Edition synchrone de plusieurs objets : services et interaction

Raphaël Hoarau, Stéphane Conversy

Edition Synchrone de Plusieurs Objets : Services et Interaction

Raphaël Hoarau
Université de Toulouse
ENAC - IRIT
7, rue Edouard Belin
31055, Toulouse, France
+33 5 62 17 43 44
raphael.hoarau@enac.fr

Stéphane Conversy
Université de Toulouse
ENAC - IRIT
7, rue Edouard Belin
31055, Toulouse, France
+33 5 62 17 40 19
stephane.conversy@enac.fr

RESUME
Les interactions graphiques permettant de modifier plusieurs objets en même temps n'ont pas fait l'objet d'études systématiques. En nous basant sur des entretiens en contexte de designers, et sur les systèmes et travaux existants, nous identifions les services nécessaires à une interaction efficace avec plusieurs objets : gestion des ensembles, gestion des actions, et incitation à la conception exploratoire. Nous présentons ensuite des interactions qui fournissent un sous-ensemble des services identifiés, en se basant sur une boite de propriété améliorée. Ce travail peut inspirer les designers pour la conception d'interactions plus cohérentes et plus puissantes.

Mots clés
Interaction graphique, manipulation directe, interaction instrumentale, conception exploratoire.

ABSTRACT
Graphical interactions that modify many objects at once have not been studied systematically. Based on contextual inquiries of designers, and on previous systems and work, we have identified the services required for an efficient interaction with multiple objects: set management, action management, and support for exploratory design. We present a number of interactions that provide a subset of the identified services, by relying on an improved property sheet. This work may enable designers to design more consistent and more powerful interactions.

Categories and Subject Descriptors
H.5.2 [**Information interfaces and presentation**] User Interfaces – *Interaction Styles.*

General Terms Design, Human Factors.

Keywords Graphical Interaction Design, direct manipulation, instrumental interaction, Exploratory Design.

IHM'11, October 24-27, 2011, Sophia Antipolis, France

1. INTRODUCTION
Un grand nombre d'outils interactifs permettent de manipuler des objets de façon individuelle, par exemple changer leur couleur ou leur épaisseur de trait : création graphique (schéma, AutoCAD), systèmes de présentation (PowerPoint), de conception d'interfaces, ou de création de routes aériennes. D'autres outils offrent des moyens d'action supplémentaires, dépassant ceux du monde réel. Par exemple, certaines applications proposent de manipuler des groupes, des styles (CSS ou WPF), ou des « master » (PowerPoint), pour manipuler des ensembles identifiés d'objets « en une seule fois ».

Ces interactions avec des objets multiples ont été peu étudiées en tant que telles. Certes, il existe des guides de conception, ou des paradigmes comme la manipulation directe, ou l'interaction instrumentale, 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 pour appliquer une interaction sur plusieurs objets ?

Cet article a pour objectif de présenter les concepts sous-jacents à l'édition synchrone de plusieurs objets. Après avoir illustré la problématique par des scénarios, nous définissons les services nécessaires pour rendre efficace cette activité, et nous décrivons des interactions nouvelles qui implémentent ces services.

2. SCENARIOS DE TRAVAIL
Afin d'étudier des cas concrets et réalistes, nous avons mené des entretiens en contexte auprès de cinq « concepteurs », ce dernier terme étant à prendre au sens large : création graphique (Illustrator), d'emploi du temps pour une formation (iCal), de plan de site géographique (AutoCAD), ou de cours (PowerPoint). Nous présentons deux de ces cas d'étude sous forme de scénarios, afin de mettre en exergue les interactions nécessitant des actions sur plusieurs éléments à la fois. Certaines étapes des scénarios sont annotées par un terme en italique que nous détaillons par la suite.

2.1 Clavier logiciel
La tâche décrite dans ce premier scénario consiste en la réalisation d'un clavier logiciel au moyen d'un éditeur graphique (Illustrator). L'utilisateur commence par créer une première touche du clavier, en dessinant un rectangle à coins arrondis. Puis

il lui applique un gradient, et l'intègre dans un autre rectangle servant de contour. Il place ces objets dans un même groupe constituant la touche. Il ajoute ensuite un effet d'ombre portée, place un label dont le texte est 'A' au centre de la touche, et centre correctement l'ensemble grâce à une commande d'alignement. Il incorpore ensuite le label et la touche dans un même groupe qu'il nomme « touche » (*structuration*).

Cette première touche va servir de modèle pour créer toutes les autres. L'utilisateur duplique cette touche et applique une translation horizontale à la copie pour qu'elle soit à côté de l'originale. Il procède ainsi plusieurs fois, obtenant un ensemble de touches alignées portant toutes la lettre 'A' (Figure 1). Il modifie la lettre des touches une par une pour obtenir un clavier « AZERTYU ».

Figure 1. L'utilisateur créé une première touche, et la duplique plusieurs fois.

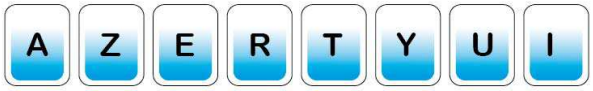

Figure 2. Le texte de la touche I n'est pas centré.

