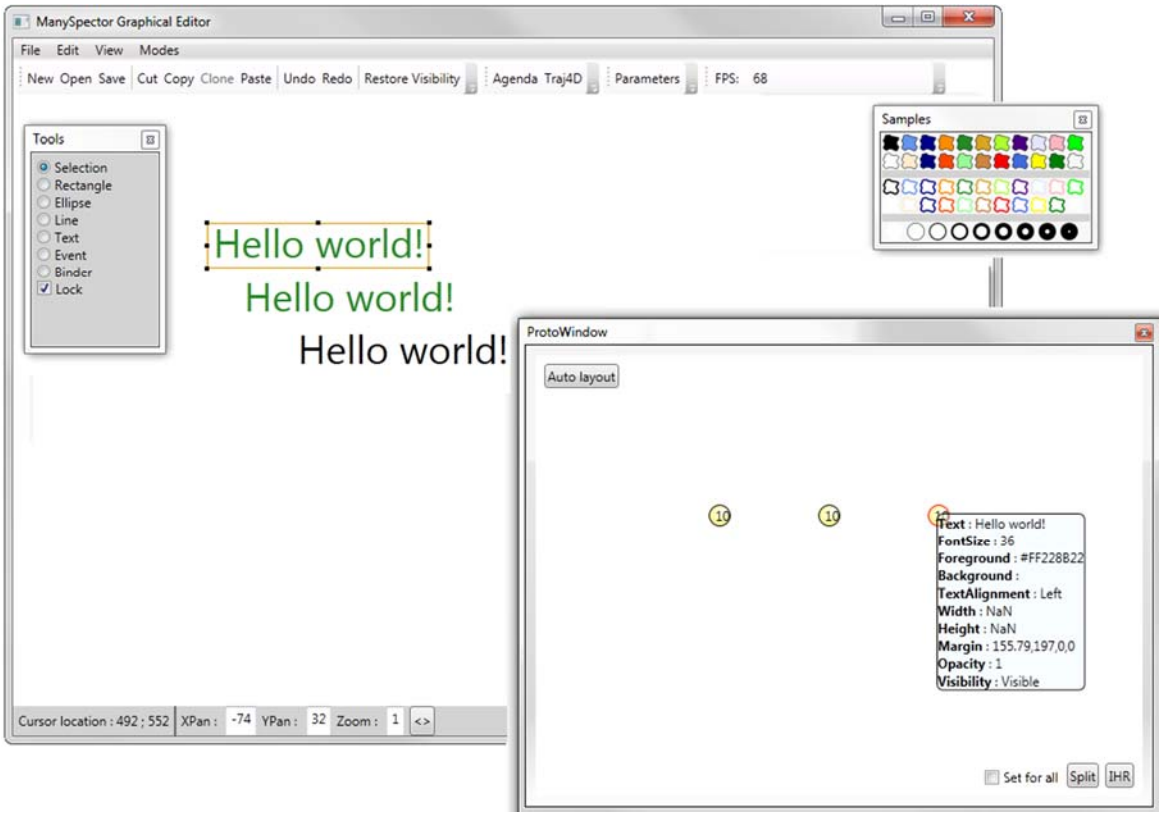


Figure 82. Sélectionner un objet de la scène sélectionne également son prototype, ce qui a pour effet d'afficher ses propriétés.

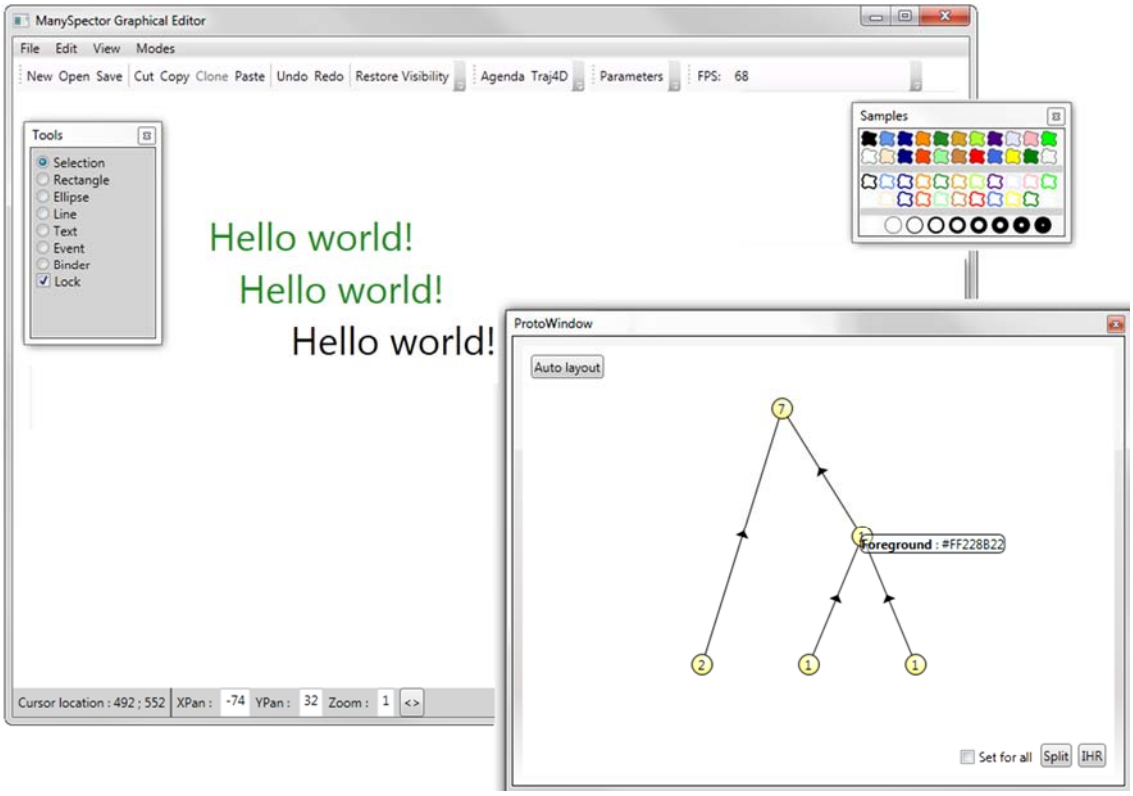


Figure 83. La restructuration a été appliquée aux trois objets et leurs prototypes, créant l'arborescence suivante : les propriétés de tailles communes aux trois objets sont abstraites au plus haut niveau. La couleur de texte verte est abstraite pour les deux objets de cette couleur.

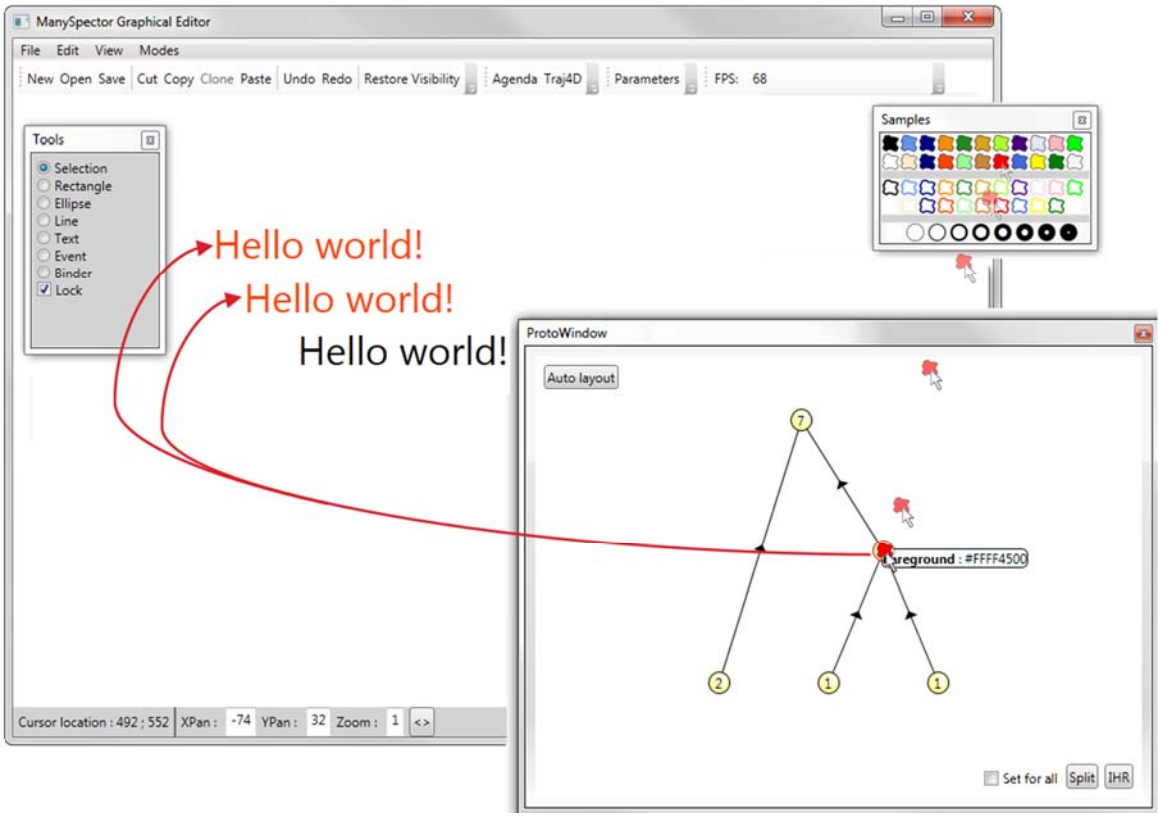


Figure 84. Déposer l'échantillon « couleur de fond rouge » sur le prototype qui déléguait la couleur verte modifie les deux objets verts.

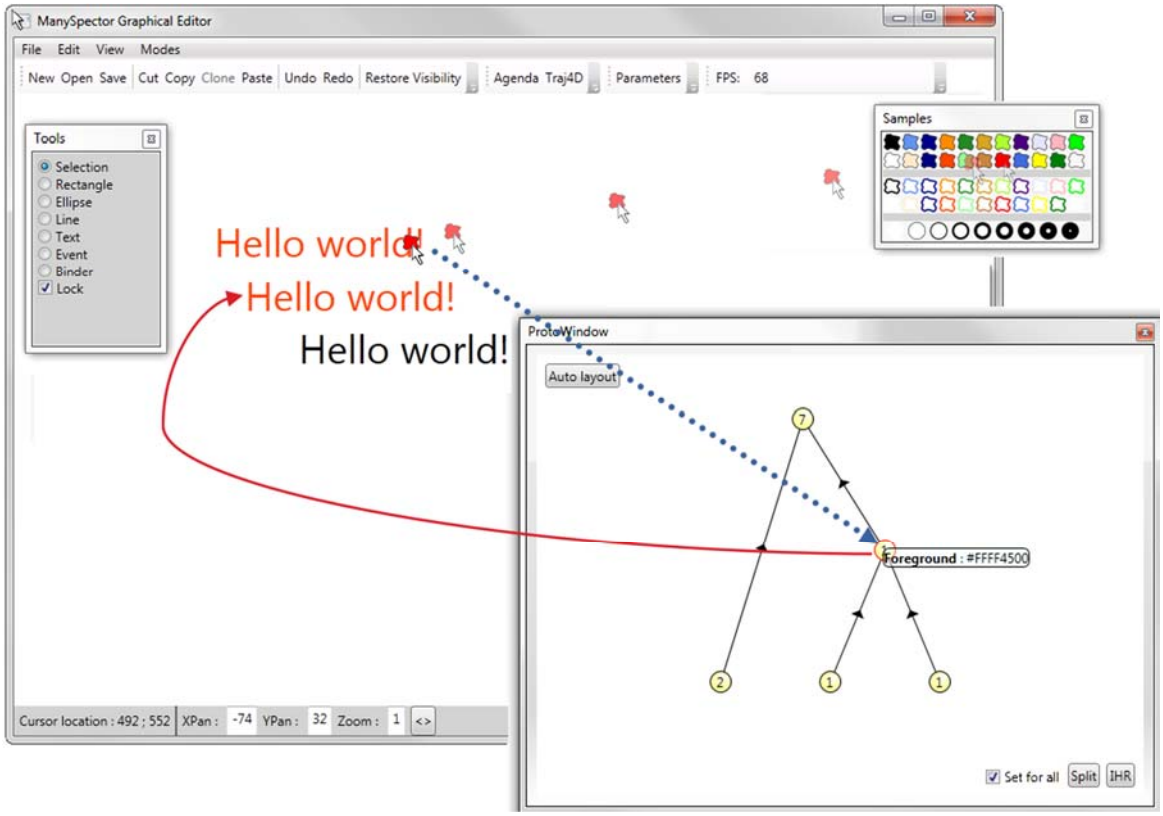


Figure 85. Modifier l'un des deux objets vert, en ayant activé la commande « set for all » modifie le prototype (flèche bleue) ce qui propage le changement sur le second objet vert (flèche rouge).

b) Densité du graphe

Il est pertinent d'avoir dans la vue contenant l'arborescence la représentation d'un prototype correspondant pour chaque objet puisque tout objet graphique est considéré comme un prototype. Cependant, la notation peut devenir complexe dès lors que la scène contenait un nombre conséquent d'objets : la vue est surchargée, obligeant l'utilisateur à scroller horizontalement pour voir l'arbre dans son ensemble (Figure 86).

Pour remédier à ce problème, nous avons fait le choix de ne pas faire apparaitre dans la vue ces prototypes. En effet, l'information qu'ils apportent est limitée puisqu'ils ne servent qu'à faire la correspondance entre un objet de la scène et un prototype, (nous présentons une alternative pour effectuer cette correspondance plus loin). Il n'y a donc plus de feuilles mais que des nœuds « parents », ainsi, la vue est plus lisible.

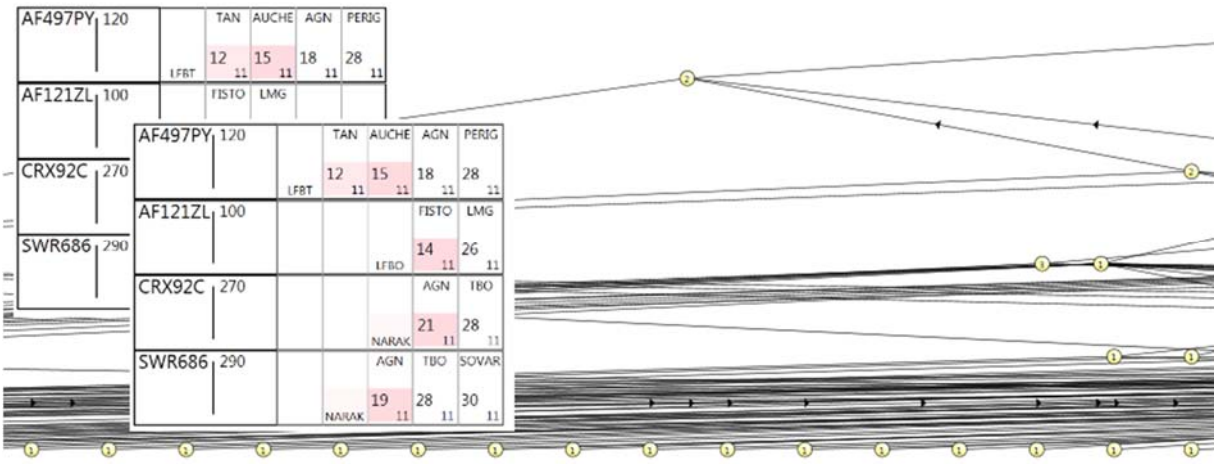


Figure 86. La scène contient un ensemble de strips du contrôle aérien (à gauche). Le nombre d'objets graphiques est important, de fait le graphe de prototypes est surchargé et difficilement lisible, requérant un défilement horizontal de la scène pour voir tous les prototypes.

c) La visualisation

Afin d'augmenter l'information offerte par la vue de l'arborescence, nous avons modifié la représentation des prototypes pour qu'ils puissent exposer visuellement certaines propriétés telles que les couleurs de fond et de contour (Figure 87). Ainsi, un prototype qui détient la propriété couleur de fond rouge, sera peint en rouge dans l'arborescence. Les propriétés couleur et épaisseur de contour sont également représentées.

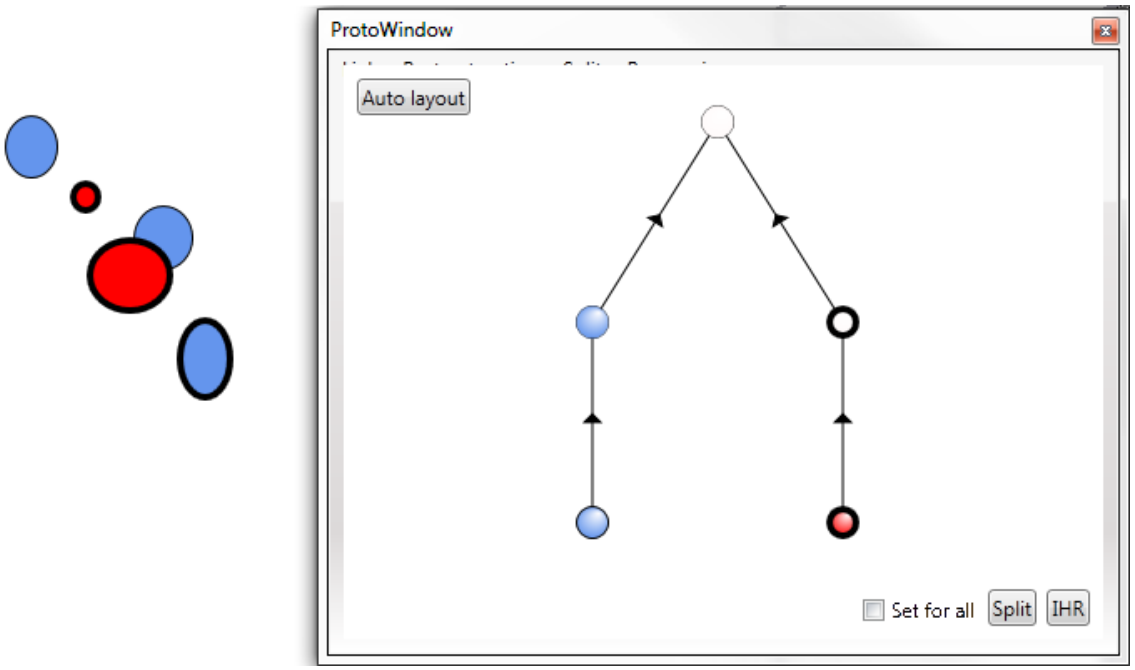


Figure 87. La représentation des prototypes a été améliorée afin de permettre de visualiser certaines propriétés telles que la couleur de fond, de contour, et l'épaisseur de contour.

En s'appuyant sur une vue externalisée pour présenter une structure des propriétés des objets de la scène, on ajoute une indirection spatiale importante (Beaudouin-Lafon, 2000). Il peut être compliqué de savoir de quels prototypes de l'arbre dépendent les objets, et inversement, de savoir quels objets dépendent d'un prototype donné (R1.4 *identifier les ensembles*). De façon similaire aux interactions décrites pour ManySpector lors du survol des valeurs, nous avons implémenté un mécanisme qui permet de mettre en évidence la correspondance entre objets et prototypes. En survolant un prototype, tous les objets de la scène qui n'en dépendent pas sont mis en retrait, afin de dévoiler ceux qui en dépendent (Figure 88). En survolant un objet de la scène, les prototypes qui lui délèguent une ou plusieurs propriétés et les liens de parentés entre ces prototypes sont mis en évidence par un halo rouge (Figure 89). Ainsi, nous conservons la propriété de correspondance entre les objets de la scène et les prototypes du graphe sans qu'ils ne soient représentés dans l'arborescence.

