HAL
open science

Modélisation, Évaluation et Génération de Techniques d'Interaction

Caroline Appert

To cite this version:

Caroline Appert. Modélisation, Évaluation et Génération de Techniques d'Interaction. Informatique [cs]. Université Paris Sud - Paris XI, 2007. Français. NNT : . tel-00539139

HAL Id: tel-00539139

https://theses.hal.science/tel-00539139

Submitted on 24 Nov 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ PARIS SUD

N° d'ordre

| 8 | 6 | 6 | 1 |

THÈSE

pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ PARIS SUD

Discipline : Informatique

préparée au Laboratoire de Recherche en Informatique

dans le cadre de **l'Ecole Doctorale d'Informatique de l'Université Paris-Sud**

présentée et soutenue publiquement

par

Caroline Appert

le 21 mai 2007

Titre :

Modélisation, Évaluation et Génération de Techniques d'Interaction

Directeur de thèse :
Michel Beaudouin-Lafon

JURY

Mme. Christine Froidevaux,	Président
Mme. Joëlle Coutaz,	Rapporteur
M. Philip Gray,	Rapporteur
M. Michel Beaudouin-Lafon,	Directeur de thèse
M. Éric Lecolinet,	Examinateur

Remerciements

Je remercie en premier lieu mon directeur de thèse, Michel, qui a toujours su me soutenir aux moments appropriés. D'une part, il est un excellent chercheur et professeur que j'admire et il est évident que je n'aurais pu avoir un meilleur modèle. D'autre part, son écoute et ses conseils qui m'ont permis de surmonter les moments de doute font de lui un homme que j'apprécie sincèrement.

Je remercie les membres de mon jury pour avoir lu attentivement mes travaux, pour s'être déplacés parfois de loin pour m'écouter et pour m'avoir fait part de remarques constructives dans leurs rapports et lors de ma soutenance de thèse.

Je remercie bien évidemment tous les membres d'in|situ|. Je remercie tout particulièrement Emmanuel, Wendy et Jean-Daniel avec lesquels j'ai collaboré au cours de ces années de thèse et qui m'ont chacun transmis des éléments précieux et variés de leur savoir respectif. Double merci à Emmanuel pour être un co-bureau cool (!) et un ami d'une grande qualité. Merci à tous ceux qui étaient ou sont encore à in|situ| avec lesquels j'ai partagé des moments humains et ai beaucoup ri, je sais qu'ils se reconnaîtront.

Merci aux étudiants qui ont utilisé la boîte à outils SwingStates et qui m'ont permis de l'améliorer.

Mercis à ma mère, Adé, Jérémy et Sophie. Ils m'ont fait confiance lorsque j'ai décidé de me lancer dans la réalisation d'une thèse et ont été des épaules solides pour moi pendant ces années.

Mille mercis à Benoît, il est un magnifique compagnon de vie. Sa compréhension, sa patience et sa gentillesse ont été la condition sine qua none de la réalisation de cette thèse mais également de mon bien-être depuis de nombreuses années. Enfin, merci, merci et merci à tous mes amis avec une pensée particulièrement tendre pour Chloé, Sylvain et Jérémie, Véra, Djordje et Ugo, Emmanuelle alias Moineau, Julia et Delphine. Sans toutes ces formidables personnes, la vie n'aurait tout simplement pas de sens.

Table des matières

1 Introduction **11**
1.1 Constat : Des techniques plus efficaces... non utilisées. 13
1.2 Les causes d'un tel constat 14
1.2.1 Comment sélectionner des techniques ? 14
1.2.2 Comment implémenter l'interaction avancée ? 15
1.3 Solution proposée 16
1.4 Plan 17
1.5 Champ de recherche 17

2 Modéliser les techniques et leur contexte d'utilisation avec CIS **19**
2.1 Du périphérique d'entrée à la tâche : Décrire l'interaction 21
2.1.1 Au niveau du périphérique d'entrée 21
2.1.2 Les modèles "centrés objet" 24
2.1.3 Au niveau de la tâche 27
2.1.4 CIS : au niveau de la technique et du contexte 35
2.2 Le modèle CIS 37
2.2.1 Décrire une technique d'interaction 37
2.2.2 Opérationnaliser un contexte d'utilisation 43
2.3 Conclusion 46

3 Explorer l'espace de conception avec SimCIS et CIS **49**
3.1 SimCIS : prédire l'efficacité d'une technique en contexte 51
3.1.1 Les lois empiriques 51
3.1.2 Complexité d'une technique d'interaction 55
3.2 Explorer l'espace de conception : Optimiser au niveau de l'action 61
3.2.1 Agir au niveau de l'action grâce aux lois empiriques 61
3.2.2 Optimiser un arc : OrthoZoom 63
3.2.3 Optimiser un nœud : ControlTree 74
3.3 Explorer l'espace de conception : Travailler au niveau de la structure de graphe 82
3.3.1 La génération automatique de techniques d'interaction 83

3.3.2 Concevoir une nouvelle structure CIS . . . 84
3.4 Conclusion . . . 89

4 Validation de CIS et Extensions 91
4.1 Valider CIS . . . 93
4.1.1 Validation du modèle : L'expérimentation . . . 93
4.1.2 Validation du modèle : Les résultats . . . 98
4.2 Raffiner CIS . . . 103
4.3 Étendre CIS . . . 106
4.4 Un modèle de choix pour l'interaction multi-échelles . . . 108
4.4.1 Travaux antérieurs . . . 108
4.4.2 Opérationalisation de la tâche de recherche multi-échelle . . . 109
4.4.3 Les techniques d'interaction multi-échelles évaluées . . . 110
4.4.4 Expérimentation . . . 112
4.4.5 Vers un modèle de recherche plus général . . . 118
4.5 Conclusion . . . 121