Cependant, arrivé à la lettre 'I', il s'aperçoit que le texte 'I' n'est pas centré par rapport à la touche (Figure 2). Le premier objet était mal spécifié : si les trois objets (texte, fond, contour) ont bien été alignés, le texte n'était pas centré. Avec les lettres A, Z, E, R, T, Y, U, le problème n'est pas visible. Ce n'est qu'à la lettre 'I' que le concepteur s'en est rendu compte. Chaque lettre étant dans un groupe hétérogène, le système ne lui propose pas de commande pour modifier chacun des éléments textuels des groupes en les sélectionnant tous *(désignation*). Il doit cliquer à plusieurs reprises sur chaque groupe pour atteindre le texte et avoir accès aux commandes d'alignement de texte. Cependant, il ne peut pas faire cette interaction pour tous les textes en même temps car ils sont contenus dans des groupes différents. Aussi, l'utilisateur estime qu'il est plus efficace de tout refaire, plutôt que de modifier un par un chaque exemplaire de la première touche (*action*). Il décide donc de supprimer toutes les copies. Puis il dégroupe la première touche, centre le texte, et crée à nouveau chacune des touches, avant de changer chaque lettre. Ce scénario illustre plusieurs tâches que doit réaliser l'utilisateur.

2.1.1 Gestion des ensembles

L'utilisateur s'est appuyé sur la capacité du système à permettre de créer des ensembles, les modifier et les maintenir. Par exemple, il a créé un groupe pour faire une touche à partir d'un rectangle, d'un contour rectangulaire, et d'un texte. Sur cette vue du logiciel utilisé, la seule opération possible sur un groupe existant est le dégroupement (pas de modification, d'ajout, ou de retrait).

2.1.2 Spécifier/désigner une propriété et une valeur

Le système doit permettre de spécifier facilement une ou plusieurs propriétés que l'utilisateur souhaite modifier. Par exemple, l'utilisateur doit changer la propriété « alignement » avec la valeur « centré ».

2.1.3 Actions à portée multiple

Les actions à portée multiple offrent à l'utilisateur de pouvoir agir sur un maximum d'éléments en un minimum d'actions. Ainsi, si l'utilisateur utilise un groupe pour une touche du clavier, c'est notamment pour déplacer en une seule action (clic+mouvement) trois objets à la fois, tout en conservant les positions relatives des trois objets. A l'inverse, il ne peut pas modifier en une seule fois la propriété « alignement ».

2.2 Dessin de strips

Dans ce scénario, l'utilisateur 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 3 montre le résultat final. L'utilisateur 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.

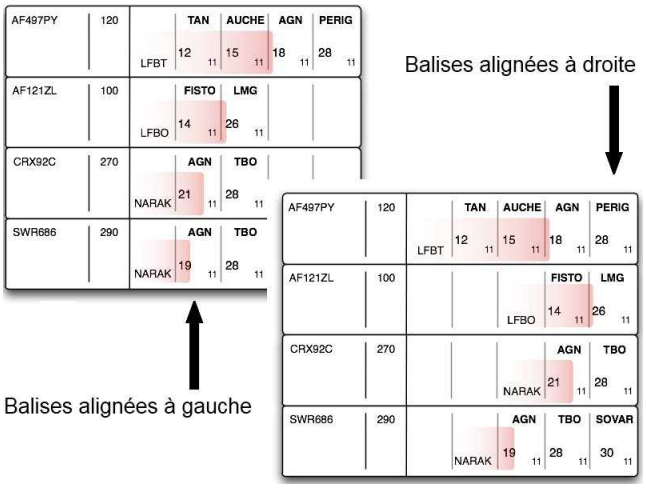

Figure 3. Deux versions d'un tableau de strips.

Il duplique à plusieurs reprises le premier strip et superpose les copies afin de faire des tableaux de strips. L'utilisateur 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 3). 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. Outre les taches déjà évoquées avec le premier scénario, ce scénario illustre deux nouvelles activités.

2.2.1 Recherche/désignation

Quand le nombre d'objets graphiques augmente, il est plus difficile de rechercher avec un simple parcours visuel des objets particuliers. Dans le scénario précédent, 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.

2.2.2 Conception exploratoire

Dans le scénario précédent, l'utilisateur explore des solutions possibles, et modifie des parties de solutions déjà conçues. Il exerce une activité de « conception exploratoire » (« exploratory design »), dont le résultat final désiré n'est pas connu à l'avance [8] [22]. La conception exploratoire se manifeste durant les activités de dessin et d'édition graphique (sketching), mais aussi de réalisation de slides de présentation, ou encore de conception d'une hiérarchie de classe. En combinant action, visualisation du résultat et réflexion, elle engendre une co-découverte du problème et de la solution : il n'est plus nécessaire de réfléchir avant d'agir, mais d'agir pour aider à réfléchir.

3. ETAT DE L'ART

3.1 Existence et importance du problème

3.1.1 Critères ergonomiques

Pouvoir apporter beaucoup de modifications en un faible nombre d'actions est un besoin déjà identifié. Le critère ergonomique de *brièveté*, et plus précisément le sous-critère d'*actions minimales*, décrit la nécessité de limiter les étapes par lesquelles doivent passer les utilisateurs [3].

3.1.2 Dimensions Cognitives

Dans les dimensions cognitives [7] le problème observé dans le scénario du clavier logiciel est nommé *viscosité*. Il se manifeste quand la structure de l'information contient beaucoup de dépendances entre ses parties, impliquant qu'un petit changement nécessite de nombreux ajustements de la part de l'utilisateur. La viscosité est un frein à la modification et à la conception exploratoire [8]. Comme il peut être coûteux de faire des changements, l'utilisateur se retient d'explorer des solutions possibles.