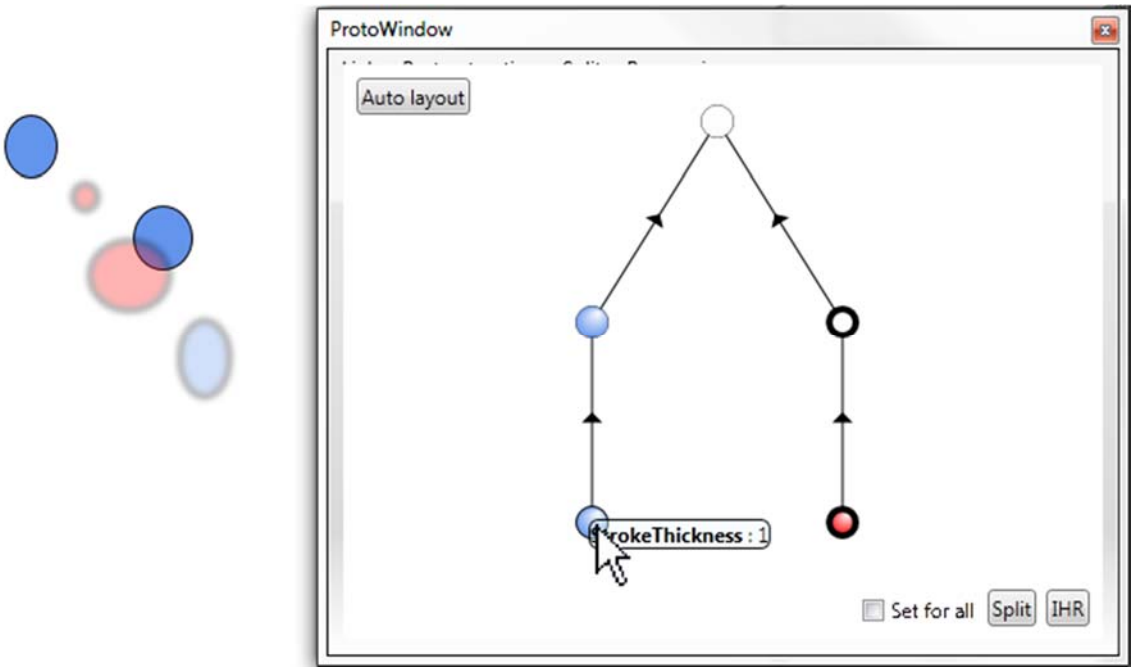


Figure 88. Survoler un prototype a pour effet de révéler dans la scène les objets qui ont des propriétés déléguées par le prototype. Ils sont mis en évidence en estompant tous les autres objets.

a) Modification de la structure

La génération automatique d'une structure peut être limitée du fait qu'elle est entièrement construite par le système, et ainsi ne pas correspondre aux attentes de l'utilisateur. Par exemple, l'utilisateur peut vouloir construire lui-même des prototypes et certains liens de parenté parce qu'il aimerait exposer des valeurs de propriétés qui ne sont pas présentes au sein des objets de la scène. Il peut alors s'avérer frustrant de ne pas pouvoir modifier la structure. L'utilisateur peut donc créer des nœuds et des

liens de parenté par manipulation directe. En faisant un glisser-déposer d'une valeur de propriété (issue du panel de l'échantillon ou de ManySpector) dans un espace vide de la vue, l'utilisateur crée un nouveau prototype contenant cette valeur de propriété. Il devient alors possible de déléguer explicitement à des objets de la scène des prototypes crées manuellement (R1.3 *modifier les ensembles*) (Figure 90).

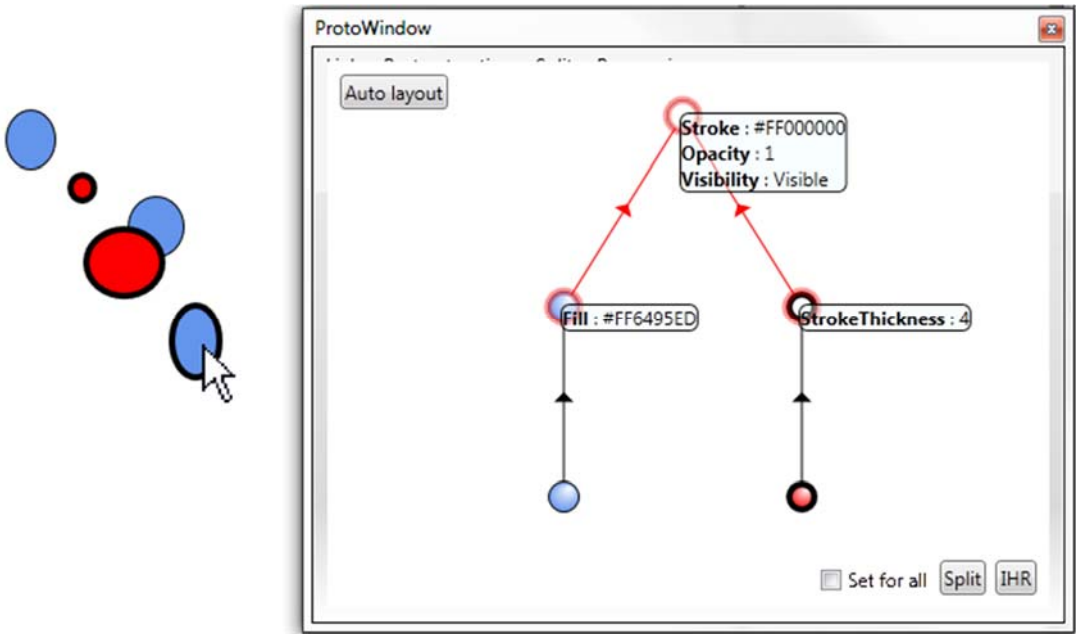


Figure 89. En survolant un objet de la scène, tous les prototypes qui lui délèguent des propriétés sont mis évidence par un halo de couleur rouge.

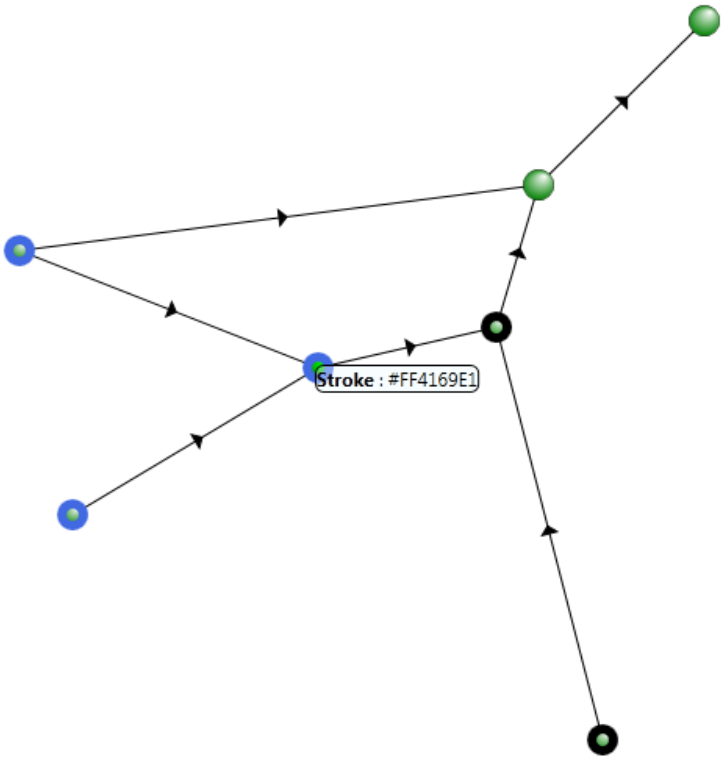


Figure 90. Exemple de graphe de délégation de propriétés créé par l'utilisateur.

3. Conclusion sur la structuration automatique

Le principal intérêt de la structuration automatique est de permettre de créer une structure a posteriori, et ainsi, de ne pas limiter la conception exploratoire. L'on retrouve les avantages d'une structure explicite, à savoir la persistance de celle-ci et la capacité à définir un modèle qui peut donner du sens aux objets. L'idée d'utiliser une vue externalisée pour créer un graphe de délégation augmente le pouvoir d'action grâce à la possibilité de déléguer des propriétés de prototypes abstraits (qui n'existent pas physiquement dans la scène). Ce concept est similaire à celui des styles, comme par exemple le langage CSS, où des valeurs de propriétés sont définies hiérarchiquement.

À la différence de la création automatique, la réalisation d'une feuille de style CSS demande d'écrire du code afin d'attribuer des valeurs de propriétés aux objets graphiques associés à la feuille de style. Un style dans un éditeur de texte tel que Word se construit également en sélectionnant dans une fenêtre modale des ensembles de valeurs de propriétés par catégorie de texte (titre, accentuation, etc.). L'utilisateur doit interagir avec le style afin d'avoir du pouvoir d'action sur ses éléments. A l'inverse, la création automatique permet d'interagir directement sur les éléments d'intérêts et de générer un style à partir de leurs propriétés.

La restructuration automatique peut donc se voir comme un moyen de générer automatiquement des styles basés sur les propriétés des objets de la scène. Il devient donc possible de modifier plusieurs objets en interagissant directement sur le style, ou bien sur n'importe quel objet de la scène en demandant une propagation des modifications pour tous les objets ayant le même style.

En l'état, ces interactions ont été jugées trop complexes pour être exploitées efficacement et être sujettes à des expérimentations. L'indirection spatiale est trop importante malgré les solutions de mise en évidences des correspondances entre objets de la scène et les prototypes du graphe (figure). Par ailleurs, la complexité de la visualisation induit une absence de passage à l'échelle, le graphe n'étant pas exploitable dès que le nombre de prototypes et de liens devient important.

Nous pensons que le concept de création automatique peut être utile et peut faciliter la construction de structures explicites, mais cela demande des investigations plus conséquentes dans la visualisation de telles structures et plus de consistance entre les interactions sur les objets d'intérêts qui sont le point d'attention majeur de l'utilisateur et une hiérarchie d'objets virtuels externalisée. Des techniques de visualisation pourraient apporter une meilleure compréhension du graphe, comme le « bundling »

(Hurter et al., 2013), où l'on agrège des liens proches pour faciliter la lecture. Une solution pourrait être de s'affranchir d'un graphe externe et de nœuds abstraits, afin que toutes les dépendances soient générées directement entre les objets de la scène comme présenté dans le chapitre sur la structuration explicite (p. 82). Mais comment choisir dans ce cas quel objet déléguerait ses propriétés à d'autres, et comment l'utilisateur peut-il en avoir conscience ? Augmenter les interactions possibles avec les représentations des prototypes est une piste intéressante : en permettant la manipulation directe de ces représentations à la façon d'un « surrogate » (Kwon et al., 2011), il devient possible d'étendre les manipulations à un ensemble d'objets (R2.3 *portée de l'action*).

Se pose ensuite la question de la « remise à plat » du graphe par l'algorithme de structuration. Ne risque-t-il pas d'y avoir des différences trop importantes entre une hiérarchie de départ et la nouvelle qui empêcherait l'utilisateur de comprendre son évolution ? Est-ce alors un problème de ne pas disposer d'un algorithme incrémental ? Dans le cas où l'on utilise la restructuration dans l'unique but d'abstraire des propriétés communes et les modifier, l'utilité du graphe est moindre si l'on ne procède qu'à des manipulations d'objets de la scène. En revanche, si l'utilisateur modifie lui-même sa structure, une restructuration automatique ayant lieu ensuite pourrait être pénalisante puisqu'elle détruirait celle établie manuellement.

F. Une autre structure implicite : l'historique des actions de l'utilisateur

La prise en charge d'historiques d'actions est indispensable à tout système interactif (Shneiderman, 1987). Cela permet à l'utilisateur de corriger ses erreurs et, par conséquent d'explorer des solutions sans crainte du faux pas. Cependant, les historiques sont très généralement conçus de façon linéaire : ils fonctionnent telle une pile, dans laquelle les commandes réalisées par l'utilisateur sont ajoutées les unes après les autres. Cette implémentation à des limites : elle oblige l'utilisateur à parcourir les actions dans l'ordre inverse de leur réalisation. Plus précisément, pour annuler une action ayant eu lieu à un moment précis, l'utilisateur doit auparavant annuler tout celles ayant été réalisées après elle. Des travaux ont été menés autour des historiques non linéaires (Vitter, 1984)(Scull et al., 2009) pour mieux supporter le processus créatif. Nous nous sommes intéressés à l'historique en tant que service favorisant la conception exploratoire, mais aussi en tant que structure. Nous avons donc implémenté dans l'éditeur graphique un historique linéaire et régional, et avons exploré des prototypes d'interactions qui peuvent améliorer l'exploration et la

comparaison de solutions, la perception des changements, et la réutilisation des actions de l'utilisateur.

1. L'historique d'action

Nous nous sommes intéressés aux historiques d'actions car il s'agit d'un service utile pour l'exploration mais surtout parce qu'il s'agit en soi d'une structure : constitués de l'ensemble des actions réalisées par l'utilisateur, les historiques sont des regroupements d'éléments organisés séquentiellement. En tant que structure, il est important qu'ils puissent être supportés par des services qui satisfont les exigences que nous avons définies. Cette structure n'est pas construite par l'utilisateur, elle est définie par les actions qu'il réalise dans son activité de conception exploratoire, il s'agit donc d'une structure implicite.

Les historiques satisfont peu les exigences définies. Ils sont généralement masqués à l'utilisateur dans les éditeurs graphiques. Le seul moyen de savoir s'ils contiennent des actions annulables ou réutilisables est donné par le bouton représenté généralement par une flèche qui n'est plus grisé. En étant masqué, l'historique ne permet pas d'identifier quelles sont les actions qui le constituent (R1.4 *identifier les ensembles*), de rechercher ou de désigner une action particulière (R1.1 *rechercher,* R1.2 *désigner*), on ne peut accéder qu'à celle en tête de pile. En tant que structure implicite, on ne modifie pas volontairement l'ensemble (R1.3), c'est l'utilisation des commandes *annuler* et *refaire* ainsi que les actions réalisées dans l'espace de travail qui modifie la structure.

Certains outils proposent cependant une visualisation simple de celui-ci (Figure 91). L'utilisateur peut alors être informé des actions passées et il peut rapidement revenir à un état précédent de sa conception en sélectionnant la dernière opération à restaurer, sans avoir à annuler toutes les actions suivantes une par une. Bien que cela augmente la capacité de désignation, cela ne permet toujours pas de désigner une action particulière sans avoir à annuler toutes les précédentes.

Par ailleurs, la modification de la structure n'est pas perceptible en raison des changements abrupts survenant lors de l'annulation d'une opération (R2.4 *percevoir les conséquences*). Le support à la conception exploratoire (R3) est limité en raison de la linéarité des historiques : il n'est pas possible de faire des comparaisons autres que temporelles, et seulement pour deux états juxtaposés.

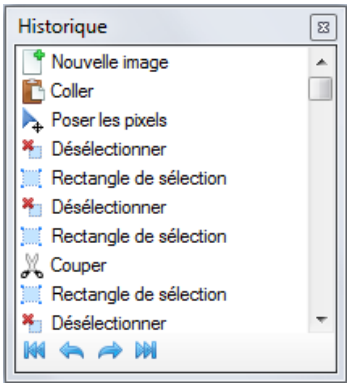


Figure 91. L'historique de l'outil de dessin Paint.NET. Les actions sont visibles et sélectionnables.

2. Transitions d'états animées

L'utilisation des commandes *annuler* et *refaire* permettent de revenir à l'état précédant la réalisation d'une opération ou de revenir à des états suivants. Cette fonctionnalité peut être utile pour comparer deux états juxtaposés d'une conception, en répétant successivement les commandes *annuler* et *refaire*, l'utilisateur peut faire apparaitre les différences entre les deux états de la scène et l'évaluer (Terry & Mynatt, 2002a). Ce mode de comparaison a pour inconvénient d'être temporel et non spatial, l'utilisateur doit mémoriser les différences entre deux états pour pouvoir les comparer. Qui plus est, le changement est abrupt : il peut être difficile de visualiser les changements si la scène est dense (par exemple, dans une scène contenant un nombre important de trajectoires d'avions), si le changement diffère très peu ou s'il a lieu hors de l'espace visible. Les transitions d'états animées sont des moyens pouvant permettre aux utilisateurs de mieux percevoir le changement (R2.4)(Schlienger, Conversy, Chatty, Anquetil, & Mertz, 2007). Par exemple, Gliimpse (Dragicevic, Huot, & Chevalier, 2011) est un outil qui exploite les transitions animées dans un contexte d'édition textuelle afin d'aider à percevoir les modifications de texte entre des vues différentes. En modifiant du texte dans une vue d'édition différente de la vue de rendu (exemple : une page html, du code LaTex, une page de wiki), il n'est pas aisé de repérer ce qui a été modifié lors du basculement dans l'autre vue. Gliimpse permet, grâce à la transition animée, de suivre des éléments textuels d'une vue à l'autre, et ainsi de ne pas perdre le focus sur des éléments d'intérêts (Figure 92).

Figure 92. Transition d'un texte depuis la vue de rendu vers la vue d'édition (Dragicevic et al., 2011, fig. 1).

Les deux préoccupations que nous avions concernant la perception des changements étaient de pouvoir retrouver les objets concernés et de pouvoir percevoir les différences. Nous avons alors implémenté dans l'éditeur graphique un mécanisme de transition animées afin d'éviter les changements abrupts entre deux états d'une scène : tout changement de propriété est animé à chaque utilisation des commandes annuler et refaire. Par exemple, en cas d'une annulation d'un changement de position d'un objet, celui-ci se voit déplacé de sa position actuelle à sa position précédente (Figure 93). Ceci permet à l'utilisateur de suivre l'objet, et ainsi de savoir où se trouve son ancienne position, ce qui peut s'avérer difficile si le déplacement se fait instantanément ou s'il est hors du champ de vision. La Figure 94 montre l'état d'une scène comportant un grand nombre de trajectoires générées aléatoirement, avant et après annulation d'une opération de translation partielle d'une trajectoire. Un changement abrupt rend très difficile la perception des différences entre les deux scènes. Grâce à l'animation, l'utilisateur peut suivre les points modifiés jusqu'à leur position d'origine sans les perdre de vue.

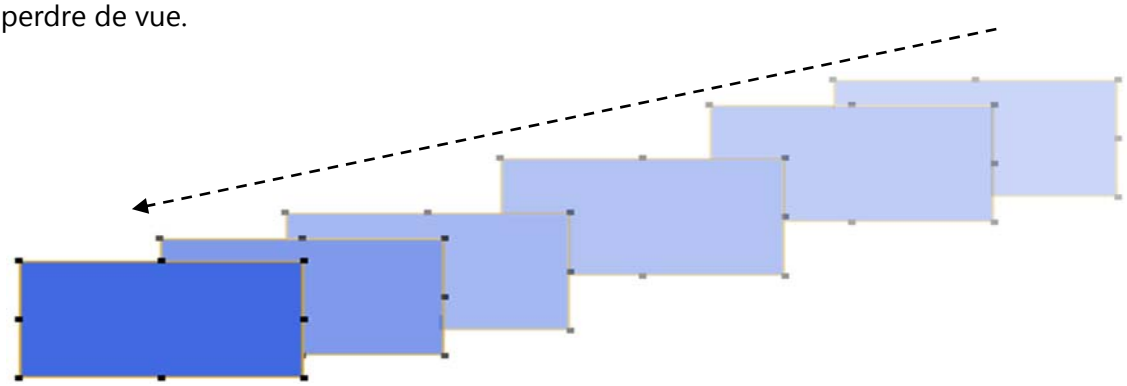


Figure 93. Annulation d'une opération de translation d'un rectangle. Une transition animée permet de suivre son déplacement de sa position actuelle à sa position d'origine.

Dans le cas où l'objet modifié n'est pas présent dans l'espace visible de travail (suite à une opération de défilement par exemple), animer la modification n'est pas suffisant. Une autre alternative pour percevoir ces changements serait d'utiliser des signaux visuels. Des travaux de recherche ont été menés autour des alarmes visuelles dans les interfaces graphiques. Bien qu'appliqués plus particulièrement au domaine du contrôle aérien, Chatty, Bustico, & Athènes, (1999) présentent des signaux animés afin d'attirer l'attention de l'utilisateur sur des évènements survenant dans l'interface. (Baudisch, (2003) présente également des travaux utilisant les alarmes visuels dans des petites surfaces (smartphones). L'utilisation de tels signaux (flash en périphérie, cercles concentriques rétrécissants) permettent d'informer l'utilisateur de modifications ayant eu lieu hors de la scène visible et ainsi de trouver les objets qui ont été modifiés.