5 Implémenter les techniques d'interaction avec SwingStates 123
5.1 Les boîtes à outils existantes et l'approche de SwingStates . . . 125
5.2 Les machines à états et le langage Java . . . 127
5.2.1 Machines à états vs. Écouteurs d'événements . . . 127
5.2.2 Syntaxe des machines à états avec SwingStates . . . 129
5.3 Un canvas à personnaliser . . . 132
5.3.1 Le modèle graphique . . . 133
5.3.2 Les tags . . . 136
5.3.3 Les machines à états pour le canvas de SwingStates . . . 137
5.3.4 Les animations . . . 139
5.4 Combiner les machines à états . . . 142
5.4.1 Combiner en séquence pour communiquer . . . 142
5.4.2 Combiner en parallèle pour factoriser . . . 144
5.5 Redéfinir l'interaction de composants graphiques existants . . . 146
5.6 Évaluer une boîte à outils, un problème complexe . . . 149
5.6.1 Les principes de conception . . . 149
5.6.2 L'approche par benchmarks . . . 150
5.7 Conclusion . . . 153

6 Mener des expérimentations contrôlées avec TouchStone 155
6.1 L'architecture de TouchStone . . . 157
6.2 Concevoir des expérimentations contrôlées . . . 158
6.3 Exécuter des expérimentations contrôlées . . . 166
6.3.1 Le module d'expérimentation . . . 167
6.3.2 Définir des composants d'expérimentation . . . 173
6.3.3 La gestion de l'entrée généralisée . . . 180
6.4 TouchStone : Validation et exemples d'utilisation . . . 182

6.4.1 Évaluation de la plateforme de conception 182
6.4.2 TouchStone et CIS 183
6.5 Conclusion 189

7 Conclusion et Perspectives 191
7.1 Les contributions 193
7.1.1 Complexity of Interaction Sequences 193
7.1.2 SwingStates 194
7.1.3 TouchStone 194
7.1.4 Ce qu'il manque... 195
7.2 SwingStates et l'entrée généralisée 195
7.3 CIS et l'interaction gestuelle 196
7.4 À plus long terme 198

Bibliographie 199

8 Annexes 211
8.1 Code complet de la balle rebondissante avec SwingStates 213
8.2 Code complet de l'expérimentation de Hick-Hyman 214
8.2.1 Le bloc `ReactionBlock` 214
8.2.2 L'intertitre `StartButton` 215
8.2.3 Le critère `PressOnTag` 216

Table des figures

1.1 Exemples de technique d'interaction 13

2.1 Représentation de l'espace de conception des périphériques de Card et al. 22
2.2 modèle 3-state : un stylet et une souris à deux boutons 23
2.3 Configuration ICON 23
2.4 IOG pour spécifier un interrupteur sécurisé 24
2.5 L'environnement Petshop 25
2.6 Le modèle de l'interaction instrumentale 26
2.7 Description UAN de l'interaction pour déplacer un icone 28
2.8 Architecture cognitive 30
2.9 Les niveaux hiérarchiques d'une activité 33
2.10 Exemples d'activités, actions et opérateurs 33
2.11 Les modèles candidats par niveau d'abstraction 35
2.12 Une acquisition CIS 38
2.13 Une validation CIS 38
2.14 Le graphe d'interaction d'une palette fixe 39
2.15 Ordre des actions dans une technique 41
2.16 Parallélisme des actions dans une technique 41
2.17 Propriétés de factorisation et de fusion 42
2.18 Propriété de développement 43
2.19 Deux séquences d'interaction 45
2.20 Le niveau d'abstraction de CIS 46
2.21 Sous-ensemble de l'interaction décrite avec CIS 47

3.1 Pointage d'une cible 52
3.2 Suivi de chemin et interaction dans un menu 53
3.3 Pointage VS. Franchissement 54
3.4 CrossY : Un éditeur graphique basé sur le franchissement 55
3.5 Deux séquences d'interaction 56
3.6 SimCIS 56
3.7 SimCIS : calcul des complexités 57

3.8 Complexités en temps et en actions . . . 58
3.9 Object Movement Diagrams . . . 59
3.10 La tâche “connect the dots” . . . 60
3.11 Diminuer D . . . 62
3.12 Principe d’OrthoZoom . . . 64
3.13 Utiliser la dimension orthogonale pour contrôler le facteur de zoom . . . 66
3.14 Valeurs qui ne peuvent être atteintes en restant dans les bornes . . . 67
3.15 La tâche expérimentale . . . 69
3.16 Facteur de zoom et direction de déplacement . . . 69
3.17 Analyse du temps de mouvement . . . 70
3.18 Analyse du temps de mouvement en fonction d’ID . . . 71
3.19 Apprentissage des deux techniques . . . 71
3.20 Analyse des erreurs . . . 72
3.21 Multi-Scale Table Of Contents (MSTOC) . . . 73
3.22 Structure CIS pour la sélection d’un nœud au niveau suivant . . . 76
3.23 L’affichage nœud-lien dynamique de ControlTree . . . 76
3.24 Une interaction typique avec ControlTree . . . 78
3.25 Zones d’interaction autour du nœud courant . . . 79
3.26 Degré et précision angulaire dans le cas d’un seuil fixe . . . 80
3.27 Stratégies simplistes pour faciliter une sélection . . . 80
3.28 Stratégie de ControlTree pour faciliter une sélection . . . 81
3.29 Entrées/Sorties de l’outil Toto . . . 83
3.30 Exemple d’interface adaptative . . . 84
3.31 L’interaction iconique classique pour effacer un fichier . . . 85
3.32 Changer l’ordre et conférer de la persistance . . . 86
3.33 Développer ou fusionner . . . 87
3.34 Un graphe d’interaction combinant parallèlisme et persistance . . . 88
3.35 Propriétés des trois techniques évaluées . . . 88
3.36 Complexité en temps de trois techniques dans deux contextes . . . 88
3.37 Object Movement Diagrams pour la palette bi-manuelle . . . 89