Comme nous l'avons vu, la viscosité est un problème pour l'exploration et la modification. Pour réduire la viscosité, une solution consiste à créer des « abstractions », une « power command » qui agit sur plusieurs objets [8]. L'abstraction est une dimension cognitive, qui se définit comme une classe d'entités, ou un regroupement d'éléments que l'on va traiter comme une seule entité afin de réduire la viscosité, ou de rendre la notation plus proche du modèle conceptuel. Les styles dans un éditeur de texte sont un exemple d'abstraction.

Les abstractions peuvent être coûteuses. Leur création et édition demandent temps et effort. L'investissement dans la gestion d'une abstraction est à comparer à l'investissement en actions simples pour résoudre de petits problèmes : construire des structures peut-être plus coûteux que de répéter un petit nombre d'actions. Par ailleurs, l'abstraction peut être un frein au design exploratoire : si l'abstraction n'est pas la bonne, il faut la modifier. Enfin, les abstractions doivent être apprises. Par exemple, un utilisateur novice ne va peut-être pas utiliser l'abstraction « style » de son traitement de texte s'il n'en a pas connaissance.

3.2 Structuration des scènes

3.2.1 Notion de groupe

Afin d'agir sur plusieurs élément en une seule fois, les éditeurs graphiques traditionnels permettent d'agir sur une sélection définie par l'utilisateur ou de former des groupes, la sélection pouvant être vue comme un groupe temporaire. Cependant, la seule opération possible pour un groupe est sa destruction (pas de modification, d'ajout, ou de retrait).

3.2.2 Notion de « Master »

La notion de master fait référence à un élément qui sert de modèle à d'autres. Sketchpad [19] introduisait déjà la notion de « master ». Dans Sketchpad, il était possible de dessiner une forme graphique, et de la dupliquer en créant ainsi des instances de cette forme. Modifier le Master permettait alors de modifier chacune des instances. De même, Kabuki [21] 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. Le principe de master offre du support à la structuration et permet d'agir sur plusieurs éléments en une seule fois.

3.2.3 Notion de « Tags »

De même, les Tags de Tk [18] permettent de structurer des ensembles, avec comme particularité la possibilité pour un élément d'avoir plusieurs tags (donc d'appartenir à plusieurs groupes). Une commande Tcl peut être appliquée sur un élément ou sur un tag (*portée*). Enfin, les styles et modèles peuvent être codés dans un langage de style, comme CSS, avec une structuration de type hiérarchique. Le graphe de scène de SwingStates [1] introduit les concepts de tags intensionnels et extensionnels (présents aussi dans Presto [6]). Les tags extensionnels sont explicitement ajoutés (ou retirés) par le développeur aux instances, tandis que les tags intensionnels dépendent d'un prédicat (par exemple, tous les objets bleus). Les tags intensionnels sont donc un moyen pour structurer de manière automatique des ensembles. Les mécanismes de tag ne sont disponibles que pour les programmeurs, et rarement pour les utilisateurs.

3.2.4 Recherche graphique

La technique du *Graphical Search & Replace* [11] (disponible dans Illustrator par exemple) permet de modifier un ensemble d'éléments en décrivant par des propriétés graphiques ce que l'on cherche *(désignation*) et ce que l'on souhaite modifier (*portée multiple*). Cette technique bénéficie d'une structuration implicite de la scène en ensemble partageant des propriétés graphiques.

3.2.5 Prototypes

Les langages à prototypes comme Self offrent une alternative aux langages à base de classe pour la programmation orientée objet [25]. Ils offrent un modèle de création flexible, permettant le partage de propriétés et de comportement [13]. Ces mécanismes permettent de structurer une hiérarchie de prototypes et d'agir sur plusieurs instances en manipulant un prototype auquel les propriétés ou le comportement est délégué. Des outils ont été proposés pour permettre une structuration a posteriori : Guru

restructure automatiquement des hiérarchies de délégation, afin d'abstraire des éléments communs à plusieurs prototypes et obtenir une hiérarchie sans doublons [17]. Morphic [16], l'interface graphique de Self, réifie les prototypes en objet graphique (les Morphs), et permet leur construction et leur édition par manipulation directe.

3.2.6 Macros et programmation par démonstration

Les macros définissables par l'utilisateur permettent d'automatiser des tâches répétitives [10]. L'utilisateur réalise un exemple de la tâche à accomplir sur un ensemble d'objets, et cet exemple est généralisé pour être applicable sur d'autres entrées et contextes [12]. Si les macros offrent du degré d'action en permettant de réaliser plusieurs actions sur plusieurs éléments, leur création est de l'ordre de la structuration. Ces techniques sont qualifiées de « programmation par démonstration » ou « programmation par l'exemple » [14]. La programmation par démonstration peut reposer sur l'interprétation correcte des actions de l'utilisateur par la machine. Nos travaux s'inscrivent dans une perspective d'instrumentation, qui ne compte pas sur l'intelligence (parfois insuffisante ou erronée) du système.

3.3 Conception et évaluation des techniques d'interaction

Les techniques de manipulation individuelle, comme la manipulation directe [20], ou instrumentale [4] sont efficaces pour interagir avec un objet seul. Ces méthodes permettent de réduire le nombre d'actions que doit réaliser l'utilisateur.

3.3.1 Réification, polymorphisme et réutilisation

La réification en instruments [5] permet d'offrir des moyens d'interaction sur des objets qui vont en contrôler d'autres. Agir sur l'interface autrement qu'en agissant sur les objets primaires étend les possibilités d'action. Les degrés définis pour les instruments, degrés spatiaux et temporels, de compatibilité et d'intégration, permettent d'évaluer leur efficacité. La réification peut être considérée comme l'acte de rendre manipulable directement une abstraction. Le polymorphisme, en plus de factoriser le vocabulaire gestuel, permet la portée multiple en autorisant la formation de groupes hétérogènes. On réduit le nombre d'actions à accomplir en évitant de reformer ces groupes si ces actions sont polymorphiques. De même, la réutilisation réduit le nombre d'actions nécessaires à l'établissement d'un ensemble d'objets.