Les déplacements ne sont pas les seules propriétés animées. En fait, tout changement de propriété visuelle est sujet à une transition animée. Un changement d'épaisseur de contour, ou un redimensionnement dévoilera progressivement la nouvelle taille ou épaisseur. Un changement de couleur de fond ne sera pas non plus instantané : l'objet prendra progressivement la nouvelle couleur selon une interpolation entre celle d'origine et celle à atteindre. Sur un objet de type évènement d'agenda, changer la durée fera augmenter (ou réduire) sa longueur puisque cette propriété est visuellement représentée par la longueur de l'objet. Ainsi, l'utilisateur peut se rendre compte des différences sur l'objet avant et après modification.

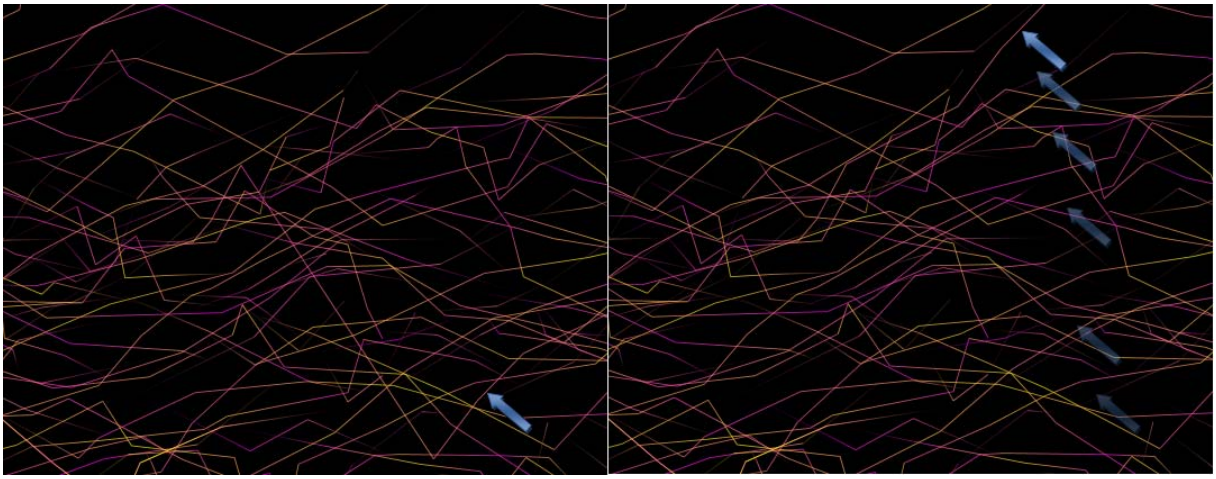


Figure 94. La scène contient un entrelacement de trajectoires. L'utilisateur annule la dernière action : une translation de la trajectoire marquée par une flèche dans la vue de gauche. La vue de droite décrit la transition qui permet de repérer la trajectoire lorsqu'elle revient à sa position d'origine.

Les animations suivent le principe du « slow-in/slow-out » (Thomas & Johnston, 1981) : elles sont lentes au début, accélèrent progressivement et ralentissent à la fin. Cela rend l'animation plus réaliste et agréable à observer qu'une animation à vitesse constante (Kochanek & Bartels, 1984). Les animations sur les changements de propriétés peuvent améliorer la compréhension du système et faciliter le suivi des objets d'intérêts.

3. Histoglass : l'exploration archéologique

Pour pallier les limites de la linéarité des historiques, nous avons prototypé un outil « d'exploration archéologique » appelé Histoglass, qui permet d'observer localement les différents états d'une partie de la scène. L'outil se présente sous la forme d'une « toolglass » (Bier et al., 1993), une palette transparente et déplaçable qui donne de l'information en survolant des éléments de la scène.

Le scénario d'utilisation est le suivant : l'utilisateur souhaite annuler un changement ayant eu lieu sur un objet en particulier, cependant, compte tenu des contraintes dues

à la linéarité de l'historique, il va être obligé d'annuler des opérations réalisées sur d'autres objets qu'il souhaite pourtant conserver. Grâce à l'histoglass, il peut survoler le ou les objets d'intérêts et à l'aide de la molette annuler uniquement les opérations qui les concernent (Figure 95, Figure 96 et Figure 97).

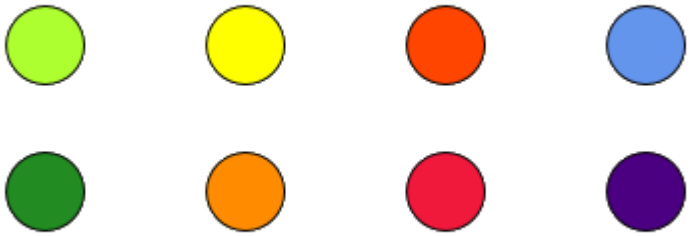


Figure 95. Une palette de couleur créée par l'utilisateur.

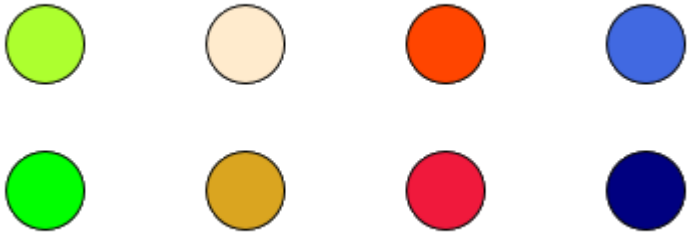


Figure 96. L'utilisateur explore d'autres ensembles de couleurs pour sa palette qu'il ajuste.

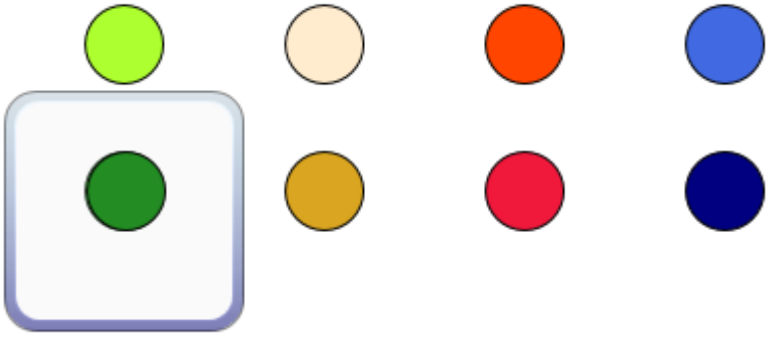


Figure 97. L'utilisateur souhaite retrouver la couleur qu'il avait choisi à l'origine pour le cercle vert en bas à gauche. Il est compliqué de retrouver exactement la même couleur (en spécifiant les valeurs exactes 'R','G' et 'B'). Il préfère annuler ses actions pour retrouver la couleur, cependant, un historique linéaire l'obligerait à annuler d'autres couleurs qu'il a choisis. Il procède alors en amenant l'histoglass vers le cercle et restore l'ancienne couleur.

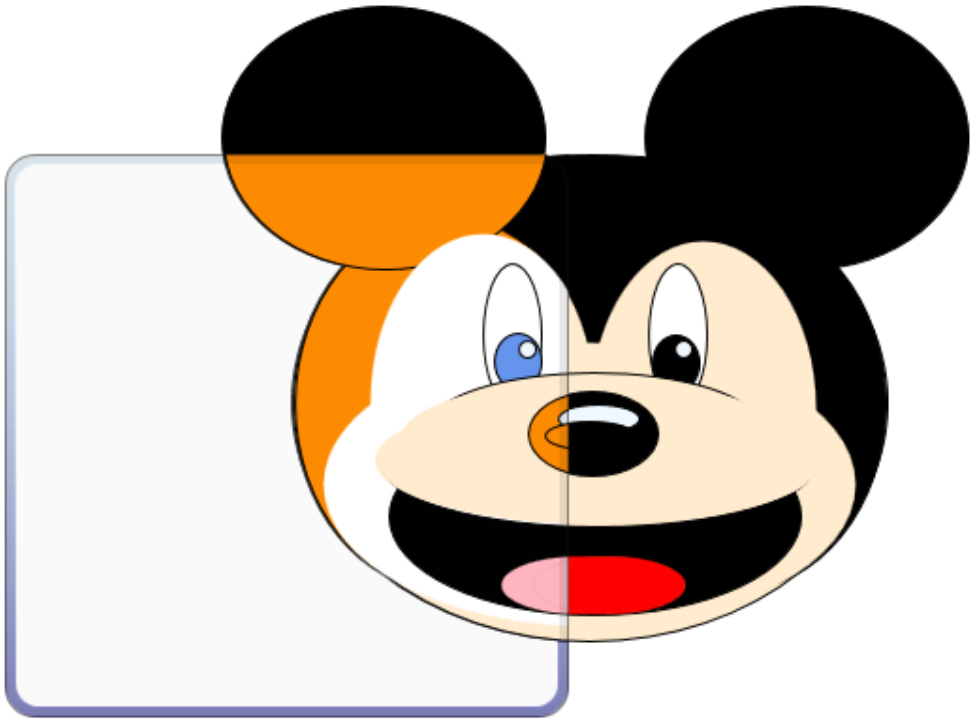


Figure 98. L'histoglass survole une partie de la scène et révèle des états précédents des objets, ce qui permet de les comparer avec les objets de l'état courant en ne survolant qu'un fragment des objets.

Par ailleurs, l'histoglass permet de faire des comparaisons côte-à-côte de deux états d'un objet. En survolant un fragment d'un objet d'intérêt et en annulant une action, seul le fragment de l'objet survolé dévoilera son état précédent, ce qui permet de comparer directement cet état avec celui actuel (Figure 98). Des comparaisons spatiales côte-à-côte permettent plus facilement à l'utilisateur d'évaluer le résultat de ses actions en sollicitant la mémoire à court et favorisent ainsi le processus créatif (Terry & Mynatt, 2002a). L'histoglass présente l'avantage d'être rapide à manipuler pour comparer les états d'un même objet et annuler les actions localement : les seules actions requises sont le déplacement de celle-ci, et le déroulement des actions sur l'objet à l'aide de la molette. Il n'est donc pas nécessaire de rechercher les actions concernant l'objet dans l'historique pour les annuler (R1.1 *rechercher* et R1.2 *désigner*). L'histoglass est alors est un moyen permettant la réutilisation d'états passés dans l'état présent et de comparer ou restaurer des états.

4. Réutilisation des commandes

Nous avons souhaité exploiter la visualisation des commandes afin de proposer des techniques d'interactions qui permettent de les réutiliser. Non seulement l'utilisateur peut savoir quelles ont été ses précédentes actions (R1.4 *identifier*), mais il peut également les réappliquer sur d'autres objets d'intérêt. En sélectionnant une

commande dans l'historique (R1.2 *désigner*), il peut grâce à une opération de glisser et déposer, réutiliser le résultat d'une commande sur un autre objet (R2.2 *paramètre de l'action*) (Figure 99). Ces interactions sont cohérentes avec l'utilisation des panels d'échantillons et ManySpector. Par exemple, en faisant un glisser-déposer de la commande d'ajout d'objet dans un espace vide de la scène, l'utilisateur créer un nouvel objet à cet endroit, dont les propriétés sont les mêmes que celles de l'objet crée par la commande. Sélectionner plusieurs commandes peut être un moyen de créer une composition de commandes afin de créer des séquences d'actions personnalisées telle une « macro » réutilisable.

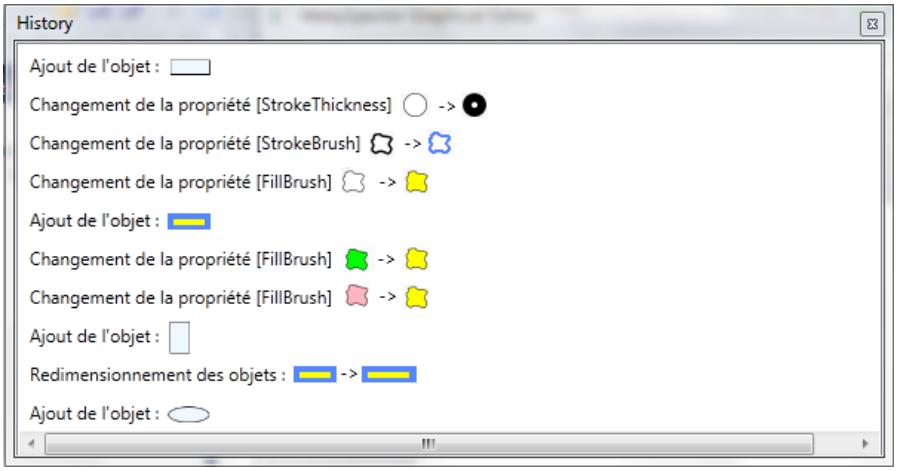


Figure 99. Historique d'actions interactif. L'utilisateur peut sélectionner une action et la réutiliser en la déposant dans la scène ou bien sur un objet de la scène.

5. Réutiliser les actions pour augmenter le pouvoir d'action

La création de commandes reprend certains principes de la programmation par démonstration. Nous souhaitons aider l'utilisateur à exploiter par des services interactifs les actions précédemment réalisées pour construire ses propres instruments et ainsi éviter la répétition de tâches. La création de commande n'est pas nouvelle, de nombreux outils permettent de créer des macros (macros Excel, création de scripts, macros AutoCAD etc.). En revanche le moyen pour y parvenir diffère. Nous ne sommes pas dans une optique de programmation, où de telles opérations sont possibles en écrivant du code, mais dans une optique d'instrumentation : l'utilisateur utilise les moyens interactifs mis à disposition dans l'interface pour pouvoir réutiliser les opérations qu'il a déjà réalisées. Il ne s'agit pas non plus de programmation par l'exemple, bien que la finalité soit la même (créer des instruments personnalisés à partir des actions de l'utilisateur), la programmation par l'exemple repose sur l'utilisation d'un algorithme d'inférence qui va proposer des compositions d'opération en fonction des intentions de l'utilisateur et de ses actions précédentes.

Rendre l'historique visible peut aider à avoir un meilleur aperçu des différentes étapes d'une conception, mais cela peut également être utile pour construire des compositions d'actions réutilisables sur d'autres objets, et ainsi permettre de créer des structures réduisant le nombre d'opérations à répéter. Cette piste de recherche n'a pour le moment été que survolée mais nous pensons qu'il est possible d'offrir de nouveaux paradigmes d'interaction pour considérer une conception non seulement comme quelque chose de présent, mais plutôt comme un ensemble passé-présent et en tirer profit.

G. Conclusions sur les principes de structuration

Les outils interactifs que nous avons présentés ont permis l'émergence d'un concept, celui de l'interaction basée sur les structures, ou l'interaction structurelle. Deux catégories de structures ont été identifiées, les structures explicites, qui sont construites volontairement par l'utilisateur, et les structures implicites, qui sont révélées par les actions de l'utilisateur. L'interaction structurelle se veut être un moyen d'offrir un plus grand pouvoir d'action à l'utilisateur, en permettant d'interagir avec plusieurs éléments en peu d'actions tout en offrant les services interactifs permettant de construire ces structures, d'identifier leur contenu, et d'agir sur ce contenu ou bien sur les structures elles-mêmes.

Les structures peuvent être construites *a priori*, c'est-à-dire que l'utilisateur construit la structure pour pouvoir l'utiliser, ou *a posteriori*, lorsque l'utilisateur réalise des actions sans volonté d'établir une structure mais que le système permet d'extraire ultérieurement une structure à partir de ces actions. Nous avons donc catégorisé les outils présentés dans ce chapitre selon ces deux dimensions (Figure 100).

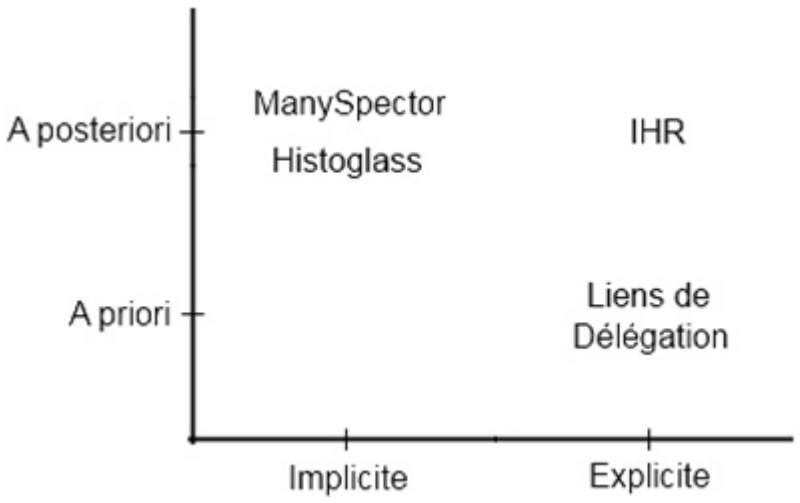


Figure 100. Catégorisation des outils.

Tableau 3. Respect des exigences de l'interaction structurelle par outil.

	Délégation	ManySpector	IHR	Histoglass
Rechercher (R1.1)	X	√	√	√
Désigner (R1.2)	X	√	√	√
Modifier (R1.3)	√	√	√	√
Identifier les ensembles (R1.4)	X	√	√	X
Spécifier la nature de l'action (R2.1)	√	√	√	X
Spécifier les paramètres de l'action (R2.2)	√	√	√	X
Spécifier la portée de l'action (R2.3)	√	√	√	√
Percevoir les conséquences des actions (R2.4)	√	√	√	√
Essayer (R3.1)	X	√	X	√
Évaluer (R3.2)	X	√	X	√
Explorer à court-terme (R3.3)	X	√	X	√
Comparer les versions (R3.4)	X	√	X	√
Structurer a posteriori (R3.5)[7]	X	√	√	√

Nous pouvons affirmer qu'une structure implicite l'est toujours a posteriori. En effet, puisqu'elle n'est pas construite volontairement par l'utilisateur mais révélée par les actions relatives à sa tâche de conception exploratoire, la structure implicite est toujours obtenue après toute intention d'établir une structure. Le tableau ci-dessus (Tableau 3) recense l'ensemble des exigences établies, et confirme leur respect pour chaque outil conçu.

[7] La délégation n'est pas a posteriori, en revanche il est possible de créer a posteriori des clones d'objets copiés.

ÉVALUATIONS

Nous avons soutenu dans les sections précédentes que nos outils sont nouveaux, cohérents et efficaces pour réaliser des interactions basées sur les structures. L'évaluation de ces revendications n'est pas une tâche simple. Nous étions particulièrement préoccupés par la clarté du concept des valeurs partagées, et le fait que celles-ci se réfèrent soit à une valeur soit à l'ensemble des objets qui partagent cette valeur : les utilisateurs allaient-ils comprendre et profiter de ce double statut ? S'il est courant de procéder à des évaluations de l'utilisabilité, nous étions plus intéressés par l'utilité. Nous souhaitions mettre en avant l'importance du problème et les bénéfices que pourraient apporter nos solutions.

L'évaluation des solutions était donc une étape cruciale afin de déterminer si elles étaient compréhensibles par les utilisateurs et utiles. Les solutions que nous souhaitions évaluer étaient celles qui permettaient d'utiliser les deux principes de structuration à partir des outils les plus aboutis : ManySpector pour la structuration implicite et la création de liens de délégation pour la structuration explicite. Pour cela cinq séances d'évaluations ont été menées auprès d'utilisateurs de différents domaines, d'une durée approximative d'une heure. Chaque évaluation était divisée en trois parties : un tutorial et deux scénarios de tests (interaction dans un ensemble « conséquent » d'objets et utilisation des techniques dans une application d'agenda), complétés par un questionnaire de cinq pages. Les questions auxquelles nous souhaitions répondre étaient les suivantes : est-il trop difficile pour les utilisateurs de saisir le concept de valeur partagée et celui des propriétés liées ? Quand bien même les utilisateurs parviendraient à comprendre ces concepts, ont-ils des difficultés à les utiliser afin d'interagir avec plusieurs objets ? Enfin, les utilisateurs peuvent-ils traduire des problèmes de haut niveau en interactions graphiques avec des valeurs partagées et les propriétés liées ?

A. Évaluation de l'utilité

Dans un article récent, Greenberg & Buxton ont discuté du rôle de l'évaluation dans la recherche en interaction (Greenberg & Buxton, 2008). En particulier, ils soulignent la différence entre *l'utilisabilité* et *l'utilité*. Selon Greenberg & Buxton, il est regrettable que des travaux de recherche soient évalués prématurément sur leur utilisabilité

risquant ainsi d'être rejetés, alors que ces travaux démontrent un potentiel d'utilité évidente.