4.1 La tâche expérimentale . . . 95
4.2 Les quatre types de séquences d’interaction . . . 96
4.3 Les prédictions de SimCIS . . . 97
4.4 Les séquences libres . . . 99
4.5 Les observations empiriques . . . 100
4.6 Temps d’exécution moyen d’une séquence . . . 100
4.7 Influence de la longueur de la séquence sur le temps d’exécution . . . 101
4.8 Séquences libres vs. séquences imposées . . . 102
4.9 Techniques et changements d’outil . . . 103
4.10 Déterminer les coefficients de la loi de Hick . . . 104
4.11 Les prédictions de SimCIS raffinées . . . 105
4.12 Les familles d’interaction et leurs profils CIS . . . 107

4.13 Représentation des techniques d'interaction multi-échelle dans des diagrammes espace-échelle . . . 111
4.14 Diagramme espace-échelle de la scène utilisée lors de l'expérimentation . . . 112
4.15 Storyboard de la tâche de recherche multi-échelle . . . 113
4.16 Dévoiler un objet avec les différentes technique . . . 115
4.17 Stratégie de contre-balancement . . . 116
4.18 Régression linéaire pour les quatre techniques . . . 118
4.19 Temps moyen par technique et en fonction du rang . . . 118
4.20 Rechercher avec une lentille de distortion . . . 119
4.21 Explorer une région éparse ou dense . . . 120

5.1 Représentation graphique d'une machine à états pour le drag'n drop . . . 128
5.2 Machines à états VS. Écouteurs d'événements . . . 130
5.3 Les classes de machines à états dans SwingStates . . . 132
5.4 Les classes d'objets graphiques dans SwingStates . . . 133
5.5 Les transformations dans le canvas SwingStates . . . 134
5.6 Un marking menu réalisé avec SwingStates . . . 135
5.7 Une animation avec SwingStates . . . 140
5.8 Utiliser un tag pour des animations . . . 141
5.9 Construction en séquence . . . 144
5.10 Construction en parallèle . . . 144
5.11 Visualiser les machines à états . . . 145
5.12 Approche hiérarchique vs. Approche SwingStates . . . 146
5.13 Des cases à cocher franchissables . . . 148
5.14 Une entrée numérique augmentée d'une "interaction joystick" . . . 148
5.15 Un *pie menu* pour colorer des widgets Swing . . . 149
5.16 Benchmark de techniques utilisé . . . 151
5.17 Exemples de projets réalisés par les étudiants . . . 152

6.1 Architecture de TouchStone . . . 159
6.2 Résumé des décisions lors de la conception d'une expérimentation . . . 160
6.3 Plateforme de conception : Spécifier les facteurs . . . 161
6.4 Plateforme de conception : Structure des blocs . . . 162
6.5 Éxécution générique de l'expérimentation . . . 163
6.6 Plateforme de conception : Contrebalancement . . . 164
6.7 Entrées/Sorties du module d'expérimentation . . . 167
6.8 Gestion des évènements de l'expérimentation . . . 168
6.9 Interface pour lancer une expérimentation . . . 170
6.10 Exécution de l'expérimentation . . . 171
6.11 Un exemple de fichiers de logs au niveau de la tâche . . . 172
6.12 Un exemple de fichiers de logs au niveau de la cinématique . . . 172
6.13 Interface pour lancer une expérimentation . . . 173
6.14 Mécanisme de fabriques et doclet de TouchStone . . . 175
6.15 Interface pour gérer les axes d'entrée . . . 181

6.16 Composants de l'expérimentation de Hick-Hyman 185
6.17 De la plateforme d'exécution à la plateforme de conception 187
6.18 PZ vs. PZB vs OZ. vs. SDAZ 189

Introduction

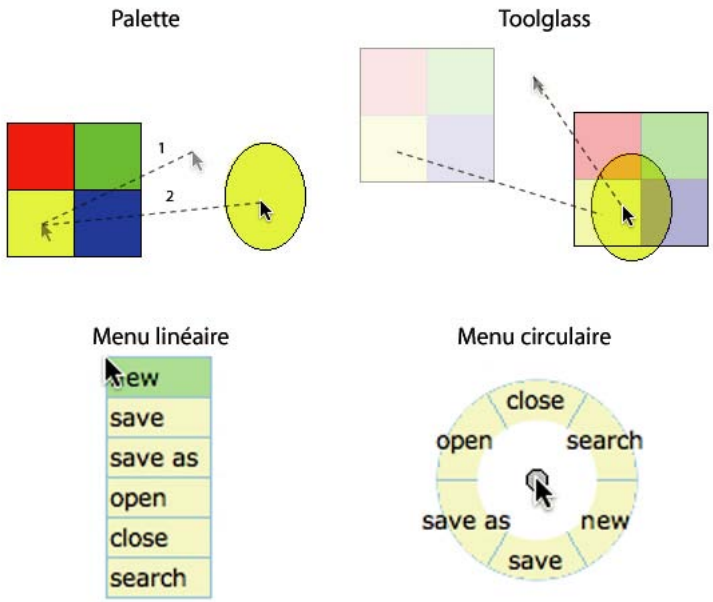


FIG. 1.1 : Exemples de technique d'interaction. À gauche : ce qui est utilisé, à droite : ce qui existe.

1.1 Constat : Des techniques plus efficaces... non utilisées.

La recherche en Interaction Homme-Machine (IHM) produit continuellement de nouvelles techniques d'interaction pour améliorer l'utilisabilité des applications graphiques. Cependant, les produits industriels n'en tirent que très rarement profit, leurs interfaces restant des compositions de "widgets" classiques. Un widget est une représentation standardisée des objets manipulés par l'utilisateur à l'écran. Ce que nous appelons widgets classiques est constitué de l'ensemble des composants graphiques qui sont utilisés pour construire des interfaces dites WIMP (Windows, Icons, Menus and Pointing). Boutons, menus ou barres de défilement sont des exemples de widgets classiques. Alors que les chercheurs conçoivent régulièrement des nouvelles techniques plus efficaces et plus adaptées, il est frustrant à la fois pour eux comme pour les utilisateurs de constater que ces résultats ne sont pas utilisés dans nos produits de tous les jours. De fait, l'état des interfaces graphiques a peu évolué depuis 25 ans et l'interface du Xerox Star [161].