3.3.2 Conception exploratoire

VisiCalc est le système emblématique de la conception exploratoire. Son efficacité, due à la représentation continue des éléments d'intérêt et au déclenchement automatique d'actions, incitait l'utilisateur à essayer plusieurs configurations, à découvrir de nouvelles solutions, voire à découvrir une nouvelle formulation du problème. Plusieurs techniques favorisent la conception exploratoire. L'undo/redo, et plus précisément les actions réversibles [20], sont un mécanisme important pour l'exploration [22]. *Side Views* [24] offre des preview de commandes interactives. *Parallel Paths* [23] est un modèle d'interaction pour l'exploration de solutions alternatives, reposant sur une arborescence de création plutôt qu'un processus linéaire, et sur la visualisation simultanée de plusieurs nœuds. De fait, agir sur un nœud supérieur avant séparation modifie l'ensemble des dérivés (*portée multiple*).

La structuration a priori est un frein à la conception exploratoire. Par exemple, les classes de certains langages dits orientés objets ont pour inconvénient de forcer le programmeur à penser à des descriptions abstraites, avant de pouvoir instancier et expérimenter avec des exemples concrets. La structuration a priori est un exemple d'engagement prématuré (premature commitment), une autre dimension cognitive.

3.3.3 Evaluation des techniques d'interaction

[15] présente l'évaluation de différentes techniques d'interaction. Ainsi, une technique n'est pas meilleure qu'une autre dans l'absolu, mais certaines se montrent plus efficaces que d'autres selon la tâche à accomplir : *copie*, *modification* ou *problem-solving* (similaire à la conception exploratoire).

CIS est un modèle qui permet de décrire une technique d'interaction, de l'analyser et prédire son efficacité dans un contexte d'utilisation pour une séquence d'interaction donnée [2]. Une interface est définie comme un ensemble d'objets manipulables de deux sortes : des objets de travail tels que des formes dessinées, et des objets outils comme un item de menu. Les actions sont de deux types : la sélection qui identifie un sous ensemble de l'espace d'interaction (déplacement d'un objet, comme le curseur vers un outil) et la validation, comme un clic souris, qui confirme l'action de sélection.

CIS définit quatre propriétés des techniques d'interaction. L'ordre et le parallélisme, définissent si la technique d'interaction impose une organisation parallèle ou séquentielle de ses actions. La persistance indique si la technique va modifier des attributs des objets outils et affecter la prochaine utilisation de la technique. La fusion est la capacité de la technique à modifier plusieurs objets de travail en définissant plusieurs manipulations en une seule fois (similaire à la *portée multiple*). Le développement correspond à la capacité donnée à l'utilisateur de créer des copies de ses outils avec différentes valeurs d'attributs. L'évaluation de CIS a permis d'observer que l'utilisateur est apte à optimiser l'utilisation d'une technique d'interaction et à l'adapter au contexte.

4. BESOINS ET DIMENSION D'ANALYSE

Dans cette section, nous synthétisons les services nécessaires à la manipulation synchrone de plusieurs objets. Cette synthèse est issue de nos observations et de l'état de l'art.

4.1 S1 : Gérer des ensembles d'objet

Il s'agit notamment de *rechercher* (S1.1) et *désigner* (S1.2) les objets formant un ensemble. Il est aussi nécessaire de *modifier* (S1.3) ces ensembles (ajouter, retirer des éléments). Au-delà de la simple constitution d'ensembles, des services de *structuration* (S1.4) des ensembles peuvent permettre à l'utilisateur de mieux conceptualiser et gérer de multiples objets (par exemple avec des Masters, ou des feuilles de style hiérarchiques). Enfin, il faut que l'utilisateur puisse *déterminer* (S1.5) les objets présents dans un ensemble (quels sont les objets qui composent cet ensemble ? à quel(s) ensemble(s) cet objet appartient-il ?), par exemple en les visualisant.

4.2 S2 : Gérer les actions

Gérer l'action consiste à la *spécifier* (par exemple en cliquant sur une commande d'« alignement », ou sur un item d'un menu) (S2.1), à *spécifier des paramètres* de l'action (« vertical » ou « horizontal ») (S2.2), et à *percevoir la conséquence* (S2.3). Une même action peut être spécifiée différemment. Par exemple,

Kabuki, un outil permettant de spécifier des propriétés visuelles pour des interfaces graphique complexes [21] permet d'utiliser des drag and drop, ou des appuis sur '+' et '-' pour incrémenter/décrémenter des valeurs. Ces types d'action ont des empans, et des granularités différentes, adaptées à chaque type d'utilisation : exploration de valeurs possibles (d'n'd) ou ajustement (incrément). Percevoir les conséquences permet à l'utilisateur de prendre conscience de l'effet d'une action après son déclenchement [20], voire avant son application.

4.3 S3 : Favoriser la conception exploratoire

Pour pouvoir supporter efficacement la conception exploratoire, il est important que l'utilisateur puisse avoir à sa disposition des outils lui permettant d'*essayer* (S3.1) et d'*évaluer* (S3.2) des solutions rapidement (expérimentation à court terme) sur une partie de son travail, et de *comparer* différentes versions (S3.3) [22]. Une fois satisfait de ses résultats, l'utilisateur doit appliquer les nouvelles propriétés au reste de son travail. Si le système ne supporte pas efficacement cette tâche, l'utilisateur va devoir répéter les mêmes actions pour propager ses changements. Enfin, il est préférable que la structuration puisse se faire *a posteriori* (S3.4): si la structuration est une solution au problème de la répétition, elle est peut être un frein à l'exploration si elle doit être faite a priori.