Tableau 4. Grille d'évaluation d'Olsen.

Réduisent la viscosité	Flexibilité	ManySpector permet de faire des changements rapides sur l'ensemble de la solution qui peuvent être évalués. La délégation aussi, mais cela demande une conception au préalable (coût d'abstraction).
	Levier d'expression	Les valeurs partagées de ManySpector aident à réduire le nombre de choix à faire pour obtenir une solution.
	Correspondance	Les valeurs de propriétés sont représentées selon leur type pour une meilleure correspondance. Par exemple la propriété heure est représentée par une horloge miniature, la propriété couleur par un rectangle de couleur etc.
Peut amener de nouveaux participants		Ce critère n'est pas vraiment applicable, dans le sens où les solutions sont utiles pour des utilisateurs de profils variés pour une tâche individuelle.
Puissance en combinaison		Nos solutions peuvent communiquer entre elles, par exemple, il est possible de créer des liens de délégation vers des valeurs de ManySpector.
Supporte le passage à l'échelle ?		ManySpector est efficace pour manipuler un certain nombre de valeurs, de par la révélation progressive des valeurs de celles-ci, mais il y a une limite due aux interacteurs. On pourrait envisager d'avoir des « range sliders » en lieu et place de champs de texte par valeur par exemple, pour étendre le passage à l'échelle. Les liens de délégation ne sont pas adaptés à une grande échelle, bien que l'utilisation combinée des liens et de ManySpector augmente la capacité à définir des liens pour des grands ensembles.

Nous avons fait au mieux pour éliminer les problèmes d'utilisabilité évidents, mais nous restons conscients que les techniques d'interactions proposées peuvent souffrir de problèmes d'utilisabilité encore non-identifiés ou considérés comme mineurs dans l'accomplissement des tâches. Nous sommes plus préoccupés par la garantie que nos concepts soient compréhensibles et utiles, et une évaluation de l'utilisabilité aurait pu soulever des problèmes ayant un impact sur l'évaluation de leur utilité.

Dans le chapitre couvrant l'analyse des besoins, nous nous sommes appuyés sur les travaux d'Olsen pour démontrer l'importance de la problématique (Olsen,Jr., 2007). La grille qu'il propose s'applique également aux solutions afin d'évaluer leur utilité. Le (Tableau 4) répond aux critères définis pas Olsen pour ManySpector et les liens de délégations. Les critères sont la réduction de la viscosité, la capacité à amener de nouveaux types d'utilisateurs, et le passage à l'échelle.

B. Expérimentations

Des séances d'évaluations auprès d'utilisateurs ont été menées afin de répondre aux questions que nous nous posions concernant la compréhensibilité des concepts de structuration. Chaque session d'évaluation est divisée en trois parties, chacune dédiée à l'une des trois questions énoncées : compréhension des concepts et des outils, facilité d'accomplissement des tâches, et réalisation de requêtes graphiques de haut niveau. La première partie consiste en un tutorial apprenant aux utilisateurs le principe des liens et valeurs partagées, et comment interagir avec ces concepts dans l'éditeur graphique. Les deux autres parties sont des scénarios conçus pour qu'ils implémentent les exigences que nous avons proposées en page 69.

1. Tutorial

Dans le tutorial, nous avons instruit les utilisateurs à la création d'objets, à la création de liens, au changement de propriété (couleur, épaisseur de contour…), avec un seul objet ou des ensembles d'objets. Le tutorial durait dix minutes, et incluait quinze tâches simples (Tableau 5). Les utilisateurs manipulaient eux-mêmes la souris et réalisaient les interactions tandis qu'ils étaient attentifs à nos instructions. L'objectif de ce tutorial n'était pas seulement d'instruire les utilisateurs, mais aussi de s'assurer qu'ils comprennent les interactions. Nous leur demandions s'ils étaient confiants dans leur compréhension des interactions en les observant accomplir des tâches simples sans instructions pour le confirmer. Nous n'avons pas confirmé le potentiel de découverte étant donné que nous avions commencé par un tutorial. Par potentiel de découverte, nous entendons la capacité des utilisateurs à découvrir l'utilisation des outils de façon

spontanée, sans avoir été instruit de leur utilisation au préalable. Cet aspect est laissé pour des travaux futurs.

Tableau 5. Tâches du tutorial.

Utilisation de l'éditeur
Créer des objets graphiques
Dessiner quatre rectangles Dessiner quatre ellipses
Sélectionner
Sélectionner plusieurs objets graphiques à l'aide du rectangle de sélection Sélectionner plusieurs objets graphiques à l'aide des méta-instruments Sélectionner plusieurs objets graphiques à l'aide des raccourcis clavier Ctrl+A (sélectionner tout), Ctrl+Q (sélectionner tous les visibles)
Couper, Copier, coller
Couper ou copier plusieurs objets graphiques à l'aide des raccourcis clavier Ctrl+X / Ctrl+C, Ctrl +V
Redimensionner
Agrandir deux rectangles en utilisant les poignées de redimensionnement
Utilisation du panel d'échantillons
Changer la couleur de fond d'un rectangle sélectionné par un fond vert Changer la couleur de fond d'une ellipse non sélectionnée par un fond rose Changer la couleur de fond de plusieurs objets sélectionnés Changer le fond d'une ellipse et d'un rectangle par un fond bleu Changer le fond d'une autre ellipse et d'un autre rectangle par un fond jaune
Utilisation de Manyspector
Identifier des ensembles d'objets dans la scène en survolant des valeurs partagées Modifier la largeur d'un objet avec la molette Augmenter l'épaisseur de tous les objets jaunes Donner la plus grande largeur à toutes les ellipses Retirer tous les objets bleus de la sélection Étendre la sélection à tous les rectangles Ne garder dans la sélection que les ellipses
Utilisation de l'outil de création de liens de délégation
Créer un lien d'un rectangle bleu vers une ellipse jaune sur la propriété « couleur de fond » Changer la couleur de fond du rectangle Changer la couleur de fond de l'ellipse Changer à nouveau la couleur de fond du rectangle

2. Deuxième partie : édition multiple d'une scène riche

La deuxième partie de la session était un vrai test. Le test requérait toujours l'utilisation de l'éditeur graphique, mais cette fois avec une scène contenant un grand nombre (50) d'objets différents. Nous avions demandé aux utilisateurs de réaliser des tâches complexes (Tableau 6), telle que « changer l'épaisseur de tous les cercles jaunes pour la valeur la plus haute des épaisseurs de l'ensemble des objets». L'un des bénéfices espéré des valeurs partagées est d'aider les utilisateurs à sélectionner un ensemble d'objets avec des interactions minimales. C'est pour cette raison que nous avons rendu les tâches impliquant la sélection traditionnelle (le rectangle de sélection, ou bien l'ajout successif d'objets à une sélection préétablie avec « shift-clic ») de plus en plus difficiles, d'une part parce qu'elles impliquent des objets multiples (R2.3 *portée de l'action*) ou parce qu'elles impliquent des propriétés graphiques qui ne sont pas perceptivement pré-attentives (R1.1 *rechercher,* R1.4 *identifier les ensembles*). Par exemple, une tâche telle que « changer la couleur de toutes les ellipses » est difficile car elle demande à l'utilisateur de trouver toute les ellipses dans une scène. Il s'agit d'une tâche visuelle non pré-attentive et qui requiert une recherche linéaire (et fastidieuse) des objets graphiques un par un (le lecteur peut le constater par lui-même sur la Figure 101).

Nous n'avions donné aucune instruction, laissant les utilisateurs réaliser les tâches par eux-mêmes selon la voie qui leur semblait la plus pertinente ils étaient alors libres d'effectuer les tâches avec les outils de leurs choix, que ce soit en sélectionnant les formes de façon traditionnelle, ou bien en utilisant ManySpector (R1.2 *désigner*).

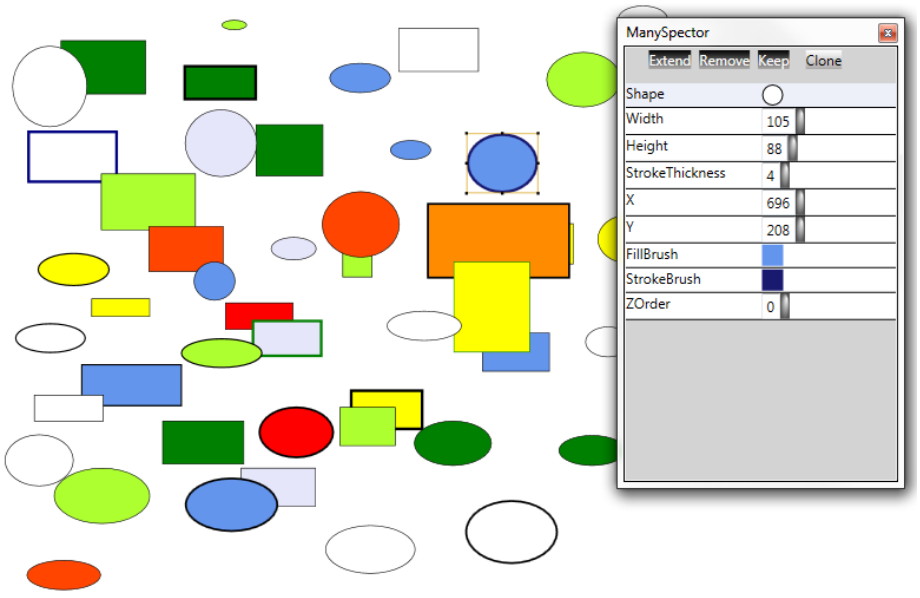


Figure 101. Scène contenant un grand nombre (50) d'objets.

Tableau 6. Tâches du second scénario, chacune faisant l'objet d'une évaluation de leur difficulté.

Donner la plus grande épaisseur des objets jaunes à tous les objets jaunes.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Donner la hauteur maximale des objets de la scène aux objets blancs de la moitié supérieure de la scène.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Sélectionner tous les objets blancs dont l'épaisseur est supérieure ou égale à 3.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Donner un contour rouge aux carrés de cette sélection.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Sélectionner tous les objets rouges, orange et jaunes.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Donner la même nuance de rouge aux objets rouges de nuances différentes.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Donner un contour blanc à tous ces objets.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Faire un lien de délégation d'un rectangle vert clair vers tous les objets verts, et faire varier leur couleur.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5
Faire un lien d'un objet ayant un contour à 1 à tous les autres objets ayant un contour à 1 et faire varier ce contour.	1 : Pas difficile				5 : Très difficile
	□ 1	□ 2	□ 3	□ 4	□ 5

Les objectifs de cette partie était d'observer dans quelle mesure les utilisateurs utilisent de façon volontaire des valeurs partagées et des liens de délégation, s'ils parviennent à effectuer des tâches graphiques non-triviales (R2.1 *nature des actions* et R2.2 *leurs paramètres*), et avec quelle efficacité ils interagissent avec les valeurs partagées et les liens.

3. Troisième partie : modification d'un agenda

La troisième partie du test impliquait l'utilisation d'une application de type agenda. Les utilisateurs manipulaient des évènements dans une vue semaine. Les évènements sont représentés par des rectangles contenant le titre de l'évènement, l'heure de début et l'heure de fin. Les évènements sont placés horizontalement en accord avec le jour de la semaine, et verticalement selon l'heure de la journée. En lieu et place des propriétés graphiques, ManySpector présente les propriétés relatives à un évènement de l'agenda, telles que le titre, la durée, les heures de début et de fin, le lieu etc. comme dans l'inspecteur d'iCal, l'application de Mac OS X, ou encore Google Agenda. ICal permet déjà de modifier plusieurs évènements en une seule fois. Mais ces évènements sont des récurrences d'un même évènement. Contrairement à iCal, ManySpector présente des valeurs partagées, ce qui permet des modifications d'évènements sans liens entre eux.

Les utilisateurs se retrouvaient face à un emploi du temps partiellement rempli (Figure 102) et nous leur demandions d'agir comme s'ils étaient des enseignants devant programmer des séances de cours pour la semaine en collaboration avec le bureau des programmes (le rôle que nous jouions), chargé de la gestion de l'emploi du temps d'une promotion d'élèves. Cet emploi du temps comprenait à la fois des évènements personnels fictifs et des heures d'enseignements. Par exemple, nous leur avons demandé de placer deux heures de cours mercredi après-midi. Puis nous leur avions expliqué que lorsque que l'on exprime « placer un cours à 10h » cela signifiait implicitement « 10h15 », heure habituelle des cours en matinée ; ils devaient donc changer les horaires de début de tous les cours à 10h pour 10h15. La liste des tâches qu'ils devaient accomplir est présentée dans la table suivante (Tableau 7). Certaines d'entre elles pouvaient être récurrentes, (comme la tâche de modification de la durée des « TP de Java »), afin d'inciter les utilisateurs à faire de la structuration explicite. La Figure 103, présente le résultat final.

Tableau 7. Tâches du troisième scénario

Ajouter trois évènements à l'emploi du temps.	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Mettre les créneaux personnels en bleu, et les enseignements en vert. (La sélection traditionnelle est désactivée pour cette tâche)	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Déplacer les enseignements de 10h et 13h à 10h15 et 13h15. (La sélection traditionnelle est désactivée pour cette tâche)	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
On ignore encore la durée finale des TP de Java, on sait que l'on risque de devoir la modifier à plusieurs reprises. Faire varier la durée de tous les TP de Java.	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Changer le terme « Ergo » en « Ergonomie ».	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Renseigner les groupes de TP « S1 » et « S2 ».	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Mettre le nom complet des groupes « IENAC S1 » et « IENAC S2 » au lieu de « S1 » et « S2 ».	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5

Ce que nous souhaitions évaluer grâce à ce scénario était la capacité des utilisateurs à traduire des tâches de haut niveau en interactions graphiques avec nos outils. Les tâches étaient de haut-niveau et demandaient aux utilisateurs d'essayer (R3.1), de percevoir les conséquences (R2.4), d'évaluer (R3.2) et d'effectuer de l'exploration à court terme (R3.3). La scène du calendrier contenant un nombre restreint d'éléments, nous nous attendions au fait que les utilisateurs se reposent sur la sélection

traditionnelle, c'est pourquoi nous les avions contraints à ne pas se servir de la sélection traditionnelle mais d'utiliser plutôt ManySpector.

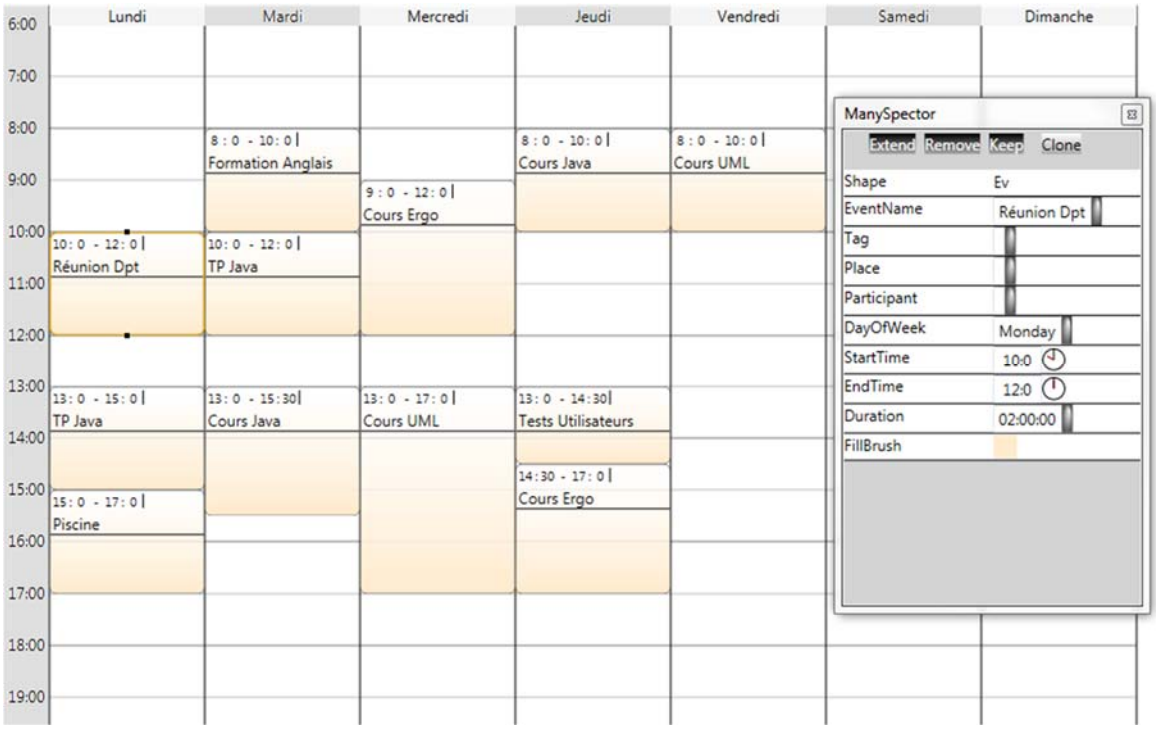


Figure 102. Agenda préremplí que devaient modifier les utilisateurs.

Figure 103. Agenda obtenu après avoir réalisé l'ensemble des tâches.

4. Profil des utilisateurs

Les tests ont été menés auprès de cinq utilisateurs, aux profils variés. Trois d'entre eux utilisent des éditeurs d'agenda quotidiennement. L'un d'entre eux est un designer graphique habitué aux outils tels qu'Illustrator en utilisation professionnelle et le dernier d'entre eux est un utilisateur occasionnel des outils graphiques tels que les outils de présentations. Seul le designer graphique était impliqué dans les processus de conception participative conduits à propos de ce projet, les quatre autres utilisateurs découvraient les interactions pour la première fois.

5. Procédure

Nous demandions aux utilisateurs de procéder en « think aloud », c'est-à-dire d'énoncer à voix haute leurs intentions ainsi que leur compréhension des résultats de leurs actions (Anders & Simon, 1980). Tout en observant leur utilisation des outils, nous notions leurs essais, erreurs, succès et difficultés. A la fin de chaque partie, exception faite du tutorial, nous leur faisions remplir un questionnaire (Tableau 8) afin de leur permettre d'évaluer la difficulté et la lourdeur de chacune des tâches accomplies selon une échelle de Likert allant de 1 (négatif) à 5 (positif), ainsi que d'exprimer des remarques ou commentaires. A la fin de la séance, un questionnaire plus général leur était soumis afin qu'ils évaluent l'utilité et leur appréciation des techniques utilisées, suivant toujours une échelle de Likert allant de 1 à 5. Pour finir, nous leur demandions de remplir un dernier questionnaire afin de mesurer leur satisfaction concernant la séance d'évaluation (Tableau 9). L'ensemble des valeurs obtenues par cette méthode est présenté en annexes du manuscrit page 162.

6. Résultats

Nous n'avons pas observé de problèmes majeurs de compréhension des concepts, des tâches et des outils. Les utilisateurs parvenaient à manipuler les valeurs partagées ainsi que les liens de délégation et à réaliser les tâches simples données lors du tutorial. En les questionnant au sujet de leur confiance, certains d'entre eux répondirent qu'ils avaient le sentiment d'avoir besoin d'un peu d'apprentissage pour procéder correctement, pour « bien faire », selon leurs propres mots. Nous leurs avions montré des interactions cohérentes, cependant les utilisateurs pensaient ne pas pouvoir être familiarisés à ces techniques en un temps si court de par leur nombre. De plus, compte tenu des multiples possibilités offertes pour accomplir une tâche, les utilisateurs étaient toujours désireux de trouver le meilleur moyen de les accomplir. Notre confiance dans la compréhensibilité des concepts s'est accentuée dès lors que les utilisateurs se sont montrés plus à l'aise et en confiance en réalisant les deuxième et troisième parties.

Nous avons même constaté que certains utilisateurs ont essayé des interactions que nous n'avions pas conçues mais qui étaient pourtant parfaitement sensées, telles qu'utiliser des instruments de sélection (*Keeper*, *Remover* et *Extender*) directement sur des échantillons de valeur afin d'éviter la nécessité de réaliser une sélection préalable de la scène entière, ou de déposer des valeurs sur un nom de propriété dans ManySpector afin de l'appliquer à tous les objets ayant cette propriété, ou encore de déposer un échantillon dans un espace vide entre deux valeurs partagées afin d'étendre la sélection. Cela suggère que l'outil conçu était cohérent et prévisible.