À titre d'exemple, la figure 1.1 illustre comment deux widgets connus de tous les utilisateurs, la palette d'outils et le menu linéaire contextuel, pourraient être respectivement remplacés par la *toolglass* [30] et le *pie menu* [42], deux techniques plus récentes et plus efficaces. Contrairement à la palette qui impose deux sélections en séquence de l'outil puis de l'objet auquel appliquer cet outil, la toolglass peut-être déplacée en même temps que le curseur grâce à un second périphérique d'entrée contrôlé par la main non dominante. L'utilisateur amène conjointement curseur et outil au-dessus de l'objet de son choix pour ne cliquer qu'une fois sur l'objet au travers de l'outil semi-transparent. La toolglass pouvant être déplacée en même temps que le curseur, l'utilisateur peut garder ses outils à "portée de curseur". De

même, contrairement au menu linéaire pour lequel le temps de sélection d'un item varie selon la position de cet item dans le menu, le menu circulaire (ou *pie menu*) permet de sélectionner n'importe quel item en temps constant en spécifiant une direction. Alors que l'utilisation de ces widgets avancés pourraient améliorer l'utilisabilité des interfaces, il est très rare de les rencontrer dans nos applications de tous les jours. L'objectif de cette thèse est d'identifier les causes d'un tel état de fait et de proposer des solutions pour faciliter l'introduction de techniques performantes et adaptées au contexte d'utilisation. Nous proposons une *approche générative* de conception, c'est-à-dire la définition d'outils et de guides pour explorer un espace de conception, créer des nouveaux éléments dans cet espace et de les comparer afin de faire un choix informé.

1.2 Les causes d'un tel constat

Ici, nous tentons de comprendre pourquoi les techniques d'interaction dites "avancées" ne s'imposent pas dans les interfaces commerciales. Pour cela, nous nous mettons à la place du concepteur d'interfaces qui doit choisir l'ensemble des techniques qui constitueront une interface graphique (phase de conception) puis dans la peau du développeur d'interface qui doit implémenter ces techniques (phase de développement). Nous ne nous préoccuperons pas des problèmes d'ordre commercial et juridique comme, par exemple, des aspects liés à la propriété intellectuelle qui sont cependant présents dans le domaine des interfaces homme-machine (par exemple, les toolglasses, décrites ci-dessus, sont brevetées par Xerox).

1.2.1 Comment sélectionner des techniques ?

Notre concepteur d'interfaces est confronté au problème de choisir et composer un ensemble de techniques pour concevoir l'interface d'une application ayant des fonctionnalités données pour un ensemble d'utilisateurs qui ont bien évidemment eux aussi leurs caractéristiques propres.

Face à une large littérature scientifique, notre concepteur peut prendre son courage à deux mains et entreprendre de trouver quelles techniques lui seraient favorables pour son problème. Il se retrouve face à une grande quantité d'articles dont la plupart rapporte les résultats d'expérimentations contrôlées. Or une expérimentation contrôlée s'effectue en laboratoire sur une tâche bien spécifique et compare les performances de différentes techniques pour exécuter cette tâche. Le premier problème que notre concepteur doit donc affronter est le problème bien connu de la validité des résultats issus de ce type d'évaluation : comment les interpréter pour un problème réel ? Passent-ils à l'échelle ? Le second problème est celui de l'échantillonnage utilisé pour choisir le sous-ensemble de techniques testées dans l'expérimentation. Comment faire pour comparer deux techniques qui ont été évaluées dans deux études différentes ? Comment unifier les résultats s'il n'y a pas de technique à l'intersection des deux études ? Si les tâches ou les conditions expérimentales sont différentes ?

Très courageux, notre concepteur décide de mener sa propre expérience pour comparer les techniques qu'il juge prometteuses dans un cadre expérimental correspondant à son problème réel. Lourde tâche... Les phases qui constituent le déroulement d'une telle expérience, c'est-à-dire conception du cadre expérimental, réalisation de l'expérience et interprétation des résultats obtenus, requièrent chacune son lot de compétence et de temps. La conception inclut, par exemple, l'opérationnalisation de la tâche réelle que

l'on veut étudier, c'est-à-dire sa définition en termes de variables d'intérêt (ou facteurs ou variables indépendantes) sur lesquelles l'expérimentateur peut agir dans le but de collectionner des mesures qui doivent elles aussi être identifiées. C'est un exercice difficile qui requiert des connaissances et de la pratique : il est courant de constater au moment de l'analyse des résultats que l'on a mesuré les effets de facteurs non contrôlés ou que les effets ne sont pas significatifs et donc qu'il est impossible de conclure quoi que ce soit. Ce genre de déception est particulièrement fréquente lorsque le cadre expérimental se veut le plus "réaliste" possible. La réalisation de l'expérimentation en elle-même requiert un environnement pour conduire cette expérience (un lieu, un expérimentateur, etc.) et des participants, tout ceci se traduisant par une quantité de temps et d'énergie non négligeables. Enfin, n'oublions pas que des connaissances statistiques sont indispensables afin de "faire parler" les chiffres lors de la phase d'analyse des résultats.

Il reste une alternative qui pourrait permettre à notre concepteur d'interface d'économiser du temps sur l'évaluation empirique en faisant une comparaison théorique des techniques candidates. Il dispose de modèles descriptifs dont certains permettent également de prédire les performances d'une technique. Ces modèles ne visent pas à remplacer les études auprès des utilisateurs mais à gagner du temps sur celles-ci en éliminant les solutions prédites comme non prometteuses. Malheureusement, nous verrons dans ce manuscrit que l'utilisation des modèles existants pose là encore des problèmes. Ces modèles sont assez nombreux mais leurs niveaux d'abstraction sont souvent trop faibles ou trop élevés pour répondre à la simple question "Laquelle de ces techniques est la plus efficace dans ce contexte d'utilisation ?". De plus, nous verrons que l'utilisation de chacun de ces modèles requiert l'apprentissage du modèle et le temps nécessaire à réaliser des descriptions et prédictions.