Tableau 1. Synthèse des recommandations

S1 : Gérer des ensembles d'objet	1.1 *Rechercher les objets*
	1.2 *Désigner les objets*
	1.3 *Modifier les ensembles*
	1.4 *Structurer*
	1.5 *Déterminer les objets*
S2 : Gérer les actions	2.1 *Spécifier l'action*
	2.2 *Spécifier des paramètres*
	2.3 *Percevoir la conséquence*
S3 : Favoriser la conception exploratoire	3.1 *Essayer des solutions*
	3.2 *Evaluer les solutions*
	3.3 *Comparer les solutions*
	3.4 *Structurer a posteriori*

5. CONCEPTION D'INTERACTION

Nous avons prototypé plusieurs techniques afin d'apporter plus de puissance d'action à l'utilisateur. Ces techniques ont été réalisées en suivant une démarche de conception participative : le besoin a été défini au moyen des scénarios de travail présentés au début de l'article, suivis de séances de brainstorming et de conception qui ont permis d'aboutir à ces prototypes. Nous décrivons en détail l'une d'entre elles (orientée vers le design exploratoire), et nous survolons une deuxième (orientée vers la structuration).

5.1 Vue générale de l'application

Ces différentes techniques sont implémentées dans un éditeur graphique divisé en quatre parties : une palette d'outils (sélection et création) sur la gauche, la scène au milieu, le panel d'échantillons sur le bord supérieur droit, et la boîte de propriétés sur le bord inférieur droit (Figure 4). La scène est la vue principale, où l'utilisateur peut ajouter de nouveaux objets par manipulation directe (clic et relâchement). La sélection se fait en cliquant sur un objet ou bien en traçant un rectangle pour sélectionner plusieurs objets à la fois, comme dans les éditeurs graphiques habituels. Les objets sélectionnés sont identifiables par une ombre portée. Le panel d'échantillons contient un ensemble de valeurs pour les formes, les couleurs de fond (représentées par un carré plein coloré), les couleurs de contours (représentées par un carré blanc au contour coloré), et les épaisseurs de contour (représentées par des cercles de contours d'épaisseurs différentes).

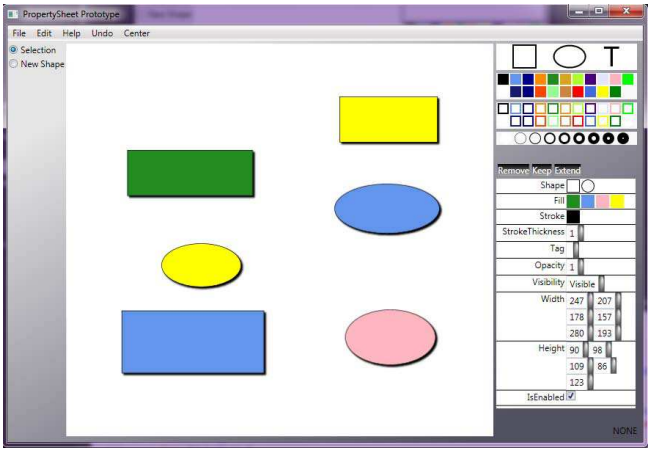

Figure 4. Vue de l'application. La scène est au centre, les échantillons en haut à d. et la boîte de propriété en bas à d.

Pour pouvoir modifier une propriété d'un objet dans la scène, l'utilisateur peut faire glisser un échantillon et le déposer sur l'objet. Un feedback est montré dès que l'objet est survolé, afin de permettre à l'utilisateur de comprendre son action, mais aussi d'évaluer le changement avant qu'il ne soit appliqué lors du drop (S2.3 : *perception*) (S3.2 : *évaluation*). Ces interactions ne sont pas vraiment nouvelles. Les sections suivantes présentent un nouveau type de boîte de propriété ainsi que de nouvelles interactions pour offrir plus de pouvoir d'action à l'utilisateur.

5.2 La boîte de propriété

Une boite de propriété offre deux services à l'utilisateur : visualiser des valeurs (avec de la « révélation progressive » [9] [20]), et les modifier. Dans les boites de propriétés classiques, seules les valeurs identiques pour une même propriété sont montrées et modifiables (voir Figure 5, gauche). L'utilisateur peut changer cette valeur pour une propriété donnée, et le système propage le changement pour tous les objets partageant la valeur (S1.2) (S2.2). En revanche, les autres propriétés, celles qui ont des valeurs différentes (multi-valuées), voient leur valeur représentée par un champ blanc. L'utilisateur ne pourra qu'assigner une même nouvelle valeur à tous les objets. Ces champs blanc limitent la visualisation (elle n'informe pas l'utilisateur des valeurs différentes) et les actions (pas de modification précise pour plusieurs objets).

Notre version de la boîte de propriété diffère en présentant toutes les valeurs pour une propriété multi-valuée (voir Figure 5, droite) au lieu de ne rien montrer. Nous nous appuyons sur la représentation des ces valeurs pour définir un ensemble d'interactions offrant certains des services identifiés plus haut : des requêtes de sélection avec des exemples graphiques, du raffinement de sélection, de la modification de propriétés pour plusieurs objets avec un meilleur degré de précision, et de la structuration de scène.