Tableau 8. Questionnaire d'évaluation des outils.

Comment évalueriez-vous l'utilité de la boîte de propriété ?	1 : Pas utile				5 : Très utile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Aimeriez-vous avoir cet outil dans vos applications ?	1 : Pas du tout				5 : Absolument
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Comment évalueriez-vous l'utilité de l'édition de lien de dépendances ?	1 : Très utile				5 : Pas utile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Aimeriez-vous avoir cet outil dans vos applications ?	1 : Pas du tout				5 : Absolument
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Dans le scénario « interaction graphique dans un ensemble large », avez-vous trouvé difficile de traduire certaines requêtes demandées dans la manipulation ?	1 : Pas difficile				5 : Très difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Dans le scénario « création et modification d'un agenda », avez-vous trouvé difficile de traduire certaines requêtes demandées dans la manipulation ?	1 : Très difficile				5 : Pas difficile
	☐ 1	☐ 2	☐ 3	☐ 4	☐ 5
Avez-vous des idées/suggestions d'améliorations ?					

Tableau 9. Questionnaire de satisfaction concernant l'utilisation des outils. Les cases en rose indiquent les réponses moyennes des deux scénarios.

	Fort désaccord	Désaccord	Neutre	D'accord	Fortement d'accord
J'ai trouvé facile de faire ce que je souhaitais en réalisant la tâche					
J'ai trouvé qu'accomplir la tâche était facile					
J'ai trouvé frustrant d'accomplir la tâche					
Il était clair de savoir comment réaliser la tâche					
Après avoir accompli une tâche, je savais toujours quoi faire ensuite					
Je devais me concentrer fortement pendant l'accomplissement de la tâche					
J'ai été troublé au cours d'une tâche					
J'étais en confiance en réalisant les tâches					
Compléter les tâches était amusant					
Compléter les tâches était satisfaisant					
Compléter une tâche me rendait nerveux					
J'ai pensé que compléter la tâche était compliqué					
J'ai trouvé que compléter la tâche était trop long					
J'étais stressé en réalisant une tâche					
J'étais agréablement surpris en accomplissant les tâches					

Lors de la deuxième partie, les utilisateurs ont rencontré quelques difficultés en devant réaliser des tâches graphiques complexes (cf. Figure 104, *facilité de traduction dans le scénario graphique*, moyenne : 3.6, écart-type : 0.5). Cela peut s'expliquer par le fait qu'ils étaient toujours dans un contexte d'apprentissage des interactions. Ils nous ont aussi dit qu'ils trouvaient les tâches plutôt abstraites : bien qu'elles étaient délibérément complexes, elles manquaient de sens (personne ne change le contour de tous les cercles jaunes au maximum des contours en situation réelle). Ils ont eu des difficultés pour les comprendre et les mémoriser, ce qui entravait leur capacité à élaborer une solution. Les quatre utilisateurs non-designers ont trouvé les requêtes bien moins difficiles dans la troisième partie, où les tâches de manipulation d'évènements d'emploi du temps étaient plus riches de sens. Malgré tout, les utilisateurs ont été capables de réussir toutes les tâches de la deuxième partie par eux-mêmes (cf. Figure 104, *facilité des neuf sous-tâches dans le scénario graphique*, moyenne : 4.6, écart-type : 0.5).

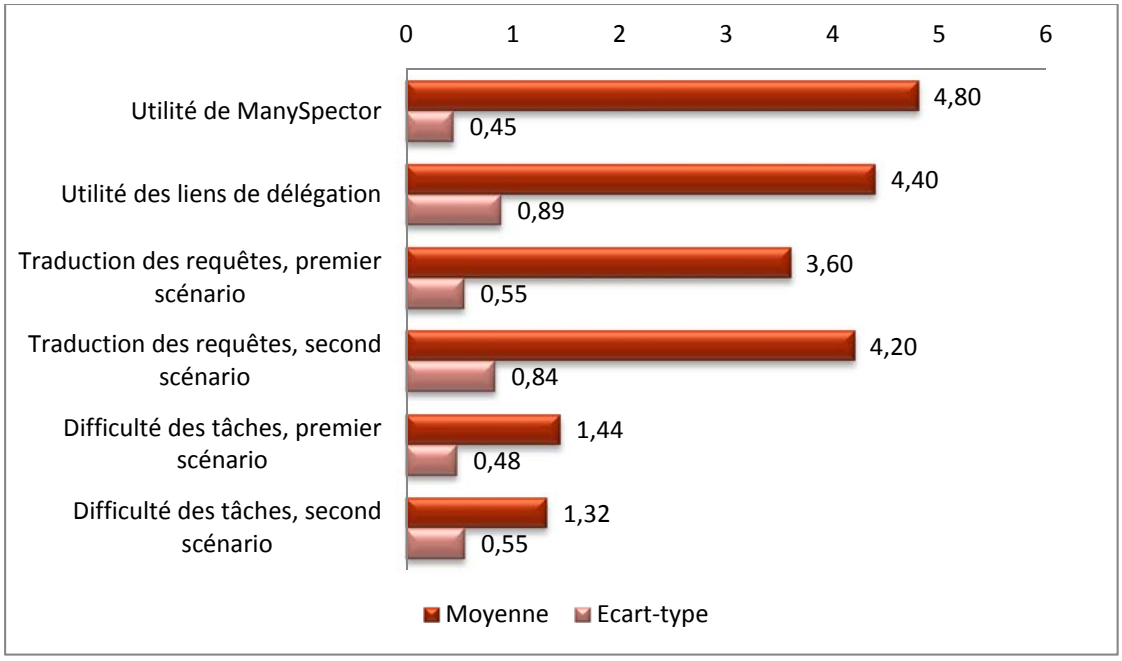


Figure 104. Moyennes et écart-types des difficultés d'utilisation des outils.

La troisième partie ne révélait pas de difficultés qu'auraient pu rencontrer les utilisateurs lorsqu'ils eurent à traduire des tâches de haut niveau en interactions dans le scénario de l'emploi du temps (cf. Figure 104, *facilité de traduction dans le scénario agenda*, moyenne : 4.2, écart-type : 0.8). Nous avons observé une tendance à utiliser la sélection traditionnelle pour des tâches très simples. Lorsque nous restreignions les utilisateurs à employer nos outils exclusivement, ils n'avaient pas plus de difficulté à accomplir les tâches (cf. Figure 104, *facilité des sept sous-tâches dans le scénario*

agenda, moyenne : 4.7, écart-type : 0.5). Cela suggère que les interactions peuvent être appliquées à d'autres contextes que l'édition graphique.

L'utilisation volontaire était un critère que nous souhaitions évaluer lors de ces séances. Nous avons pu observer ce que à quoi nous nous attendions : avec des tâches qui impliquent des propriétés pré-attentives, telles que « changer tous les objets jaunes en objets bleus », les utilisateurs pouvaient parfois s'orienter plus naturellement vers la sélection traditionnelle. Toutefois, ils se sont tournés d'eux-mêmes vers les valeurs partagées offertes par la structuration implicite pour des tâches non-pré-attentives, ou lorsque le nombre d'objet était trop conséquent. Ils ont établi également des liens de délégation lorsque nous leur demandions de faire des interactions répétitives sur un même ensemble d'objets : après quelques répétitions, certains utilisateurs choisissaient de faire d'un objet spécifique un master. Cela leur permettait d'être plus efficaces qu'en élaborant une nouvelle sélection à l'aide de ManySpector.

Même si nous n'avions pas pour objectif d'évaluer l'utilisabilité, les tests ont dévoilé quelques problèmes. Par exemple, interagir avec les champs de texte dans ManySpector s'est avéré difficile : les utilisateurs pouvaient vouloir cliquer sur un champ de texte pour saisir sa valeur en tant que valeur partagée puis la déposer sur une autre valeur, mais au lieu de cela le champ de texte passait en mode d'édition. Les utilisateurs ont également rencontré certaines limites aux interactions proposées, dans certains cas, ils auraient aimé pouvoir garder dans la sélection (avec l'outil Keeper) des objets basés sur une combinaison de valeurs et non selon une seule valeur. Comme prévu, les liens de délégations manquaient de visibilité et de lisibilité lorsqu'ils étaient trop nombreux.

Globalement, les évaluations nous ont offert des réponses positives à nos préoccupations puisque les outils remplissent les exigences établies. Les utilisateurs ont su comprendre les interactions, réaliser des tâches graphiques complexes et traduire des tâches de haut-niveau par l'intermédiaire de ces interactions. Les utilisateurs ont jugé ManySpector très utile (cf. Figure 104, *utilité de ManySpector,* moyenne : 4.8, écart-type 0.4). Ils ont également aimé la structuration explicite par les liens, moins cependant que les valeurs partagées (cf. table *utilité des liens de délégation,* moyenne : 4.4, écart-type 0.9). Ils ont également apprécié le fait qu'il n'y avait pas de stratégie imposée et qu'ils pouvaient accomplir les tâches de la manière qu'ils le désiraient.

CONCLUSION

Le but de la thèse était d'étudier des moyens d'interaction efficaces permettant la manipulation synchrone d'objets multiples. Nous présentons les contributions de la thèse, et nos perspectives pour des travaux futurs concernant l'interaction structurelle.

A. Contributions

1. Interactions avec des objets multiples

La thèse apporte une analyse de besoins inhérents aux activités d'édition graphique. Les utilisateurs ont besoin de structurer leur travail afin d'obtenir un plus grand pouvoir d'action sur leurs objets d'intérêts. Ils ont également besoin d'être capables de désigner des objets et des propriétés spécifiques (inclus ou non dans des structures) et de rechercher des objets pouvant avoir des caractéristiques similaires à d'autres. Ils ont aussi besoin d'explorer des solutions.

De l'analyse de ces besoins a été dérivé un ensemble d'exigences de conception relatives aux interactions basées sur les structures : exigences concernant la gestion des ensembles (rechercher, désigner des objets, modifier les objets et les ensembles, identifier les objets et les ensembles), la gestion des actions (spécifier leur nature, leurs paramètres, leur portée et percevoir leurs conséquences), et la conception exploratoire (essayer, évaluer, explorer à court-terme, comparer les versions et structurer a posteriori).

Nous avons conçu quatre outils interactifs qui illustrent la mise en œuvre de ces exigences. Les liens de délégation permettent de créer explicitement un graphe de délégation de propriétés de manière interactive. Un tel graphe peut également être construit automatiquement grâce à l'IHR, un outil de restructuration automatique. La délégation de propriétés augmente le pouvoir d'action des utilisateurs en leur permettant de propager une modification issue d'une interaction sur un objet à plusieurs autres. ManySpector est un inspecteur de propriété qui révèle une structuration implicite de la scène, et permet de construire des requêtes de sélection et des requêtes graphiques de façon interactive afin de modifier des objets de façon synchrone. Pour finir, Histoglass est un outil qui permet d'observer des états précédent

d'objets afin d'émettre des comparaisons de solutions et de réutiliser le résultat d'actions passées dans la scène courante.

Les interactions conçues sont cohérentes : elles sont toutes basées sur des opérations de glisser-déposer sans modes, qu'il s'agisse d'interaction avec les objets, les échantillons de valeurs, les valeurs partagées, les prototypes etc. Elles supportent également la conception exploratoire en associant feedback immédiat, feedforward et structuration a posteriori : les modifications sont visibles immédiatement au survol des objets, ce qui permet d'essayer et d'évaluer des changements sans les appliquer réellement. La mise en évidence d'objets ayant des relations à d'autres est un moyen efficace pour identifier les ensembles d'objet. Survoler des valeurs partagées montre les objets ayant ces valeurs, en réduisant fortement l'opacité des autres objets. De même, survoler un nœud de la hiérarchie de délégation met en évidence les objets dépendant de ce nœud dans la scène par le même procédé, et survoler un objet révèle les nœuds qui lui délèguent des propriétés.

L'utilisation des valeurs partagées donne accès à plusieurs sous-ensembles et étend notamment la portée des interactions. Bien sûr, les systèmes existants permettent aux utilisateurs d'obtenir les mêmes résultats finaux, en s'appuyant sur des concepts similaires (Flash, Sketchpad). Ces systèmes fournissent effectivement les mêmes *fonctionnalités*, mais pas les mêmes *interactions*. Par exemple, les outils existants permettent aux utilisateurs d'effectuer une recherche graphique, mais avec une manipulation indirecte (par le biais d'une boîte de dialogue, d'un menu). Cela empêche les utilisateurs de rapidement essayer et tester des changements et entrave la conception exploratoire. En outre, les interactions ne sont pas toujours cohérentes. Par exemple dans Illustrator, il est possible d'accéder à une arborescence d'objets graphique, mais les utilisateurs ne peuvent l'utiliser que pour sélectionner une branche puis appliquer un ensemble limité de changements sur la sélection.

Nous avons mené des expérimentations afin d'évaluer les interactions et les outils de structuration. Les expérimentations ont confirmé leur potentiel d'utilité.

2. Interaction Structurelle

Ce travail nous a conduit à l'ébauche d'un nouveau paradigme d'interaction complétant ceux de la manipulation directe et de l'interaction instrumentale : l'interaction structurelle. L'originalité de l'interaction structurelle repose sur la manipulation directe et instrumentale de *structures*, qu'elles soient explicites ou

implicites. L'interaction structurelle offre un moyen d'interagir de façon synchrone avec des objets multiples.

Les paradigmes d'interaction qui reposent sur des métaphores du monde réel sont efficaces mais ont des limites (Frohlich, 1993). Certes, en permettant d'interagir sur des objets d'intérêts comme nous interagissons physiquement avec des objets ou des instruments du quotidien, ces paradigmes réduisent la distance entre l'exécution d'une interaction et l'évaluation de son effet sur le monde. Cependant, la manipulation directe de métaphores du monde réel n'est pas adaptée aux interactions sur des objets multiples, puisque dans la réalité nous ne manipulons directement qu'un objet physique à la fois. Quant à l'interaction instrumentale, si elle prend en compte la manipulation de plusieurs objets, elle ne permet pas de rendre compte de l'utilisation d'entités comme les groupes ou les masters qui certes, servent d'instrument, mais n'en sont pas vraiment pour autant.

Ces structures n'ont pas d'équivalent dans le monde réel, et sont spécifiques à l'informatique : seule l'informatique, en tant que science de l'artificiel et en tant que technologie, permet de les conceptualiser et rendre concret et manipulable afin d'offrir plus de pouvoir d'action aux utilisateurs. En ce sens, l'interaction structurelle peut être considérée comme un élément de l'essence de l' « informatique considérée un outil » : elle permet de tenir la promesse d'augmentation de la capacité d'action de l'utilisateur, en allant au-delà des limites imposées par les paradigmes du monde réel.

Comme nous l'avons montré dans l'état de l'art, nous nous sommes intéressés aux bénéfices que peut apporter la programmation informatique, qui est adaptée à la réalisation de tâches répétitives. L'interaction structurelle se situe donc dans un continuum entre programmation et interaction, car il ne s'agit pas de définir des outils qui nécessitent de formuler des contraintes, d'écrire du code, ou de programmer des macros (qui ont eux aussi pour avantage la possibilité d'agir sur des objets de façon synchrone), mais de créer des techniques d'interaction qui peuvent avoir ces avantages, tout en étant fidèles aux principes de la manipulation directe et de l'interaction instrumentale, à savoir la représentation continue des objets d'intérêts, les actions physiques, et un degré d'indirection minimal.

B. Leçons apprises

La formulation originale du sujet de thèse était différente de la problématique telle qu'exposée dans le présent document. Il s'agissait en effet d'explorer les concepts sous-jacents à la manipulation synchrone d'objets à partir de liens de dépendances.

L'accent était mis sur la visualisation et la manipulation de liens de dépendances entre objets (Master, style etc.). Le sujet de thèse demandait donc à l'origine de concevoir des nouvelles interactions pour créer et manipuler ces dépendances et de mener des recherches concernant leur visualisation. La notion de structure n'était pas identifiée comme importante.

Le processus de conception participative établi nous a permis, par le biais des enquêtes menées en contexte, de mieux définir la problématique. Les utilisateurs étaient effectivement confrontés bien souvent à l'impossibilité de modifier en une seule action des éléments qui partageaient pourtant des propriétés similaires en raison des limites du système : éléments inclus dans des groupes, éléments identiques mais sans liens entre eux, etc. Cette contrainte s'avérait pénalisante dès l'instant où leur activité était une activité de conception exploratoire, c'est-à-dire lorsque le résultat n'était pas connu à l'avance et que le risque d'avoir à modifier le travail déjà établi est important.

Comme nous le prévoyions, les dépendances s'inscrivaient alors comme une réponse à la problématique et, grâce aux ateliers de conception menés, nous avons pu concevoir une première technique d'interaction : celle des liens de délégation. Bien que cette technique offre un pouvoir d'action augmenté, en permettant à l'utilisateur de structurer sa scène et d'agir sur plusieurs objets à la fois, nous restions conscients de ses faiblesses : le coût et l'effort requis pour établir les liens, et les problèmes de visualisation. C'est pourquoi, inspirés par les travaux de Moore, nous nous sommes orientés vers la création automatique d'une telle hiérarchie de délégation. En supprimant le temps et l'effort nécessaire pour établir cette structure, nous pensions améliorer l'utilisabilité des interactions permettant de créer des liens de délégation, mais nous n'avions pas obtenu de grands bénéfices quant à leur visualisation et leur manipulation.

En concevant ManySpector nous avons pu constater que ce qu'il était possible de faire grâce aux liens et la création automatique de liens, ManySpector le permettait également, et ce plus efficacement. En d'autres termes, les promesses de la structuration explicite (le pouvoir d'action) étaient en partie tenues efficacement par la structuration implicite, sans le coût introduit par la gestion d'ue structure explicite. Ceci a été d'ailleurs confirmé par les évaluations menées auprès d'utilisateurs.

Cependant, une qualité faisant défaut à la structuration implicite de ManySpector vis-à-vis des techniques de structuration explicite est la persistance des structures créées : le caractère éphémère des structures implicites utilisées implique qu'il n'est pas possible

de « conceptualiser » la scène, c'est-à-dire de l'organiser selon un pseudo-modèle de données sous-jacent au « monde » étudié. Cet aspect n'a volontairement pas été étudié pendant la thèse : nous nous sommes concentrés sur l'interaction avec les structures et n'avons pas cherché à étudier l'apport « cognitif » de la structuration à l'activité de conception exploratoire.

C. Perspectives

Les outils interactifs que nous avons réalisés présentent certains problèmes. Il subsiste certes quelques problèmes d'utilisabilité déjà mentionnés lors des évaluations, mais le plus important est notamment de trouver des solutions concernant le passage à l'échelle. Les liens de délégations, par exemple, deviennent peu utilisables s'ils sont trop nombreux car la scène devient vite un amas de liens se chevauchant difficilement différenciables. ManySpector ne permet pas non plus d'interagir efficacement avec un trop grand nombre de valeurs partagées, la solution utilisant la barre de défilement et la révélation progressive n'est pas suffisante si la liste de valeurs s'étend trop verticalement.

Pour résoudre ces problèmes, nous pensons pouvoir améliorer certains aspects de ManySpector, comme la présentation des valeurs partagées. Plutôt que de lister l'ensemble de valeurs présentes dans la sélection (représentation discrète), il pourrait être intéressant d'avoir des spectres de valeurs (représentation continue) comme sur les Figure 105 et Figure 106. Ainsi, il serait possible d'accéder à des valeurs non sélectionnées. Cela présenterait plusieurs avantages : pouvoir accéder à ces valeurs pour les utiliser comme des échantillons, dans l'idée d'attribuer ces valeurs de propriété à d'autres objets, ou bien d'étendre la sélection à des objets ayant ces valeurs (en utilisant les méta-instruments *extend*, *remove* et *keep*). De plus, comme les valeurs sont représentées de façon continue, il pourrait être possible de désigner non pas qu'une valeur à la fois mais des intervalles de valeurs, comme sur la Figure 107. Ces améliorations pourraient augmenter la capacité de l'utilisateur à spécifier la nature de l'action (*R2.1*) et sa portée (*R2.3*).