1.2.2 Comment implémenter l'interaction avancée ?

La phase de conception n'est donc pas une tâche facile car il est difficile d'appliquer les résultats rapportés dans la littérature dans des problèmes d'utilisation réels et les outils actuels d'évaluation restent difficiles à utiliser. Mais les problèmes ne s'arrêtent pas là... Supposons maintenant que notre concepteur a finalement choisi un ensemble de techniques appropriées. Maintenant, c'est à notre développeur de jouer ! Notre concepteur ayant fait un bon travail, il y a de grandes chances que notre développeur ait à réaliser une interface incluant des techniques d'interaction "avancée". Il faut donc s'intéresser aux outils qu'il a à sa disposition pour implémenter ce type de techniques.

Les boîtes à outils qui sont largement utilisées pour développer des applications graphiques, comme Java Swing [114], Gtk [113] ou Qt [61] ont été conçues afin que le développeur final puisse facilement construire une interface en assemblant des widgets. L'ensemble des widgets qu'elles contiennent sont les fameux widgets WIMP que nous avons déjà évoqués et les paradigmes de programmation qu'elles proposent ne sont pas appropriés pour favoriser l'implémentation de nouvelles techniques ayant un rendu graphique différent ou, plus encore, un contrôle différent. Sur l'exemple du pie menu, il faut pouvoir aisément définir l'apparence d'un menu circulaire ainsi que le fait de sélectionner un item par une direction. Les markings menus [105], une version plus récente des menus circulaires ont un contrôle encore plus compliqué à programmer : lorsque l'utilisateur presse le bouton de la souris pour invoquer le menu, celui-ci ne s'affiche que si l'utilisateur n'a pas sélectionné d'item au bout de 300 ms. Ainsi, l'utilisateur expert peut spécifier une direction bien connue sans être perturbé par l'affichage du menu. Pour implémenter

un tel contrôle, le développeur utilisera typiquement une approche par écouteurs d'évènements indépendants qui communiquent entre eux par des variables globales. En effet, les paradigmes de programmation de ces boîtes à outils consistent à créer un écouteur par type d'évènement (*press*, *drag*, *release*, *time out*, etc.) pour spécifier le traitement à exécuter quand ce type d'évènement se produit (c'est-à-dire le *callback*). Cette approche mène très souvent à des programmes qui ressemblent à des spaghettis de callbacks [128] difficiles à relire, à corriger et à réutiliser.

Comme nous le détaillerons au chapitre 5.1, de nombreuses boîtes à outils expérimentales ont été développées pour faciliter la programmation de l'interaction avancée. Cependant, ces boîtes à outils sont très peu utilisées en dehors de leurs laboratoires originels et les applications que nous rencontrons tous les jours restent toujours développées avec les mêmes boîtes à outils de production. Les développeurs sont probablement trop habitués à leurs outils habituels et ne prennent pas le temps d'apprendre de nouvelles boîtes à outils.

1.3 Solution proposée

Le problème principal est donc le manque d'outils appropriés pour faciliter la génération et la mise en oeuvre de techniques d'interaction avancées tant par la difficulté de mesurer l'efficacité de celles-ci que pour les programmer. Ce que nous proposons dans cette thèse est une *approche générative*, c'est-à-dire permettre non seulement d'explorer un espace de conception des techniques d'interaction pour décrire, comparer l'existant et envisager des alternatives dans les premières phases de conception, mais aussi des outils pour implémenter rapidement les interactions choisies puis pour les tester de façon plus formelle et contrôlée. Contrairement aux recherches sur la génération automatique d'interfaces (cf. section 3.3.1) qui produisent des outils qui peuvent amener à des compositions intéressantes de widgets classiques, une *approche générative* sert à aider les concepteurs, évaluateurs et développeurs d'interface à explorer un espace de conception dans le cadre d'un contexte d'utilisation réel. À cette fin, nous avons conçu et développé un ensemble d'outils pour les accompagner durant le processus qui consiste à mettre en oeuvre des techniques innovantes pour un usage réel.

Tout d'abord, nous proposons un modèle, *Complexity of Interaction Sequences (CIS),* qui définit un espace de conception utile pour décrire les techniques d'interaction existantes et pour en envisager de nouvelles. Le niveau d'abstraction de CIS est celui de la technique d'interaction. Ce niveau n'est pas trop faible afin de ne pas rentrer dans les détails lors de la conception tout en appréciant les propriétés morphologiques des techniques pour les comparer qualitativement. Aussi, ce niveau n'est pas trop élevé et prend en compte une opérationnalisation du contexte d'utilisation pour mesurer l'efficacité. À cet effet, le modèle CIS est accompagné de *SimCIS*, un outil prenant en entrée un contexte opérationnalisé et une technique décrits avec CIS pour prédire le temps d'exécution de cette technique dans ce contexte. CIS est donc un modèle destiné aux concepteurs d'interface pour faire les choix préliminaires d'une interface.

CIS ne modélise pas tous les détails de l'interaction et n'est pas un outil qui vise à se substituer aux expérimentations contrôlées. L'efficacité d'une technique dépend également de sa qualité de finition (notamment au niveau de la représentation graphique) qui ne peut être évaluée qu'au travers d'une

expérimentation une fois la technique programmée. Nous travaillons donc également à développer la plateforme *TouchStone* spécifiquement dédiée aux expérimentations contrôlées. D'une part, la plateforme TouchStone aide à concevoir des plans expérimentaux et à les exécuter et, d'autre part, elle peut agir comme un entrepôt de plans expérimentaux, de techniques testées et de résultats de ces expérimentations qui peut être consulté ou enrichi. *Pour le concepteur, TouchStone vient donc compléter CIS*. D'autre part, nous avons évoqué la difficulté de transposer les résultats des évaluations contrôlées à un autre contexte d'utilisation. CIS et SimCIS permettent d'explorer différentes tâches et de juger leur pertinence afin d'aider à la conception d'expérimentations contrôlées en comprenant le lien entre les résultats obtenus et la tâche expérimentale. *Pour l'évaluateur, CIS et SimCIS viennent donc compléter TouchStone*.