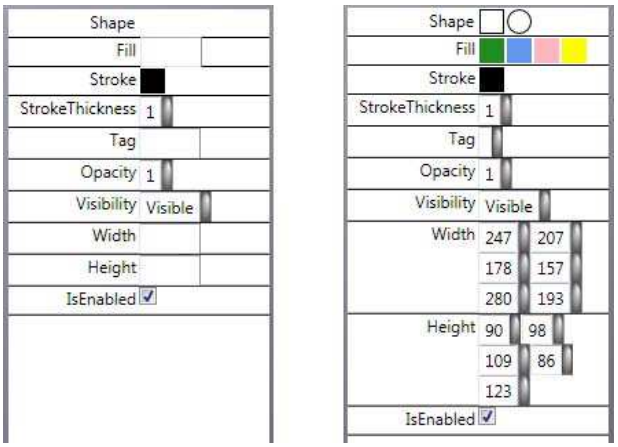

Figure 5. La sélection de l'utilisateur contient des objets de forme, couleurs et tailles différentes. Une boîte de propriété classique (à gauche) propose un champ blanc pour ces propriétés, tandis que notre boîte de propriété (à droite) présente les différentes valeurs partagées.

En fait, la représentation d'une valeur partagée dans la boîte de propriété réifie deux concepts : la valeur en soi, et l'ensemble des objets qui ont cette valeur de propriété. Similairement à l'interaction décrite pour le panneau d'échantillon, l'utilisateur peut faire glisser une représentation de valeur (considérée alors comme la valeur en soi) de la boite de propriété jusqu'à un objet de la scène pour modifier sa propriété. Par ailleurs, si une valeur partagée est numérique (par exemple la taille d'une police), l'utilisateur peut la survoler et utiliser la molette de la souris pour incrémenter ou décrémenter cette valeur avec précision (S2.1). Couplé à un feedback immédiat, ceci permet à la fois l'exploration et l'ajustement précis des propriétés, réduisant l'indirection temporelle [4] entre action et réaction (S2.3).

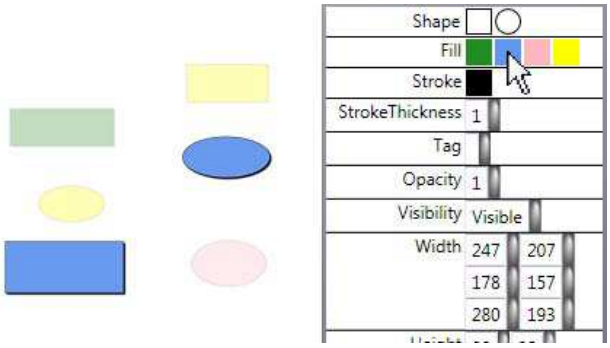

Figure 6. Le curseur survol la valeur partagée "fill: blue". N'ayant pas cette valeur partagée, le rectangle vert, le cercle rose et les deux formes jaunes sont atténués.

Survoler une représentation de valeurs (considérée ici comme réifiant [5] un ensemble d'objets) a pour effet de mettre en évidence les objets concernés en appliquant un effet de flou et une opacité réduite à tous les autres (Figure 6). Ceci facilite la compréhension des ensembles mais aussi éventuellement la détection des erreurs (S1.5). De plus, l'utilisateur peut faire glisser un échantillon du panel d'échantillons directement sur la représentation d'une valeur partagée (considérée alors comme un ensemble d'objets) pour modifier en une seule fois une propriété de tous les objets de l'ensemble (S2.1) (S2.2) (Figure 7). L'utilisateur peut aussi faire glisser une représentation d'une valeur comme s'il s'agissait d'un échantillon et la déposer sur une autre valeur partagée (Figure 8). Le feedback immédiat pendant l'interaction aide l'utilisateur à évaluer le résultat de ses actions (exploration), et les annuler avant application. Ces interactions peuvent être considérées comme des requêtes qui aident à la définition d'ensemble d'objets, avec un groupement implicite basé sur les propriétés graphiques (S3.4 : *structuration a posteriori*).

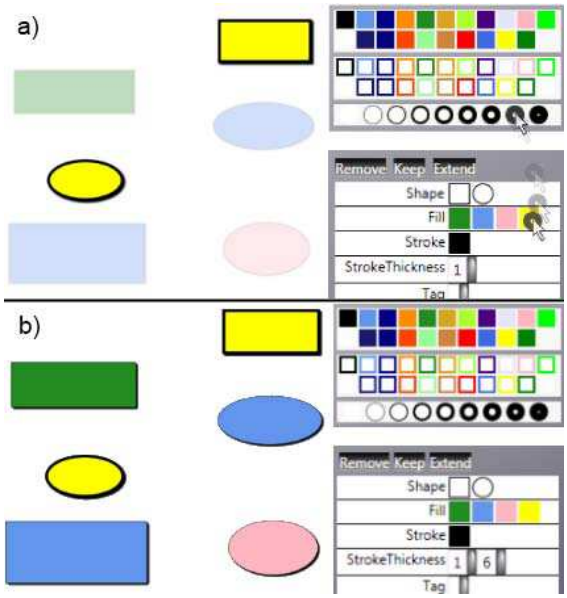

Figure 7. a) L'utilisateur drag l'échantillon "thickness: 6pt" sur la valeur partagée "fill: yellow". Un feedback immédiat donne aux objets jaunes une épaisseur de 6pt. b) L'utilisateur a lâché l'échantillon, la modification est appliquée.