D'autres améliorations possibles seraient de définir des interactions permettant de réaliser des opérations autres que l'affectation de valeur, comme l'incrémentation (actuellement possible sur une valeur numérique à la fois grâce à la molette), par exemple pour augmenter la taille de tous les objets sélectionnés de 20, ou pour diminuer la luminosité ou bien les attributs rouge, vert et bleu des objets de 10.

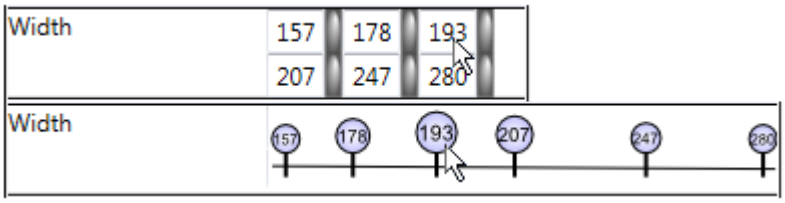


Figure 105. Perspective d'amélioration pour la représentation des valeurs numériques. Plutôt que de présenter un champ de texte par valeur différente, on dispose d'un axe ordonné sur lequel chaque valeur est positionnée. On peut imaginer une vue de type « fisheye » dans le cas où le nombre de valeurs est très élevé, ou la possibilité de zoomer et faire défiler l'axe de valeurs.

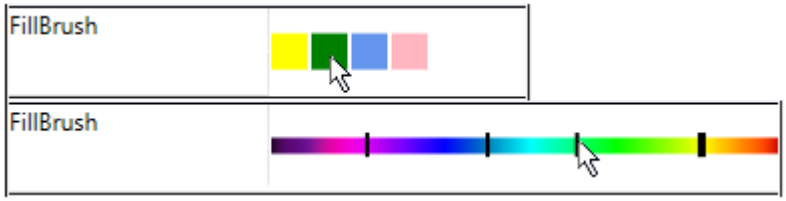


Figure 106. Même principe que la figure précédente, pour la couleur.

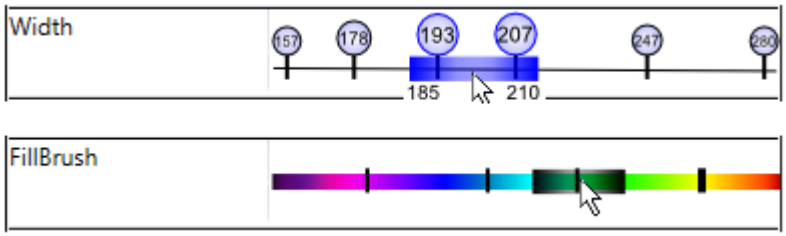


Figure 107. La représentation de valeurs continues permet de sélectionner des intervalles de valeurs et ainsi d'étendre la sélection et d'appliquer des modifications à des ensembles.

La révélation progressive des liens de délégation pourrait être une solution possible pour résoudre les problèmes de visibilités des liens, mais d'autres solutions méritent d'être explorées (Holten, Isenberg, van Wijk, & Fekete, 2011)(Hurter et al., 2013). De plus le système ne vérifie par encore les cycles lorsque l'utilisateur essaie de lier des propriétés (un objet 'A' qui délègue une propriété à un objet 'B' qui la délègue à 'C' qui lui-même la délègue à 'A' est un cycle, tout comme les délégations d'une même propriété d'un objet 'A' à 'B' et de 'B' à 'A'). Une délégation cyclique est un non-sens, causant une boucle théoriquement infinie. Un feedback approprié est alors nécessaire pour prévenir ce problème, en montrant par exemple les liens existants pour avertir d'un potentiel cycle au survol d'une propriété.

Les interactions que nous avons présentées ne sont en rien exhaustives et servent avant tout à illustrer nos propos, en exposant plusieurs façons de créer et d'interagir avec des structures. Comme nous l'avons constaté, si les structures explicites et implicites ont chacune leurs avantages et inconvénients, les résultats des évaluations et nos propres observations démontrent que les structures implicites détiennent un plus

grand potentiel pour offrir du pouvoir d'action tout en favorisant la conception exploratoire. En effet, les coûts de création en temps et effort sont bas et la structure permet d'agir efficacement sur les éléments de l'interface selon leurs propriétés.

Des travaux pourraient alors être poursuivis pour définir d'autres techniques d'interactions basées sur les structures, notamment pour les structures implicites qui restent à découvrir. En effet, les structures explicites sont déjà plutôt répandues (il existe déjà les groupes, les masters, les styles, la délégation etc.), mais quelles autres types de structures implicites pourrait-on concevoir ? Par exemple, dans nos travaux la structure implicite se révèle par la sélection de l'utilisateur, mais nous pourrions envisager qu'elle soit accessible en permanence pour toutes les valeurs de propriétés existantes parmi les objets d'intérêts.

Enfin, il reste sans doute beaucoup à faire pour découvrir de nouvelles structures explicites et améliorer l'utilisabilité des structures explicites existantes. Celles-ci sont sous-jacentes à des logiciels comme les séquenceurs musicaux, ou les systèmes d'instrumentation de processus d'ingénierie système. Même si les interactions avec ces structures sont dépendantes des activités des utilisateurs, nous pensons que l'étude de tels systèmes permettrait d'approfondir et compléter les exigences que nous avons définies, et de tester leur généralité.

BIBLIOGRAPHIE

Anders, K., & Simon, H. A. (1980). Verbal reports as data. *Psychological Review*, *87*(3), 215–251. doi:10.1037/0033-295X.87.3.215

Appert, C., & Beaudouin-Lafon, M. (2006). SwingStates: adding state machines to the swing toolkit. In *Proceedings of the 19th annual ACM symposium on User interface software and technology* (pp. 319–322). New York, NY, USA: ACM. doi:10.1145/1166253.1166302

Appert, C., Beaudouin-Lafon, M., & Mackay, W. E. (2005). Context matters: Evaluating Interaction Techniques with the CIS Model. In S. F. B., MA, LHG FSEDA, P. M. Ms. MSc, D. Moore, & R. R. Bs., , CEng MBCS (Eds.), *People and Computers XVIII — Design for Life* (pp. 279–295). Springer London. Retrieved from http://link.springer.com/chapter/10.1007/1-84628-062-1_18

Appert, C., Chapuis, O., & Pietriga, E. (2012). Dwell-and-spring: undo for direct manipulation. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems* (pp. 1957–1966). New York, NY, USA: ACM. doi:10.1145/2208276.2208339

Bastien, J. M. C., & Scapin, D. L. (1993). Ergonomic criteria for the evaluation of human-computer interfaces. Retrieved from http://hal.inria.fr/inria-00070012

Baudisch, P. (2003). *Halo: Supporting Spatial Cognition on Small Screens*.

Beaudouin-Lafon, M. (2000). Instrumental interaction: an interaction model for designing post-WIMP user interfaces. In *Proceedings of the SIGCHI conference*

on Human Factors in Computing Systems (pp. 446–453). New York, NY, USA: ACM. doi:10.1145/332040.332473

Beaudouin-Lafon, M., & Mackay, W. E. (2000). Reification, polymorphism and reuse: three principles for designing visual interfaces. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 102–109). New York, NY, USA: ACM. doi:10.1145/345513.345267

Beaudouin-Lafon, M., Mackay, W. E., Andersen, P., Janecek, P., Jensen, M., Lassen, H. M., … Jensen, K. (2001). CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets. In *Proceedings of the 22nd International Conference on Application and Theory of Petri Nets* (pp. 71–80). London, UK, UK: Springer-Verlag. Retrieved from http://dl.acm.org/citation.cfm?id=647747.734225

Bier, E. A., Stone, M. C., Pier, K., Buxton, W., & DeRose, T. D. (1993). Toolglass and magic lenses: the see-through interface. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (pp. 73–80). New York, NY, USA: ACM. doi:10.1145/166117.166126

Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, *23*(7), 396–410. doi:10.1145/358886.358895

Cechanowicz, J., & Gutwin, C. (2009). Augmented Interactions: A Framework for Adding Expressive Power to GUI Widgets. In *Proceedings of the 12th IFIP TC 13 International Conference on Human-Computer Interaction: Part I* (pp. 878–891). Berlin, Heidelberg: Springer-Verlag. doi:10.1007/978-3-642-03655-2_97

Chatty, S., Bustico, A., & Athènes, S. (1999). Votre attention s'il vous plait ! Eléments d'un espace de conception de signaux visuels. In *IHM'99* (pp. 17–24).

Conversy, S., Hurter, C., & Chatty, S. (2010). A descriptive model of visual scanning. In *Proceedings of the 3rd BELIV'10 Workshop: BEyond time and errors: novel evaLuation methods for Information Visualization* (pp. 35–42). New York, NY, USA: ACM. doi:10.1145/2110192.2110198

Cypher, A., & Halbert, D. C. (1993). *Watch What I Do: Programming by Demonstration*. MIT Press.

Diaper, D., & Stanton, N. (2003). *The Handbook of Task Analysis for Human-Computer Interaction*. Psychology Press.

Djajadiningrat, T., Overbeeke, K., & Wensveen, S. (2002). But how, Donald, tell us how?: on the creation of meaning in interaction design through feedforward and inherent feedback. In *Proceedings of the 4th conference on Designing interactive systems: processes, practices, methods, and techniques* (pp. 285–291). New York, NY, USA: ACM. doi:10.1145/778712.778752

Dragicevic, P. (2004, March 9). *Un modèle d'interaction en entrée pour des systèmes interactifs multi-dispositifs hautement configurables*. Université de Nantes, Nantes.

Dragicevic, P., Huot, S., & Chevalier, F. (2011). Gliimpse: Animating from markup code to rendered documents and vice versa. In *Proceedings of the 24th annual ACM symposium on User interface software and technology* (pp. 257–262). New York, NY, USA: ACM. doi:10.1145/2047196.2047229

Fallman, D. (2003). Design-oriented human-computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 225–232). New York, NY, USA: ACM. doi:10.1145/642611.642652

Fekete, J.-D. (1996). Les trois services du noyau sémantique indispensables à l'IHM. In *Actes des Huitièmes Journées sur l'Ingénierie des Interfaces Homme-Machine (IHM 96)* (pp. 45–50). "Grenoble. Retrieved from http://insitu.lri.fr/ fekete/ps/ihm96.pdf

Frohlich, D. M. (1993). The history and future of direct manipulation. *Behaviour & Information Technology*, *12*(6), 315–329. doi:10.1080/01449299308924396

Furnas, G. W., & Zacks, J. (1994). Multitrees: enriching and reusing hierarchical structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 330–336). New York, NY, USA: ACM. doi:10.1145/191666.191778

Green, T., & Blackwell, A. (1998). *Cognitive dimensions of information artefacts: a tutorial*. Retrieved from http://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/ CDtutorial.pdf

Green, T. R. G. (1989). Cognitive dimensions of notations. In *Proceedings of the fifth conference of the British Computer Society, Human-Computer Interaction Specialist Group on People and computers V* (pp. 443–460). New York, NY, USA: Cambridge University Press. Retrieved from http://dl.acm.org/citation.cfm?id=92968.93015

Green, T. R. G., & Petre, M. (1996). Usability Analysis of Visual Programming Environments: a `cognitive dimensions' framework. *JOURNAL OF VISUAL LANGUAGES AND COMPUTING*, *7*, 131–174.

Green, Thomas R. G. (2000). Instructions and descriptions: some cognitive aspects of programming and similar activities. In *Proceedings of the working conference on Advanced visual interfaces* (pp. 21–28). New York, NY, USA: ACM. doi:10.1145/345513.345233

Green, Thomas R. G., Borning, A., O'Shea, T., Minoughan, M., & Smith, R. B. (1998). The Stripetalk Papers Understandability as a Language Design Issue in Object-Oriented Programming Systems. In *Prototype-based programming: concepts, languages, and applications*.

Greenberg, S., & Buxton, B. (2008). Usability evaluation considered harmful (some of the time). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 111–120). New York, NY, USA: ACM. doi:10.1145/1357054.1357074

Hewett, T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., … Verplank, W. (1992). *ACM SIGCHI curricula for human-computer interaction*. New York, NY, USA: ACM.

Hoarau, R., & Conversy, S. (2011). Edition synchrone de plusieurs objets: services et interaction. In *23rd French Speaking Conference on Human-Computer Interaction* (pp. 21:1–21:8). New York, NY, USA: ACM. doi:10.1145/2044354.2044380

Hoarau, R., & Conversy, S. (2012). Augmenting the scope of interactions with implicit and explicit graphical structures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1937–1946). New York, NY, USA: ACM. doi:10.1145/2207676.2208337

Holten, D., Isenberg, P., van Wijk, J. J., & Fekete, J.-D. (2011). An extended evaluation of the readability of tapered, animated, and textured directed-edge

representations in node-link graphs. In *Proceedings of the 2011 IEEE Pacific Visualization Symposium* (pp. 195–202). Washington, DC, USA: IEEE Computer Society. Retrieved from http://dl.acm.org/citation.cfm?id=2015551.2015652

Hurter, C., Ersoy, O., & Telea, A. (2013). Smooth Bundling of Large Streaming and Sequence Graphs. Presented at the IEEE PacificVis, Sydney, Australia.

Hutchins, E. (1989). Metaphors for Interface Design. In M. M. Taylor, F. Neel, & D. G. Bouwhuis (Eds.), *The Structure of Multimodal Dialogue* (pp. 11–28). Amsterdam: North-Holland.

Hutchins, E., Norman, D. A., & Hollan, J. D. (1986). Direct manipulation interfaces. In *User Centered System Design; New Perspectives on Human-Computer Interaction* (pp. 87–124). Hillsdale, New Jersey, USA: L. Erlbaum Associates Inc.

John, B. E., & Kieras, D. E. (1996). Using GOMS for user interface design and evaluation: which technique? *ACM Trans. Comput.-Hum. Interact.*, *3*(4), 287–319. doi:10.1145/235833.236050

Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M., & Mackey, K. (1989). The Xerox Star: A Retrospective. *Computer*, *22*(9), 11–26, 28–29. doi:10.1109/2.35211

Kochanek, D. H. U., & Bartels, R. H. (1984). Interpolating splines with local tension, continuity, and bias control. *SIGGRAPH Comput. Graph.*, *18*(3), 33–41. doi:10.1145/964965.808575

Kurlander, D. (1993). Reducing Repetition in Graphical Editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 409–414). ACM.

Kurlander, D., & Bier, E. A. (1988). Graphical search and replace. *SIGGRAPH Comput. Graph.*, *22*(4), 113–120. doi:10.1145/378456.378495

Kurlander, D., & Feiner, S. (1988). Editable Graphical Histories. In *Proc. 1988 IEEE Workshop on Visual Languages* (pp. 416–423). IEEE Press.

Kurlander, D., & Feiner, S. (1992). A history-based macro by example system. In *Proceedings of the 5th annual ACM symposium on User interface software and technology* (pp. 99–106). New York, NY, USA: ACM. doi:10.1145/142621.142633

Kwon, B. chul, Javed, W., Elmqvist, N., & Yi, J. S. (2011). Direct manipulation through surrogate objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 627–636). New York, NY, USA: ACM. doi:10.1145/1978942.1979033

Laurel, B. (1986). Interface as memesis. In *User Centered System Design; New Perspectives on Human-Computer Interaction* (pp. 67–86). Hillsdale, New Jersey, USA: L. Erlbaum Associates Inc.

Lieberman, H. (1986). Using prototypical objects to implement shared behavior in object-oriented systems. In *Conference proceedings on Object-oriented programming systems, languages and applications* (pp. 214–223). New York, NY, USA: ACM. doi:10.1145/28697.28718

Lieberman, H. (2001). *Your wish is my command: programming by example*. Morgan Kaufmann Publishers.

Mackay, W. E. (2002). Which interaction technique works when?: floating palettes, marking menus and toolglasses support different task strategies. In *Proceedings of the Working Conference on Advanced Visual Interfaces* (pp. 203–208). New York, NY, USA: ACM. doi:10.1145/1556262.1556294

Mackay, W. E. (2003). Educating multi-disciplinary design teams. In *Proc. of Tales of the Disappearing Computer* (pp. 105–108).

Maloney, J. H., & Smith, R. B. (1995). Directness and liveness in the morphic user interface construction environment. In *Proceedings of the 8th annual ACM symposium on User interface and software technology* (pp. 21–28). New York, NY, USA: ACM. doi:10.1145/215585.215636

Moore, I. (1995). *Guru - A Tool for Automatic Restructuring of Self Inheritance Hierarchies*.

Moore, I. (1996). Automatic inheritance hierarchy restructuring and method refactoring. In *Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 235–250). New York, NY, USA: ACM. doi:10.1145/236337.236361

Moore, I., & Clement, T. (1996). *A Simple and Efficient Algorithm for Inferring Inheritance Hierarchies*.

Muller, M. J., & Kuhn, S. (1993). Participatory design. *Commun. ACM*, *36*(6), 24–28. doi:10.1145/153571.255960

Myers, B. A. (1992). Demonstrational Interfaces: A Step Beyond Direct Manipulation. *Computer*, *25*(8), 61–73. doi:10.1109/2.153286

Myers, B. A., & Buxton, W. (1986). Creating highly-interactive and graphical user interfaces by demonstration. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (pp. 249–258). New York, NY, USA: ACM. doi:10.1145/15922.15914

Myers, B. A., Giuse, D. A., & Zanden, B. V. (1992). Declarative programming in a prototype-instance system: object-oriented programming without writing methods. In *IN PROC. OOPSLA '92* (pp. 184–200).

Nielsen, J., & Molich, R. (1990). Heuristic evaluation of user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 249–256). New York, NY, USA: ACM. doi:10.1145/97243.97281

Norman, D. A. (1988). *The Psychology of Everyday Things*. Basic Books.

Olsen,Jr., D. R. (2007). Evaluating user interface systems research. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (pp. 251–258). New York, NY, USA: ACM. doi:10.1145/1294211.1294256

Ousterhout, J. K. (1994). *Tcl and the Tk Toolkit*. Addison-Wesley.

Schlienger, C., Conversy, S., Chatty, S., Anquetil, M., & Mertz, C. P. (2007). *Improving Users' Comprehension of Changes with Animation and Sound: An Empirical Assessment*.

Scull, C., Johnson, S., Aliaga, F., Paris, S., Su, S. L., & Durand, F. (2009). Interactive Visual Histories for Vector Graphics. Retrieved from http://dspace.mit.edu/handle/1721.1/45600

Shneiderman, B. (1987). Direct manipulation: A step beyond programming languages. In R. M. Baecker & W. A. S. Buxton (Eds.), *Human-computer interaction* (pp. 461–467). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. Retrieved from http://dl.acm.org/citation.cfm?id=58076.58115

St. Amant, R., Lieberman, H., Potter, R., & Zettlemoyer, L. (2000). Programming by example: visual generalization in programming by example. *Commun. ACM*, *43*(3), 107–114. doi:10.1145/330534.330549

Sutherland, I. E. (1964). Sketch pad a man-machine graphical communication system. In *Proceedings of the SHARE design automation workshop* (pp. 6.329–6.346). New York, NY, USA: ACM. doi:10.1145/800265.810742

Tabart, G., Conversy, S., Vinot, J.-L., & Athènes, S. (2009). Outils d'aide à la conception de rendus graphiques. In *Proceedings of the 21st International Conference on Association Francophone d'Interaction Homme-Machine* (pp. 303–312). New York, NY, USA: ACM. doi:10.1145/1629826.1629875

Taivalsaari, A. (1996). On the notion of inheritance. *ACM Comput. Surv.*, *28*(3), 438–479. doi:10.1145/243439.243441

Terry, M., & Mynatt, E. D. (2002a). Recognizing creative needs in user interface design. In *Proceedings of the 4th conference on Creativity & cognition* (pp. 38–44). New York, NY, USA: ACM. doi:10.1145/581710.581718

Terry, M., & Mynatt, E. D. (2002b). Supporting experimentation with Side-Views. *Commun. ACM*, *45*(10), 106–108. doi:10.1145/570907.570942

Terry, M., & Mynatt, E. D. (2002c). Side views: persistent, on-demand previews for open-ended tasks. In *Proceedings of the 15th annual ACM symposium on User interface software and technology* (pp. 71–80). New York, NY, USA: ACM. doi:10.1145/571985.571996

Terry, M., Mynatt, E. D., Nakakoji, K., & Yamamoto, Y. (2004). Variation in element and action: supporting simultaneous development of alternative solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 711–718). New York, NY, USA: ACM. doi:10.1145/985692.985782

Thomas, F., & Johnston, O. (1981). *The illusion of life: Disney animation*.