Nous avons également évoqué la difficulté de programmer l'interaction avancée avec les outils actuellement utilisés par les développeurs d'interfaces. La dernière contribution de cette thèse est donc une boîte à outils, *SwingStates*, qui introduit des structures de contrôle adaptées pour programmer l'interaction tout en minimisant l'effort nécessaire à la maîtrise de cet outil par les développeurs. En effet, SwingStates est une extension de Java Swing, une boîte à outils largement utilisée aussi bien dans le monde industriel qu'académique, et permet donc au développeur de rester dans leur cadre habituel de développement tout en leur offrant des facilités pour "sortir des sentiers battus".

1.4 Plan

Après une revue des modèles de description et évaluation des applications graphiques, nous posons les fondements du modèle CIS. Nous illustrons au travers d'exemples de techniques existantes et de techniques que nous avons réalisées la façon dont CIS permet d'une part d'optimiser les briques de base d'une technique d'interaction, d'autre part d'ouvrir un espace de conception doté de mesures. Les concepteurs peuvent alors l'utiliser comme source d'inspiration pour constituer un éventail de techniques candidates parmi lesquelles il peut sélectionner celle qui lui paraît la plus prometteuse. Nous proposons ensuite une validation et un raffinement de l'aspect prédictif de CIS en confrontant ses prédictions à des données empiriques collectées lors d'une expérimentation contrôlée comparant différentes techniques dans différents contextes d'utilisation. Puis, nous nous intéressons aux phases de développement des techniques précédemment sélectionnées avec l'introduction de SwingStates, notre extension de Java Swing pour programmer l'interaction avancée. Une fois la technique d'interaction finalisée, elle est prête à être évaluée formellement par une expérimentation contrôlée grâce à TouchStone, le dernier outil introduit dans cette thèse.

1.5 Champ de recherche

L'Interaction Homme-Machine (IHM) étant un domaine de recherche multidisciplinaire très vaste, il est important de préciser les frontières du travail présenté dans ce manuscrit. Les interfaces, lieux de rencontre entre l'utilisateur et le système, étudiées en IHM sont très hétérogènes. Les dispositifs d'interaction étudiés vont de petits dispositifs comme les montres à de grands systèmes comme les avions en passant par les ordinateurs de bureau, tandis que les styles d'interaction vont de l'interaction ambiante à la reconnaissance vocale en passant par les applications graphiques plus connues (les GUI pour Graphical User Interfaces) sans oublier qu'il peut s'agir de l'interaction d'un ou plusieurs humains avec un

système centralisé ou distribué. Bien que l'étude du passage à l'échelle de l'approche proposée dans cette thèse soit une perspective intéressante et importante, nous restreindrons notre étude à ces fameuses GUI avec un seul utilisateur, ce qui représente une très large proportion de la littérature en IHM et fait toujours l'objet d'une recherche active.

Modéliser les techniques et leur contexte d'utilisation avec CIS

Dans ce premier chapitre, nous présentons *Complexity of Interaction Sequences* (CIS), le modèle que nous avons défini afin d'aider les concepteurs à explorer un espace de conception et à décrire les techniques qui rentrent dans cet espace. Nous commençons par l'exposé des modèles descriptifs existants afin de comprendre et expliquer le succès limité qu'ils rencontrent en les classant selon les deux critères suivants : la partie de l'interaction qu'ils permettent de décrire et leur niveau d'abstraction. En effet, dans le cadre de notre "approche générative", le concepteur d'interface doit pouvoir garder une vue d'ensemble de l'interaction et pouvoir la décrire avec un niveau d'abstraction approprié, c'est-à-dire ni trop faible pour ne pas se noyer dans les détails, ni trop élevé pour rester pertinent dans le cadre des applications graphiques. Nous présentons ensuite le modèle Complexity of Interaction Sequences (CIS) qui permet de décrire les techniques d'interaction ainsi que leurs contextes d'utilisation. Enfin, nous identifions un ensemble de propriétés qui peuvent être utilisées pour explorer un espace de conception avec CIS.

2.1 Du périphérique d'entrée à la tâche : Décrire l'interaction

Les modèles pour décrire les systèmes interactifs sont nombreux et de natures diverses. On peut y distinguer deux grandes familles : les "modèles d'architecture logicielle" ([43] par exemple) qui servent à décrire le rôle des différents composants logiciels dans un système réactif et les "modèles de l'interaction" ([24] par exemple) qui se focalisent sur la description du comportement de l'utilisateur et des interfaces. Alors que la première famille vise à aider les développeurs à produire des programmes bien conçus, la deuxième famille décrit l'interaction plus "grossièrement" dans le but d'identifier les propriétés de l'interaction. Ces derniers sont donc généralement utiles dans les premières étapes de conception d'une interface et font l'objet de ce chapitre.

Les "modèles de l'interaction" ont pour but de décrire les mécanismes mis en oeuvre pour établir un dialogue entre l'utilisateur et le système. Nous débutons par les modèles qui décrivent l'interaction au niveau du périphérique d'entrée, capteur des actions motrices de l'utilisateur, pour évoluer vers des modèles plus larges : d'abord, ceux qui considèrent aussi les objets logiciels qui se situent au coeur de l'interaction, puis ceux qui incluent la notion de tâche ou d'activité. Dans cette dernière famille, nous distinguons trois sous-ensembles. Tout d'abord, les modèles "comportementalistes", qui se concentrent sur la description des actions motrices de l'utilisateur et des réactions de l'interface et leur agencement au cours du temps pour réaliser une tâche. Puis, les modèles "cognitivistes", qui s'intéressent plus aux processus cognitifs mis en oeuvre pour exécuter une tâche. Enfin, les modèles basés sur la théorie de l'activité qui incluent la description des aspects sociaux et environnementaux. Pour chacun de ces modèles, nous évaluerons : (1) son niveau d'abstraction, c'est-à-dire sa distance à l'implémentation, afin de voir s'il s'agit d'un modèle qui peut être utilisé dans les étapes préliminaires de conception ; (2) la partie de l'interaction couverte afin de voir si le modèle se place plutôt d'un point de vue utilisateur, machine ou les deux.