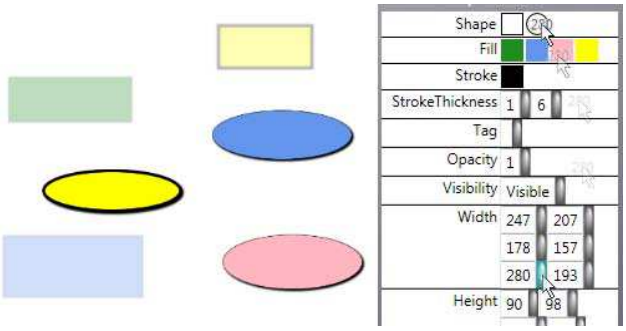

Figure 8. L'utilisateur glisse la valeur partagée "width: 280" et la lâche sur la valeur partagée "shape: circle". Tous les cercles de la sélection ont désormais une largeur mise à 280.

Pour sélectionner des objets, l'utilisateur peut cliquer sur eux, ou tracer un rectangle de sélection. Pour raffiner cette sélection (S1.3), l'utilisateur peut utiliser trois méta-instruments (c'est-à-dire des instruments contrôlant des instruments, ici la sélection) : *Remover*, *Keeper* et *Extender*. L'interaction consiste en un drag and drop d'une représentation d'un instrument sur une valeur partagée. *Remover* retire de la sélection tous les objets qui possèdent la valeur partagée (Figure 9). *Keeper* garde dans la sélection les objets qui ont cette valeur partagée et enlève tous les autres. *Extender* ajoute à la sélection tous les objets qui ne sont actuellement pas sélectionnés mais possédant cette valeur partagée. Ces interactions étendent l'ensemble des requêtes basées sur l'exemple expliqué auparavant.

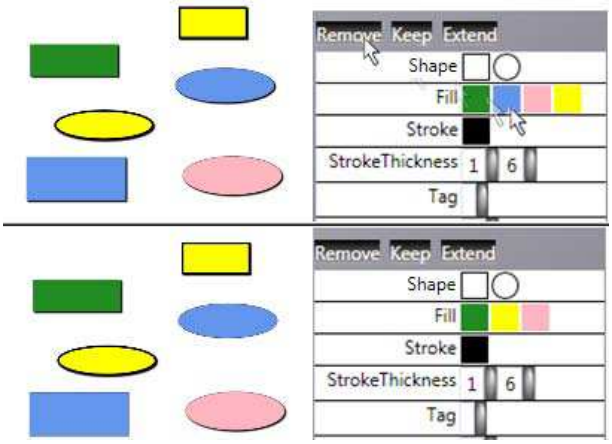

Figure 9. L'utilisateur drag l'instrument Remove et le lâche sur la valeur partagée "fill: blue". Tous les objets bleus sont retirés de la sélection.

5.3 Aide à la structuration

Nous avons adapté le mécanisme de restructuration d'héritage de prototypes à l'édition graphique. L'utilisateur sélectionne un ensemble d'objet et déclenche une commande de hiérarchisation automatique, basée sur l'algorithme de [17] (S1.4 : *structuration*). Le système permet d'afficher la hiérarchie obtenue sous forme d'arbre (Figure 10). Ainsi, il devient possible d'agir directement sur un prototype (un nœud intermédiaire de l'arbre) avec les interactions déjà présentées, afin de modifier en une seule interaction tous les objets qui en dépendent. Afin de comprendre quels sont les objets qui dépendent d'un prototype, le survol d'un objet de la scène met en valeur ses prototypes (Figure 11). De plus, survoler un prototype va mettre en évidence les objets de la scène qui lui sont liés, en réduisant l'opacité et en appliquant un effet de blur à tous les autres, de façon similaire à l'interaction décrite pour la boite de propriété. L'intérêt de cette technique d'interaction est de permettre de la structuration a posteriori (S3.4), l'utilisateur pouvant faire de l'exploration sans avoir à réfléchir a priori à sa structure.

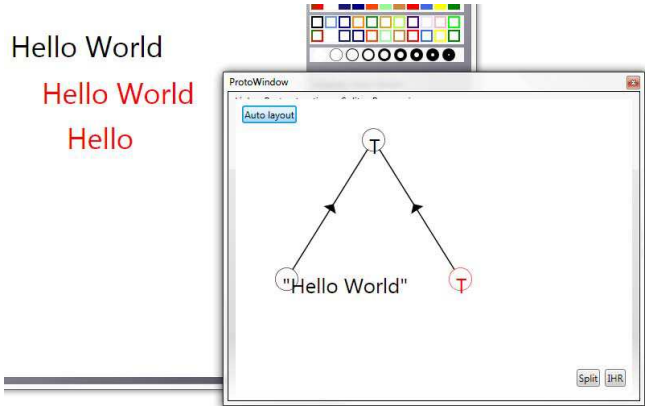

Figure 10. Après restructuration, la vue de la hiérarchie montre les propriétés communes aux éléments : couleur de texte rouge et contenu du texte pour deux d'entre eux.

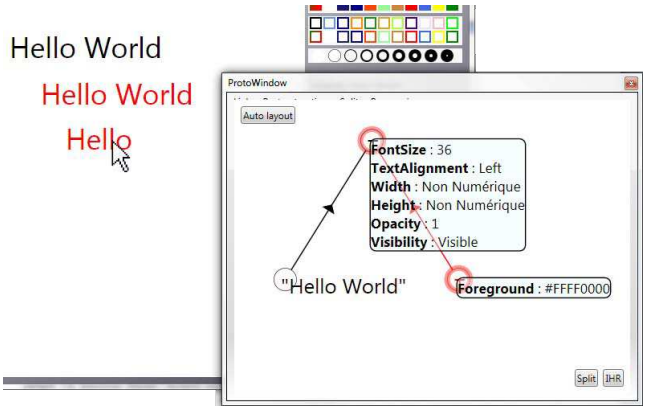

Figure 11. En survolant le texte Hello, la vue de la hiérarchie des prototypes met en valeur les nœuds contenant les propriétés du texte.