Ungar, D., Chambers, C., Chang, B.-W., & Hölzle, U. (1991). Organizing programs without classes. *Lisp Symb. Comput.*, *4*(3), 223–242. doi:10.1007/BF01806107

Ungar, D., & Smith, R. B. (1987). Self: The power of simplicity. In *Conference proceedings on Object-oriented programming systems, languages and applications* (pp. 227–242). New York, NY, USA: ACM. doi:10.1145/38765.38828

Vermeulen, J., Luyten, K., van den Hoven, E., & Coninx, K. (2013). Crossing the bridge over norman's gulf of execution: revealing feedforward's true identity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1931–1940). New York, NY, USA: ACM. doi:10.1145/2470654.2466255

Vitter, J. S. (1984). US&R: A new framework for redoing (Extended Abstract). *SIGPLAN Not.*, *19*(5), 168–176. doi:10.1145/390011.808262

ANNEXES

Dans ces annexes, nous présentons d'autres scénarios de travail issus des enquêtes contextuelles ainsi que les résultats bruts de séances d'évaluation.

A. Scénarios de travail

Nous avons présenté trois scénarios de travail dans la partie « enquêtes contextuelles et analyse des besoins » page 9 afin de mieux définir la problématique et d'identifier les besoins des utilisateurs pour manipuler plusieurs objets en peu d'action. Nous présentons six autres scénarios.

1. Versions de présentations et diapositives partagées

Lorsque Sébastien souhaite préparer ou modifier un cours, il procède souvent en assemblant des parties contenues dans différents documents. Il prend l'exemple d'un cours d'IHM pour la promotion « IENAC S ». Sébastien ouvre alors le document contenant l'introduction. Ce document est formaté selon un certain style, et surtout contient un plan, il s'agit du plan général que Sébastien doit rappeler à chaque début de cours (des cours de 2h dans la majorité des cas) mais aussi au milieu d'un cours lors d'un changement de partie. Par exemple, Sébastien souhaite donner un cours sur la sémiologie graphique, dans la catégorie représentation graphique du cours Modèle pour l'ergonomie. Seulement ce cours est contenu dans un autre document, alors que le plan est contenu dans l'introduction. Donc pour pouvoir rappeler le plan, Sébastien a deux solutions, soit il ouvre les deux documents l'un après l'autre, soit il recopie le plan dans le cours. Ceci pose deux problèmes : les styles appliqués ne sont pas forcément les mêmes, ce qui peux nuire à la mise en page, et surtout, si Sébastien modifie le plan, il lui est très difficile de savoir où il est répliqué afin de mettre à jour toutes ses copies.

Illustre : Le problème inhérent à la duplication de diapositive dans plusieurs documents ; la mise en forme est altérée, il devient vite difficile de savoir dans quels documents sont présentes les copies, et surtout, l'impossibilité de modifier toutes les copies en une seule action.

Parfois, Sébastien décide d'assembler tous les cours en un seul document. Seulement, il apporte très fréquemment des modifications à ce document (corrections orthographiques, réagencements...) qu'il aimerait voir répliquées dans le document

original. Mais les mêmes informations sont contenues dans deux fichiers différents, qu'il s'agisse de dessin (Sébastien pouvant modifier les formes, leurs position...), de contenu (modifier le texte)... Le problème étant que toutes les modifications faites ne sont pas répliquées dans tous les cours partageant ces mêmes informations. De plus, Sébastien souhaiterait pouvoir garder le contrôle sur ces modifications dans le sens où la répercussion sur les copies ne devrait être faite que lorsqu'il le souhaite, car il n'a pas forcément envie de tout modifier dans l'immédiat.

Illustre : Le fait que les diapositives présentes dans plusieurs documents soient considérées comme des entités différentes, de fait les modifications que l'on souhaite faire sur une diapositive ne sont jamais répercutées sur les copies. Cela engendre une grande viscosité à chaque modification.

Au final, Sébastien se retrouve avec des duplicatas du cours « Paradigme d'interaction » dans de nombreux autres documents (cours d'introduction, de Conception Participative, etc...), ce qui l'oblige à se rappeler quelle était la dernière version de cours et surtout où elle se trouve.

Illustre : Il n'existe pas de moyen de savoir où se trouvent les copies d'un élément ainsi que leur nombre.

De plus, ces problèmes se retrouvent au niveau des cours qui diffèrent d'une promotion à l'autre. Par exemple, Sébastien construit un cours complet d'IHM pour le Master IHM, et, en supprimant certaines informations, il construit une version plus « légère » pour des IENAC. Là encore, Sébastien souhaiterait que les modifications faites dans le cours plus léger puissent être répercutées dans l'original.

Illustre : A nouveau le problème de la modification faite sur une copie qui n'est pas répercutée sur l'original.

2. Omnigraffle

Dans ce scénario, Simon souhaite réaliser plusieurs illustrations ayant pour objet les tableaux de « paper strips » du contrôle aérien. Il souhaite notamment concevoir et illustrer un nouveau design de strips, avec des gradients colorés. La (Figure 108) montre le résultat final. Simon commence par dessiner un strip « simplifié », en prenant pour modèle un strip réel. Il utilise un rectangle horizontal, contenant des lignes verticales servant de séparateurs. Il ajoute des informations textuelles spécifiques au strip imité: identifiant de vol, altitude, nom de balises, temps de passage aux balises. Il sait qu'il aura à modifier ces informations lorsqu'il créera d'autres strips, mais ne

sachant pas encore quelles informations vont varier, il préfère utiliser des informations véridiques.

Il duplique à plusieurs reprises le premier strip et superpose les copies afin de faire des tableaux de strips. Simon modifie ensuite les informations textuelles. En réfléchissant au nouveau design avec gradients, il élabore un autre design, basé sur le précédent, en alignant les balises à droite plutôt qu'à gauche (conception exploratoire). Aussi, il copie le tableau précédent, et modifie chacun des strips de la copie pour aligner les balises à droite (Figure 108). Cependant, en essayant de montrer l'intérêt d'utiliser des gradients, il s'aperçoit qu'il doit modifier le texte de certains temps de passage aux balises, afin que deux vols soient en conflit. Il s'exécute d'abord sur le tableau en cours de conception. Afin de limiter les différences entre tableaux aux informations importantes, il doit répéter ces actions sur le premier tableau. Il commence par rechercher les éléments qui doivent être changés (recherche). Cette action de recherche se fait en parcourant l'ensemble des objets graphiques, et en essayant de repérer les éléments candidats au changement, et au risque d'en oublier. Quand il en trouve, il exécute les actions nécessaires à la modification.

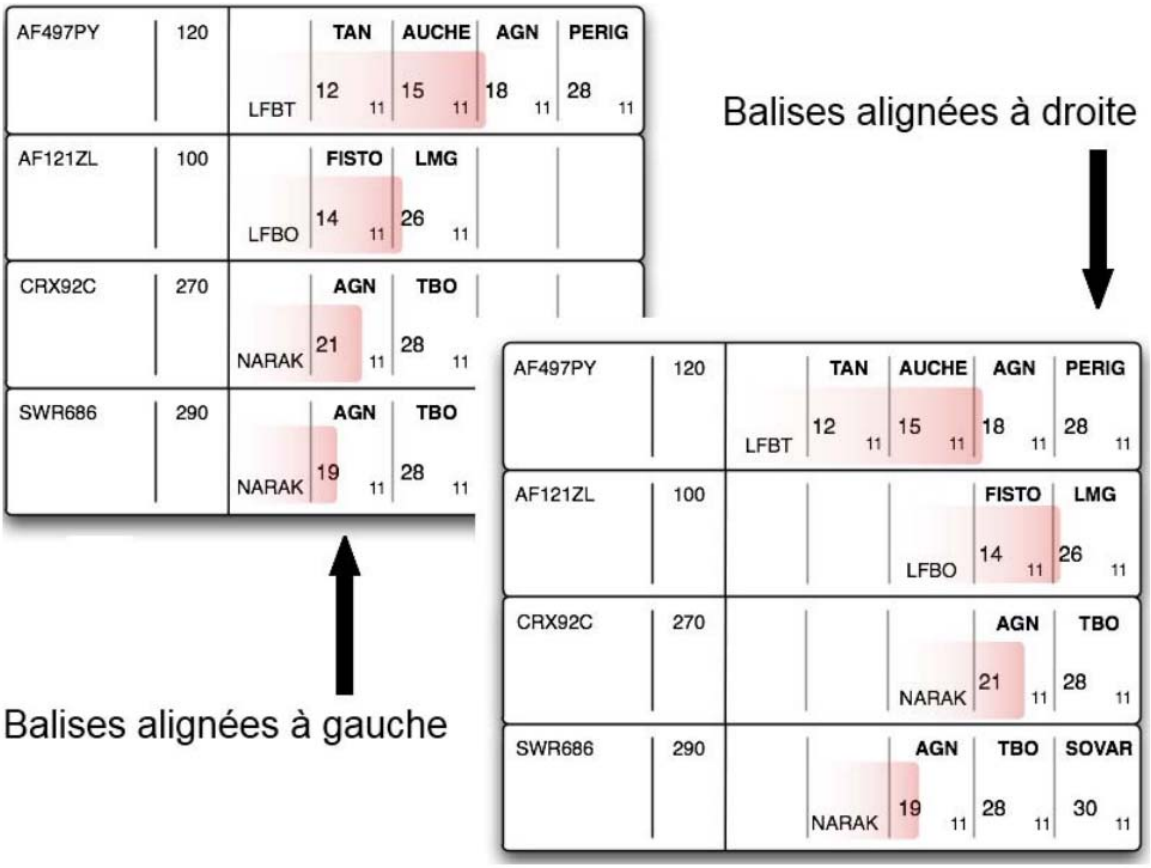


Figure 108. Deux versions d'un tableau de strips.

Illustre : Quand le nombre d'objets graphiques augmente, il est plus difficile de rechercher avec un simple parcours visuel des objets particuliers. Chaque modification devient plus coûteuse, non seulement à cause du nombre d'actions à répéter, mais aussi à cause de l'effort de recherche nécessaire.

3. iCal

Sébastien se sert du calendrier pour prévoir les évènements à venir, mais aussi pour faire des rapports d'activité à sa hiérarchie. Toutes les activités se voient donc attribuer un tag par Sébastien (« ens » pour enseignement, « rech » pour recherche etc...) dans le but de catégoriser ces activités et d'en sortir des statistiques (par exemple, la somme des heures d'enseignement pour une semaine donnée). Sébastien se rend compte qu'il faudrait décomposer certaines catégories en sous catégories : par exemple, la catégorie enseignement pourrait être divisée en cours et préparation de cours, mais aussi divisée en enseignement par promotion (IENAC, IHM...). Ce que Sébastien souhaiterait faire, c'est modifier tous les évènements correspondant à des préparations de cours, et changer le tag « ens » en « ens/prepa ». Pour cela, le seul moyen est de modifier tous les cours un par un ce qui est long et fastidieux (Figure 109).

Illustre : L'absence de moyen pour considérer des entités partageant une propriété commune comme une entité unique, afin d'agir sur l'ensemble de ces entités en une seule fois.

Le cours d'IHM apparaît plusieurs fois sur le calendrier, et il s'agit à chaque fois de la même chose. Sébastien aimerait pouvoir les lier, afin de pouvoir changer le titre ou bien la durée pour tous en une seule action. *Concrètement, Sébastien aimerait pouvoir dire que le titre de tel évènement puisse dépendre d'une seule entité.*

Illustre : A nouveau l'absence du concept de propriété commune à un ensemble d'élément.

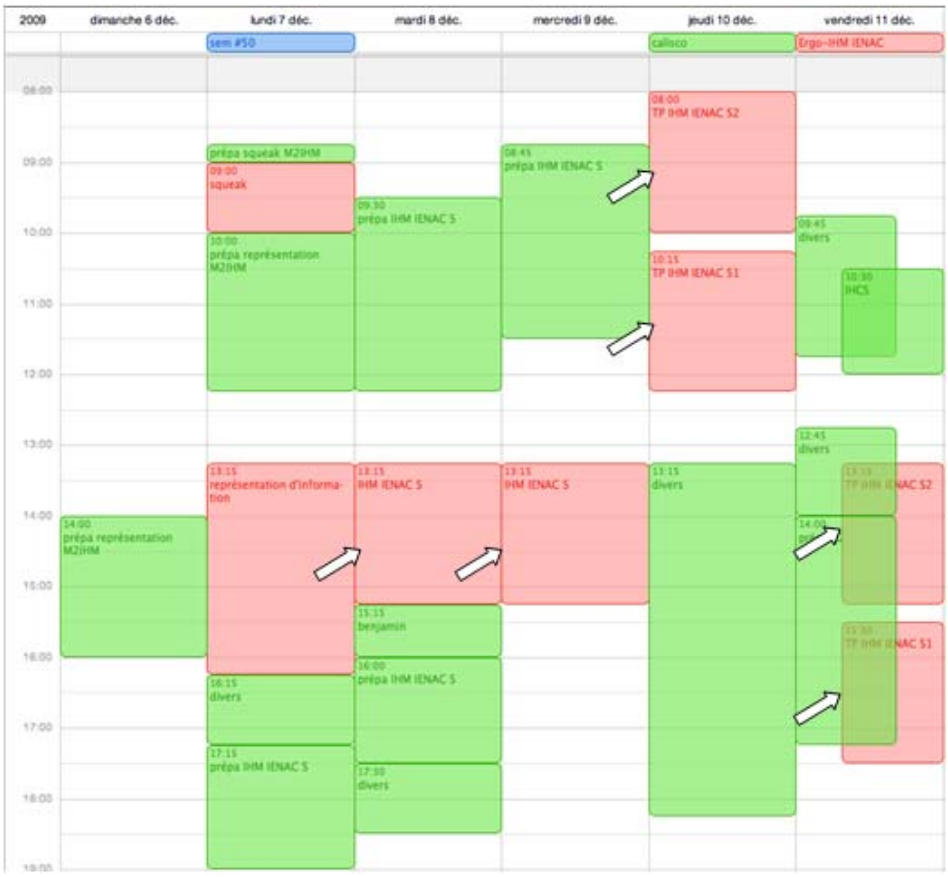


Figure 109. Vue de l'agenda, l'utilisateur doit modifier un par un les évènements marqués d'une flèche.

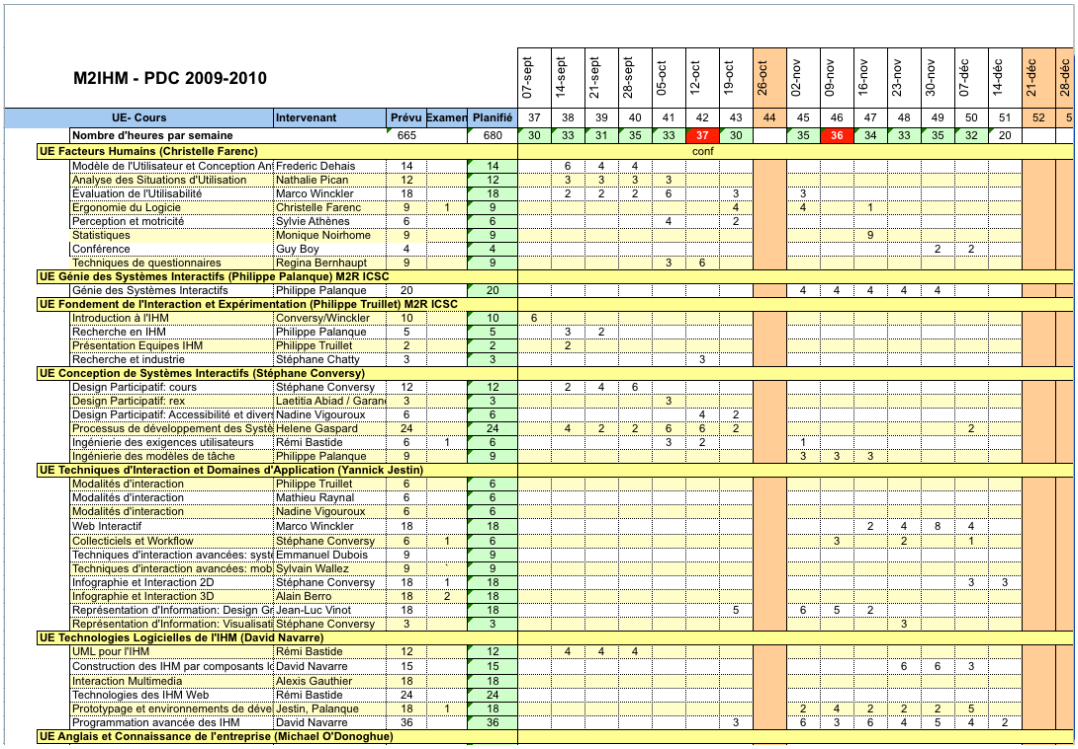
4. Scénario de travail : Planning des cours sous Excel

Sébastien utilise un fichier Excel pour établir un planning des cours (Figure 110), qu'il transmet ensuite aux autres professeurs du Master IHM, afin d'obtenir confirmation que ce planning leur convient. L'intérêt d'Excel pour cette activité est qu'il permet de vérifier si les plannings sont cohérents (en affichant automatiquement des valeurs en rouge si trop d'heures ont été allouées pour une même semaine par exemple). Afin de rendre le planning lisible, Sébastien a formaté les cellules afin d'établir un rendu en « couches » (l'unité d'enseignement (UE) étant une couche supérieure aux autres, suivies par les matières contenues dans l'UE par exemple).

Sébastien souhaite modifier son planning en ajoutant une UE. Pour aller au plus vite, il décide de copier puis de coller un ensemble de cellules correspondant à une UE présente dans le document, pour la renommer ensuite. Malheureusement, le logiciel considérant tout en tant que « cellule », il n'a pas cette notion de couche contenant des entités (un ensemble de cellules ayant un style particulier) les unes au-dessus des autres. Excel ignore donc complètement le formatage définit visuellement par

Sébastien, et se retrouve avec des cellules écrasant les autres, avec des mauvaises bordures (Figure 111). Cela l'oblige à refaire le formatage.

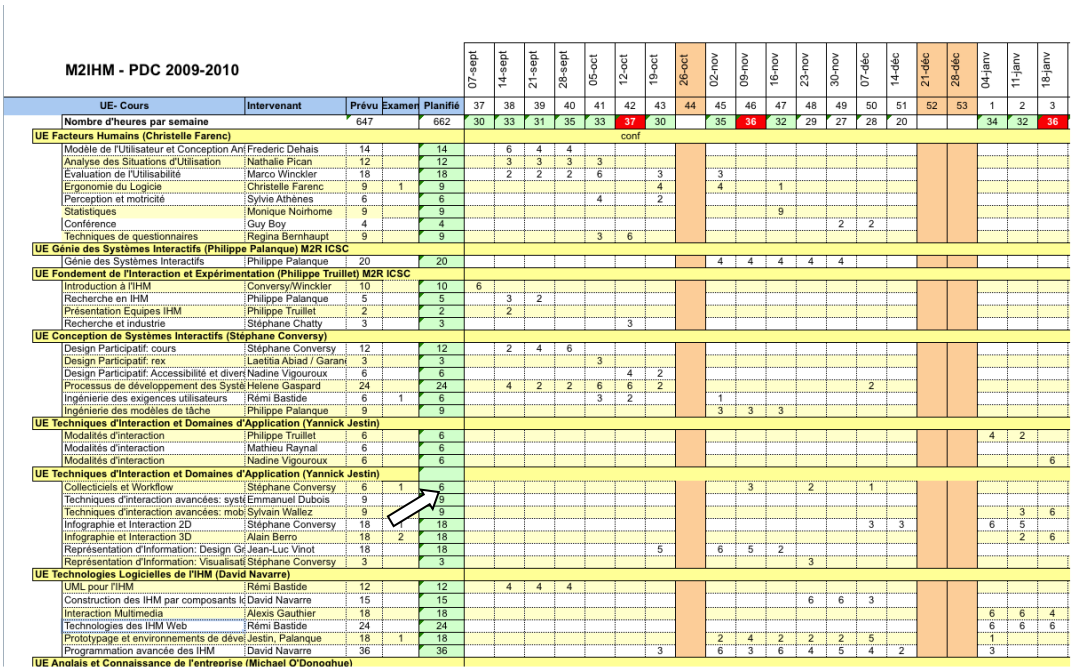
Illustre : Le fait que le logiciel n'ait pas respecté la logique de l'utilisateur dans la mise en page du planning après copie d'éléments.