2.1.1 Au niveau du périphérique d'entrée

Pendant les années 70-80, les modèles pour décrire l'interaction se sont focalisés sur des descriptions très concrètes et de bas niveau. Nous allons voir dans cette section que ces modèles sont capables de

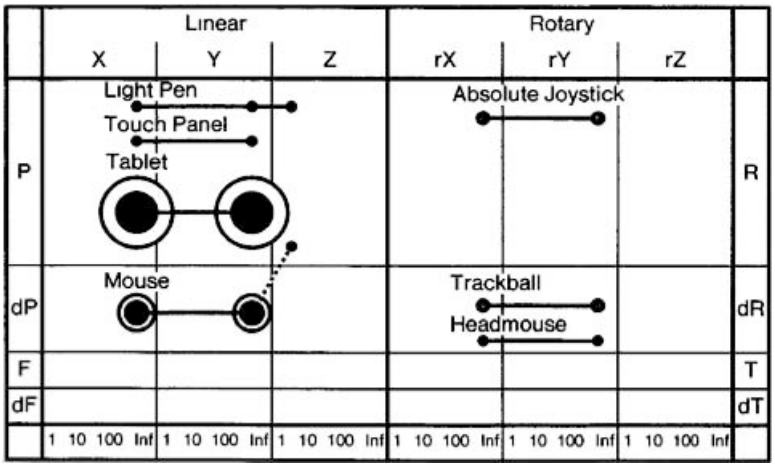


FIG. 2.1 : Représentation de l'espace de conception des périphériques de Card et al. (*Illustration extraite de [47]*)

décrire finement l'interaction au niveau du périphérique d'entrée mais ne permettent pas d'apprécier l'interaction dans son ensemble.

Foley et al. [65, 66] sont les premiers à proposer une classification des interactions selon six tâches élémentaires. Dans [66], ils proposent de classer les périphériques d'entrée selon les tâches qu'ils permettent de réaliser : *sélectionner* un objet, *positionner* un objet en 1, 2 dimensions ou plus, *orienter* un objet en 1, 2 dimensions ou plus, *dessiner*, entrer du *texte*, spécifier une *valeur* scalaire. Plutôt que de les classer par rapport aux tâches qu'ils permettent de réaliser, Buxton [39] propose une taxonomie des périphériques d'entrée dans laquelle un périphérique est classé selon les propriétés captées (position, mouvement ou pression), le nombre de dimensions captées et les groupes de muscles impliqués dans leur utilisation. En 1991, Card et al. [47] proposent une définition plus formelle et introduisent un espace de conception pour les périphériques. Dans ce dernier modèle, un périphérique d'entrée est défini comme un traducteur d'actions physiques en commandes logiques de bas niveau, il peut être placé dans un espace composé de dimensions descriptives (ex : relatif vs. absolu, linéaire vs. rotatif, etc.) et décrit comme le résultat de l'application d'un opérateur sur ces dimensions. Par exemple, la souris inclut une composition de deux "sliders" unidimensionnels pour former un dispositif de pointage en deux dimensions. La figure 2.1 illustre comment les dimensions descriptives peuvent être utilisées pour construire une représentation de l'espace de conception dans un plan. Un périphérique est une composition d'un ensemble de points dans cet espace (la taille d'un point est fonction de son *footprint*, c'est-à-dire la taille de la représentation du périphérique dans l'interface comme le curseur de la souris par exemple).

Parallèlement à l'introduction du modèle de Card et al. que l'on peut qualifier de statique, Buxton [40], motivé par les principes de la manipulation directe [159], introduit un modèle dynamique. Dans son modèle, un périphérique d'entrée est décrit sous forme d'un automate, ou machine à états. Cet automate contient trois types d'états : *Out of Range* (un mouvement n'a aucun effet sur le système), *Tracking* (un

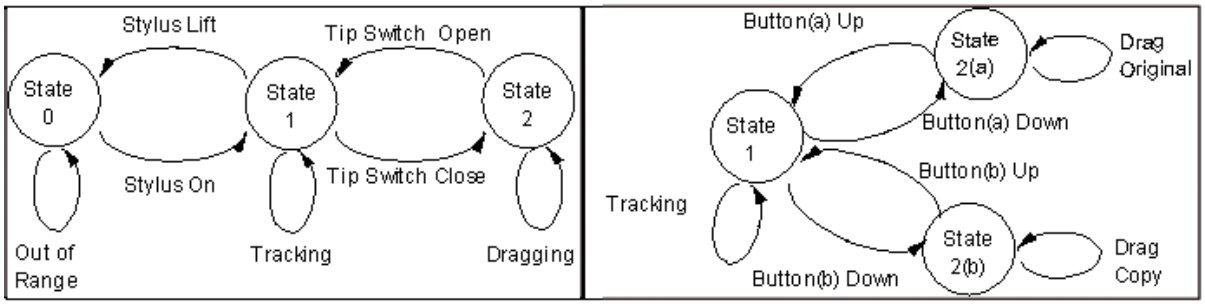


FIG. 2.2 : modèle 3-state : un stylet (à gauche) et une souris à deux boutons (à droite) (*Illustration extraite de [40]*)

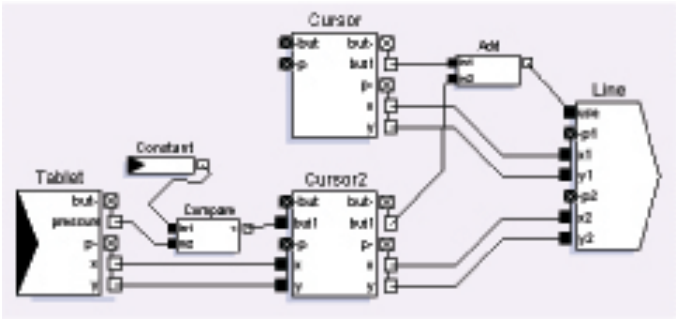
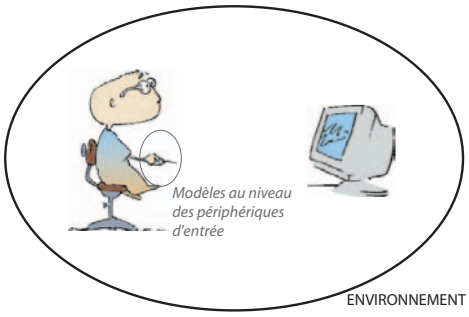