6. CONCLUSION

Dans cet article nous avons identifié un ensemble de services nécessaires à l'édition synchrone de plusieurs objets. Nous avons présenté des exemples d'outils interactifs : une nouvelle boîte de propriété, dont l'objectif est d'offrir un plus grand pouvoir d'action que les boîtes de propriétés classiques, et un mécanisme de structuration automatique de propriétés communes. Ce travail peut inspirer les designers pour la conception d'interactions plus cohérentes et plus puissantes. Nous envisageons pour la suite d'améliorer les différentes techniques que nous avons développées. Par exemple, la représentation actuelle de la hiérarchie de prototype ne passe pas à l'échelle, et nécessite des travaux complémentaires. Nous évaluerons ensuite l'efficacité de ces techniques.

7. REMERCIEMENTS

Nous tenons à remercier les membres du LII et les utilisateurs impliqués dans la réalisation de ce travail, pour le temps qu'ils nous ont accordés lors des interviews, leur participation aux séances de brainstorming, leurs retours sur les techniques d'interaction, et les discussions qu'elles ont engendrées.

8. BIBLIOGRAPHIE

[1] Appert, C., Beaudouin-Lafon, M. SwingStates: adding state machines to the swing toolkit. *In Proc. of UIST '06*. ACM, 2006, pp. 319-322.

[2] Appert, C, Beaudouin-Lafon, M, Mackay, W. E. Context matters: Evaluating Interaction Techniques with the CIS Model. *In Proc. of HCI'04*, p 279-295. Springer Verlag, 2004.

[3] Bastien, J.M.C, Scapin, D.L. Critères Ergonomiques pour l'Évaluation d'Interfaces Utilisateurs. *Tech Report 156, May 1993*.

[4] Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. *In Proc. CHI'00*. (2000), ACM, 446-453.

[5] Beaudouin-Lafon, M. and Mackay,AW. E. Reification, polymorphism and reuse: three principles for designing visual interfaces. *In Proc. AVI'00*. ACM, 2000, pp. 102-109.

[6] Dourish, P., Edwards W.K., LaMarca, A. and Salisbury, M. 1999. Presto: an experimental architecture for fluid

interactive document spaces. *ACM Trans. Comput.-Hum. Interact*. 6, 2 (June 1999), 133-161.

[7] Green, T.R.G., Cognitive dimensions of notations, in *People and computers v*, 1989, Cambridge University Press, 443-460.

[8] Green, T.R.G, and Blackwell, A. Cognitive dimensions of information artifacts: a tutorial. (*V1.2 October 1998*). 1998.

[9] Johnson J.A., Roberts T.L., Verplank W., Smith D.C., Irby C.H., Beard M., and Mackey K. The Xerox Star: A retrospective. IEEE Computer, 22(9):11–29, 1989.

[10] Kurlander, D. Reducing Repetition in Graphical Editing. *Proc. of HCI International '93*. 1993.

[11] Kurlander, D, Bier, E.A. Graphical Search and Re-place. In *Proc. of* ACM SIGGRAPH '88, 113-120.

[12] Kurlander, D., Feiner, S. A History-Based Macro By Example System. In *Proc. of* UIST '92. ACM, 99-106.

[13] Lieberman, H. 1986. Using prototypical objects to implement shared behavior in object-oriented systems. In *Proc. of OOPLSA '86*. ACM, 214-223.

[14] Lieberman, H. Your Wish is my command: Programming by example. Morgan Kaufmann, 2001.

[15] Mackay W.E. Which interaction technique works when?: floating palettes, marking menus and tool-glasses support different task strategies. In *Proc. of* AVI '02. ACM, 203-208.

[16] Maloney, J.H. and Smith, R.B. 1995. Directness and liveness in the morphic user interface construction environment. In *Proc*. UIST '95. ACM, 21-28.

[17] Moore, I. 1996. Automatic inheritance hierarchy restructuring and method refactoring. In *Proc. of OOPSLA '96*. ACM, 235-250.

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

[19] Sutherland I.E. 1963. Sketchpad: a man-machine graphical communication system. In *Proc.* AFIPS '63. ACM, 329-346.

[20] Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. IEEE Computer 16, 8, 57–69.

[21] Tabart G., Conversy S., Vinot J.L., and Athènes S. 2009. Outils d'aide à la conception de rendus graphiques. In *Proc. of* IHM '09. ACM, 303-312.

[22] Terry, M. and Mynatt E.D. Recognizing creative needs in user interface design. *Proc. of Creativity & Cognition*. ACM, pp. 38-44, 2002.

[23] Terry M, Mynatt E.D, Nakakoji K, and Yamamoto Y. 2004. Variation in element and action: supporting simultaneous development of alternative solutions. In *Proc. of CHI '04*. ACM, 711-718.

[24] Terry, M. and Mynatt, E. D. Side views: persistent, on-demand previews for open-ended tasks. In *Proc. of UIST 2002*.ACM, 2002, pp. 71-80.

[25] Ungar, D, Smith R, B. SELF: The Power of Simplicity. In *proc. of OOPSLA '87*. ACM, 227-242.