M2IHM - PDC 2009-2010

UE- Cours	Intervenant	Prévu	Examen	Planifié	07-sept	14-sept	21-sept	28-sept	05-oct	12-oct	19-oct	26-oct	02-nov	09-nov	16-nov	23-nov	30-nov	07-déc	14-déc	21-déc	28-déc
Nombre d'heures par semaine		665		680	30	33	31	35	33	37	30		35	36	34	33	35	32	20		
UE Facteurs Humains (Christelle Farenc)										conf											
Modèle de l'Utilisateur et Conception An	Frederic Dehais	14		14		6	4	4													
Analyse des Situations d'Utilisation	Nathalie Pican	12		12		3	3	3	3												
Évaluation de l'Utilisabilité	Marco Winckler	18		18		2	2	2	6		3		3								
Ergonomie du Logicie	Christelle Farenc	9	1	9							4		4		1						
Perception et motricité	Sylvie Athènes	6		6					4		2										
Statistiques	Monique Noirhome	9		9											9						
Conférence	Guy Boy	4		4														2	2		
Techniques de questionnaires	Regina Bernhaupt	9		9					3	6											
UE Génie des Systèmes Interactifs (Philippe Palanque) M2R ICSC																					
Génie des Systèmes Interactifs	Philippe Palanque	20		20									4	4	4	4	4				
UE Fondement de l'Interaction et Expérimentation (Philippe Truillet) M2R ICSC																					
Introduction à l'IHM	Conversy/Winckler	10		10	6																
Recherche en IHM	Philippe Palanque	5		5		3	2														
Présentation Equipes IHM	Philippe Truillet	2		2		2															
Recherche et industrie	Stéphane Chatty	3		3						3											
UE Conception de Systèmes Interactifs (Stéphane Conversy)																					
Design Participatif: cours	Stéphane Conversy	12		12		2	4	6													
Design Participatif: rex	Laetitia Abiad / Garan	3		3					3												
Design Participatif: Accessibilité et diver	Nadine Vigouroux	6		6						4	2										
Processus de développement des Syst	Helene Gaspard	24		24		4	2	2	6	6	2							2			
Ingénierie des exigences utilisateurs	Rémi Bastide	6	1	6					3	2			1								
Ingénierie des modèles de tâche	Philippe Palanque	9		9									3	3	3						
UE Techniques d'Interaction et Domaines d'Application (Yannick Jestin)																					
Modalités d'interaction	Philippe Truillet	6		6																	
Modalités d'interaction	Mathieu Raynal	6		6																	
Modalités d'interaction	Nadine Vigouroux	6		6																	
Web Interactif	Marco Winckler	18		18											2	4	8	4			
Collecticiels et Workflow	Stéphane Conversy	6	1	6										3		2		1			
Techniques d'interaction avancées: syst	Emmanuel Dubois	9		9																	
Techniques d'interaction avancées: mob	Sylvain Wallez	9		9																	
Infographie et Interaction 2D	Stéphane Conversy	18	1	18														3	3		
Infographie et Interaction 3D	Alain Berro	18	2	18																	
Représentation d'Information: Design Gr	Jean-Luc Vinot	18		18							5		6	5	2						
Représentation d'Information: Visualisati	Stéphane Conversy	3		3												3					
UE Technologies Logicielles de l'IHM (David Navarre)																					
UML pour l'IHM	Rémi Bastide	12		12		4	4	4													
Construction des IHM par composants I	David Navarre	15		15												6	6	3			
Interaction Multimedia	Alexis Gauthier	18		18																	
Technologies des IHM Web	Rémi Bastide	24		24																	
Prototypage et environnements de dével	Jestin, Palanque	18	1	18									2	4	2	2	2	5			
Programmation avancée des IHM	David Navarre	36		36							3		6	3	6	4	5	4	2		
UE Anglais et Connaissance de l'entreprise (Michael O'Donoghue)																					

Figure 110. Planning des cours sous Excel.



M2IHM - PDC 2009-2010

UE- Cours	Intervenant	Prévu	Examen	Planifié	07-sept	14-sept	21-sept	28-sept	05-oct	12-oct	19-oct	26-oct	02-nov	09-nov	16-nov	23-nov	30-nov	07-déc	14-déc	21-déc	28-déc	04-janv	11-janv	18-janv
Nombre d'heures par semaine		647		662	30	33	31	35	33	37	30		35	36	32	29	27	28	20			34	32	36
UE Facteurs Humains (Christelle Farenc)										conf														
Modèle de l'Utilisateur et Conception An	Frederic Dehais	14		14		6	4	4																
Analyse des Situations d'Utilisation	Nathalie Pican	12		12		3	3	3	3															
Évaluation de l'Utilisabilité	Marco Winckler	18		18		2	2	2	6		3		3											
Ergonomie du Logicie	Christelle Farenc	9	1	9							4		4		1									
Perception et motricité	Sylvie Athènes	6		6					4		2													
Statistiques	Monique Noirhome	9		9											9									
Conférence	Guy Boy	4		4													2	2						
Techniques de questionnaires	Regina Bernhaupt	9		9					3	6														
UE Génie des Systèmes Interactifs (Philippe Palanque) M2R ICSC																								
Génie des Systèmes Interactifs	Philippe Palanque	20		20									4	4	4	4	4							
UE Fondement de l'Interaction et Expérimentation (Philippe Truillet) M2R ICSC																								
Introduction à l'IHM	Conversy/Winckler	10		10	6																			
Recherche en IHM	Philippe Palanque	5		5		3	2																	
Présentation Equipes IHM	Philippe Truillet	2		2		2																		
Recherche et industrie	Stéphane Chatty	3		3						3														
UE Conception de Systèmes Interactifs (Stéphane Conversy)																								
Design Participatif: cours	Stéphane Conversy	12		12		2	4	6																
Design Participatif: rex	Laetitia Abiad / Garan	3		3					3															
Design Participatif: Accessibilité et diver	Nadine Vigouroux	6		6						4	2													
Processus de développement des Syst	Helene Gaspard	24		24		4	2	2	6	6	2							2						
Ingénierie des exigences utilisateurs	Rémi Bastide	6	1	6					3	2			1											
Ingénierie des modèles de tâche	Philippe Palanque	9		9									3	3	3									
UE Techniques d'Interaction et Domaines d'Application (Yannick Jestin)																								
Modalités d'interaction	Philippe Truillet	6		6																		4	2	
Modalités d'interaction	Mathieu Raynal	6		6																				
Modalités d'interaction	Nadine Vigouroux	6		6																				6
UE Techniques d'Interaction et Domaines d'Application (Yannick Jestin)																								
Collecticiels et Workflow	Stéphane Conversy	6	1	6										3		2		1						
Techniques d'interaction avancées: syst	Emmanuel Dubois	9		9																				
Techniques d'interaction avancées: mob	Sylvain Wallez	9		9																			3	6
Infographie et Interaction 2D	Stéphane Conversy	18		18														3	3			6	5	
Infographie et Interaction 3D	Alain Berro	18	2	18																			2	6
Représentation d'Information: Design Gr	Jean-Luc Vinot	18		18						5			6	5	2									
Représentation d'Information: Visualisati	Stéphane Conversy	3		3												3								
UE Technologies Logicielles de l'IHM (David Navarre)																								
UML pour l'IHM	Rémi Bastide	12		12		4	4	4																
Construction des IHM par composants I	David Navarre	15		15													6	6	3					
Interaction Multimedia	Alexis Gauthier	18		18																		6	6	4
Technologies des IHM Web	Rémi Bastide	24		24																		6	6	6
Prototypage et environnements de dével	Jestin, Palanque	18	1	18									2	4	2	2	2	5				1		
Programmation avancée des IHM	David Navarre	36		36							3		6	3	6	4	5	4	2			3		
UE Anglais et Connaissance de l'entreprise (Michael O'Donoghue)																								

Figure 111. Après avoir copié et collé une UE, le formatage graphique définit par l'utilisateur n'a pas été respecté.

5. Styles dans Photoshop

Jean crée une forme, un rectangle. A partir d'une boite de dialogue, il peut appliquer un ensemble d'effets à la forme, (contour, ombre portée, texture de remplissage, extrusion, biseautage etc…). Ces effets sont des propriétés de l'objet qui peuvent être modifiées. Jean enregistre cet état de l'objet, c'est à dire cet ensemble de propriétés dans un style. Jean crée ensuite une autre forme graphique, une ellipse, et cherche dans la bibliothèque des styles existants le style qu'il a créé afin de l'appliquer sur l'ellipse. De fait, cette ellipse et le premier rectangle partagent les mêmes effets graphiques. Ce nouvel objet « hérite » du style selon Jean. Ce qui est intéressant selon Jean, c'est que les effets sont indépendants de la forme, ils s'appliquent aux objets qu'il désigne peu importe leur nature.

Illustre : l'utilisateur peut enregistrer un état (un ensemble de propriétés) dans un style que l'on peut ensuite appliquer à d'autres objets.

L'intérêt premier du style selon Jean est principalement d'obtenir une certaine homogénéité entre des objets devant partager un style commun, plus que de favoriser l'action minimale. Jean modifie son premier objet en ajoutant une ombre portée. L'ajout de cette ombre n'est appliqué qu'au rectangle mais pas à l'ellipse qui avait pourtant le même style au regret de Jean.

Illustre : Il n'est pas possible de modifier un objet détenant un style afin de modifier directement ce style pour tous les objets le possédant.

6. Les couleurs dans Illustrator

Jean dessine une ellipse et lui applique un gradient, puis il modifie ce gradient en le déplaçant dans la forme. Jean enregistre ensuite ce gradient qui devient accessible dans la palette de couleurs. En l'appliquant sur une ellipse identique, on s'aperçoit qu'il n'a pas « mémorisé » toutes les propriétés de ce gradient, notamment sa position relative dans la forme. Bien sûr, si Jean tente de modifier le gradient enregistré il n'y aura pas de répercussion sur l'autre forme.

Jean crée un rectangle dont le fond et le contour sont « liés » par une molécule de couleur, qui définit une harmonie de couleurs. Jean peut donc accéder à la molécule de couleur utilisée, et s'il modifie la couleur du fond par exemple, celle du contour sera ajustée en conséquence pour rester en harmonie avec la nouvelle molécule. En revanche lorsque Jean duplique le rectangle, la copie possède bien la même molécule de couleur, mais ils ne sont pas liés dans le sens où modifier sa molécule ne modifiera pas celle de l'original et inversement. « On crée une nouvelle copie mais pas un clone ».

Illustre : Certaines valeurs de propriétés sont liées entre elles au moyen d'une « formule » (la molécule décrivant l'harmonie de couleur) et modifier une couleur de la molécule modifie également les autres. En revanche, deux molécules identiques issues d'une copie d'objet sont indépendantes, il n'est pas possible de modifier l'une afin de modifier l'autre.

B. Résultats des expérimentations

Nous présentons dans cette section l'ensemble des résultats issus des séances d'expérimentations pour chaque utilisateur.

Tableau 10. Les techniques d'interactions (1 : pas utile/pas du tout, 5 : très utile/absolument).

			Utilisateurs				
	Moyenne	Ecart-type	C.	M.	H.	L.	J.
Comment évalueriez-vous l'utilité de la boîte de propriété ?	4,8	0,45	5	5	5	4	5
Aimeriez-vous avoir cet outil dans vos applications ?	4,6	0,55	5	5	4	4	5
Comment évalueriez-vous l'utilité de l'édition de lien de dépendances ?	4,4	0,89	5	5	5	3	4
Aimeriez-vous avoir cet outil dans vos applications ?	4,4	0,89	5	5	4	3	5
Dans le scénario « interaction graphique dans un ensemble large », avez-vous trouvé difficile de traduire certaines requêtes demandées dans la manipulation ?	2,4	0,55	3	2	3	2	2
Dans le scénario « création et modification d'un agenda », avez-vous trouvé difficile de traduire certaines requêtes demandées dans la manipulation ?	1,8	0,84	1	1	2	2	3

Tableau 11. Difficulté des tâches du premier scénario (1 : pas difficile, 5 : très difficile).

			Utilisateurs				
	Moyenne	Ecart-type	C.	M.	H.	L.	J.
Donner la plus grande épaisseur de jaunes à tous les jaunes	1,75	0,96	-	1	3	1	2
Donner la hauteur maximale aux objets blancs de la moitié supérieure de la scène	2,5	0,58	-	2	3	3	2
Sélectionner tous les objets blancs dont l'épaisseur est >= à 3	1,5	1,00	-	3	1	1	1
Donner un contour rouge aux carrés de cette sélection	1,25	0,50	-	1	2	1	1
Sélectionner tous les objets rouges, orange et jaunes	1	0,00	-	1	1	1	1
Donner le même rouge aux rouges différents	1	0,00	-	1	1	1	1
Donner un contour blanc à tous ces objets	1,5	1,00	-	3	1	1	1
Faire un lien d'un rectangle vert clair vers les autres verts clairs, et faire varier la couleur.	1	0,00	-	1	1	1	1
Faire un lien d'un objet ayant un contour à 1 à tous les autres ayant un contour à 1 et faire varier ce contour.	1,5	0,58	-	1	2	2	1
Moyenne :	1,444444444						
Ecart-type :	0,480523441						

Tableau 12. Difficulté des tâches du second scénario (1 : pas difficile, 5 : très difficile).

			Utilisateurs				
	Moyenne	Ecart-type	C.	M.	H.	L.	J.
Ajouter trois évènements à l'emploi du temps.	1,5	0,58	-	2	1	2	1
Mettre les créneaux perso en rouge, et les enseignements en vert. (en limitant la sélection).	2,5	1,29	-	4	3	2	1
Déplacer les enseignements de 10h et 13h à 10h15 et 13h15 (limiter la sélection).	1,25	0,50	-	1	1	2	1
Faire varier la durée des TP Java (tâche récurrente jusqu'à structuration).	1	0,00	-	1	1	1	1
Changer le terme « Ergo » en « Ergonomie »	1	0,00	-	1	1	1	1
Renseigner les groupes de TP S1 et S2	1	0,00	-	1	1	1	1
Mettre le nom complet des groupes IENAC S1 et IENAC S2	1	0,00	-	1	1	1	1
Moyenne :	1,321428571						
Ecart-type :	0,553667426						

Tableau 13. Questionnaire pour le premier scenario (1 : fort désaccord, 5 : fortement d'accord)

			Utilisateurs				
	Moyenne	Ecart-type	C.	M.	H.	L.	J.
I found it easy to do what I wanted to do when completing the task	3,9	0,74	3	4	3,5	4	5
I found completing the task easy	3,8	0,45	4	4	3	4	4
I found completing the task frustrating	2	1,41	1	4	-	1	2
It was clear how to complete the task	3,4	0,89	4	3	2	4	4
When completing the task I always knew what to do next	3	1,00	2	2	4	3	4
I had to concentrate hard when completing the task	2,9	1,14	3	4	4	2	1,5
I got flustered when completing the task	2,5	1,00	2	3	4	2	1,5
I felt in control when completing the task	3,8	0,45	4	3	4	4	4
Completing the task was fun	3,4	0,89	3	4	2	4	4
Completing the task was satisfying	4	0,71	3	4	4	4	5
Completing the task made me nervous	1,6	0,55	1	2	2	2	1
I thought completing the task was complicated	2,4	1,14	2	2	4	3	1
I felt completing the task took too long	2,2	0,45	2	3	2	2	2
I felt under stress when completing the task	1,8	0,84	1	3	2	2	1
I enjoyed completing the task	3,8	0,45	4	4	3	4	4
I was pleasantly surprised when completing the task	3	1,00	4	2	3	2	4

Tableau 14. Questionnaire pour le second scénario (1 : fort désaccord, 5 : fortement d'accord).

			Utilisateurs				
	Moyenne	Ecart-type	C.	M.	H.	L.	J.
I felt under stress when completing the task	1,5	0,50	1	2	1	2	1,5
Completing the task was fun	4	0,00	4	4	4	4	4
When completing the task I always knew what to do next.	4	0,00	4	4	4	4	4
I was pleasantly surprised when completing the task	3,6	0,55	4	3	4	3	4
I got flustered when completing the task	2,2	0,84	3	3	2	2	1
I found it easy to do what I wanted to do when completing the task	4,2	0,45	5	4	4	4	4
I thought completing the task was complicated	2	0,00	2	2	2	2	2
I had to concentrate hard when completing the task	1,9	0,74	1	3	2	2	1,5
Completing the task made me nervous	1,6	0,55	1	2	2	2	1
Completing the task was satisfying	4	0,00	4	4	4	4	4
I found completing the task easy	4	0,00	4	4	4	4	4
It was clear how to complete the task	3,9	0,22	4	4	4	4	3,5
I enjoyed completing the task	3,8	0,45	3	4	4	4	4
I felt completing the task took too long	2	0,00	2	2	2	2	2
I found completing the task frustrating	1,6	0,55	1	2	2	2	1
I felt in control when completing the task	4,2	0,45	5	4	4	4	4

.

Résumé

De nombreux outils interactifs permettent de manipuler efficacement des objets individuellement en appliquant notamment les principes de la manipulation directe. Les interactions avec de multiples objets quant à elles n'ont été que peu étudiées. Notre thèse est qu'il est important de concevoir des interactions permettant la manipulation synchrone d'objets multiples qui soient efficaces, tout en favorisant la conception exploratoire. A partir d'enquêtes menées en contexte, nous décrivons une analyse de besoins ayant permis d'établir un ensemble d'exigences relatives aux interactions sur les ensembles. Nous présentons quatre outils interactifs qui illustrent des concepts d'interaction sur des structures : les liens de délégation, qui s'inspirent du mécanisme de délégation des langages à prototypes en permettant d'établir explicitement des dépendances entre objets (et de créer des clones) et d'étendre la portée des interactions en les propageant ; ManySpector, un nouveau type d'inspecteur de propriétés qui révèle une structuration implicite d'une scène et qui permet de construire, par des interactions instrumentales, des requêtes graphiques et de sélection ; IHR, un outil pouvant créer automatiquement une hiérarchie de propriétés déléguées ; et Histoglass, une lentille déplaçable qui permet de réutiliser localement les propriétés et les actions passées de l'utilisateur. Nous présentons par ailleurs les résultats d'évaluations menées auprès d'utilisateurs afin de mesurer l'intérêt des concepts apportés par les outils conçus. L'ensemble de ce travail nous a conduit à l'ébauche d'un nouveau paradigme d'interaction : « l'interaction structurelle », qui étend les paradigmes de la manipulation directe et de l'interaction instrumentale.

Mots-clés : Interaction homme-machine, Design d'interaction graphique, Interaction structurelle

Abstract

Although many interactive tools allows manipulating effectively objects individually by applying principles such as direct manipulation, interaction with multiple objects have been little studied so far. Our thesis is that it is important to design interactions for the synchronous operation of multiple objects that are efficient, while fostering exploratory design. From contextual inquiries, we describe an analysis of needs which established a set of requirements for interactions on sets. We present four interactive tools to illustrate concepts of interaction with structures: links delegation, which are based on the delegation mechanism in prototype languages, allowing to establish dependencies between objects (and create clones) and extend the scope of interactions by propagating them; ManySpector, a new type of property inspector that reveals an implicit structuring of a scene and allows for building, with instrumental interactions, graphical query and selection; IHR, a tool that can automatically create a hierarchy of delegated properties; and Histoglass a movable lens that allows to locally manipulate the properties and past user actions. We also present the results of evaluations conducted with users in order to measure interest in the concepts provided by the tools developed. All of this work has led us to the draft of a new interaction paradigm, "the structural interaction", which extends the paradigms of direct manipulation and instrumental interaction.

Key-words : Human-computer Interaction, Graphical interaction design, Structural interaction