FIG. 2.3 : Configuration ICON (*Illustration extraite de [59]*)

mouvement est traduit par un déplacement du curseur) et *Dragging* (un mouvement est traduit par un déplacement du curseur et d'un objet). Les transitions d'un état à l'autre sont étiquetées par les actions physiques correspondantes. La figure 2.2 montre les descriptions respectives d'un stylet et d'une souris à deux boutons avec ce modèle *3-state*. En 1998, motivés par l'interaction bimanuelle [76, 101], Hinckley et al. [90] raffinent ce modèle en y incluant des annotations qui correspondent aux dimensions que l'on trouve dans le modèle de Card et al. et montrent qu'une TouchMouse (une souris augmentée d'un capteur de contact) et un puck sur une tablette, qui ont une description identique dans le modèle de Buxton, deviennent maintenant distinguables.

Il existe des outils pour concevoir l'interaction à ce niveau d'abstraction comme ICON [59] qui est un éditeur graphique pour configurer un ensemble de périphériques d'entrée et les connecter aux actions d'une application. La Figure 2.3 montre par exemple comment composer un stylet (à gauche) et une souris (en haut) pour permettre de tracer des segments dans un éditeur graphique. La configuration décrite ici permet d'utiliser stylet et souris en parallèle pour contrôler indépendamment les deux extrémités du segment dessiné par une interaction bimanuelle.

En résumé...



Ces modèles permettent de décrire l'interaction à un niveau très fin mais, principalement, dans le sens utilisateur → machine puisqu'ils se focalisent sur la description des périphériques d'entrée. Bien que cette description concrète devra intervenir durant la conception, le niveau d'abstraction de ces modèles n'est pas adéquat pour "esquisser" les composants d'une interface pendant les stades préliminaires de la conception. De plus, comme le titre de cette section l'indique, ces modèles voient l'interaction en termes d'actions de l'utilisateur sur les périphériques d'entrée or les premières phases de conception demandent de regarder l'interface dans sa globalité.

2.1.2 Les modèles "centrés objet"

Les modèles que nous venons de voir décrivent principalement les actions du côté utilisateur mais pas les réactions du système à ces actions. Il existe des modèles qui se concentrent plus sur l'interface en décrivant les objets qui la composent.

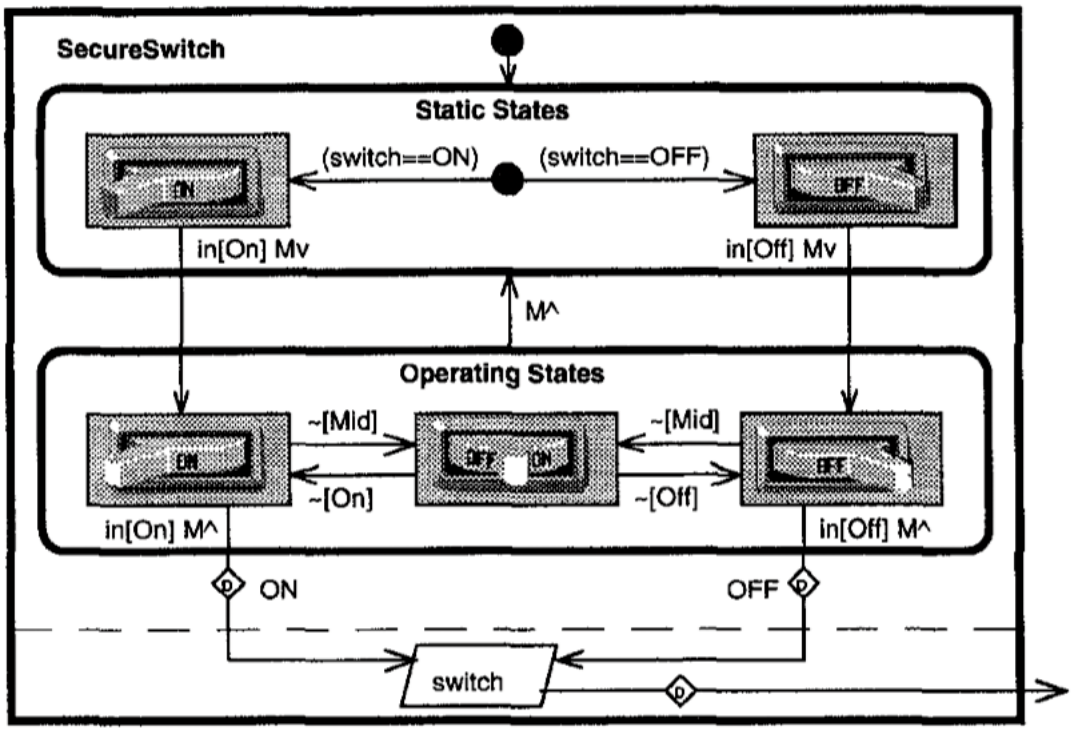


FIG. 2.4 : [IOG pour spécifier l'apparence et le comportement d'un interrupteur sécurisé] (*Illustration extraite de [51]*)

Les Interaction Object Graph (IOG) [51] sont des extensions des statecharts de Harel [86] pour décrire les objets d'interaction. Les statecharts sont un formalisme pour décrire les états d'un système et les transitions entre ces états et dans lequel plusieurs états peuvent être actifs simultanément. Les IOGs contiennent plusieurs types d'état (des états spécifiques à l'affichage par exemple) et plusieurs types de transition (des transitions spécifiques aux contraintes de placement graphique par exemple) afin de spécifier très finement un objet d'interaction. La figure 2.4 montre un exemple d'un IOG pour spécifier un interrupteur avec un niveau de sécurité (l'interrupteur contient un état neutre intermédiaire entre ON et OFF).

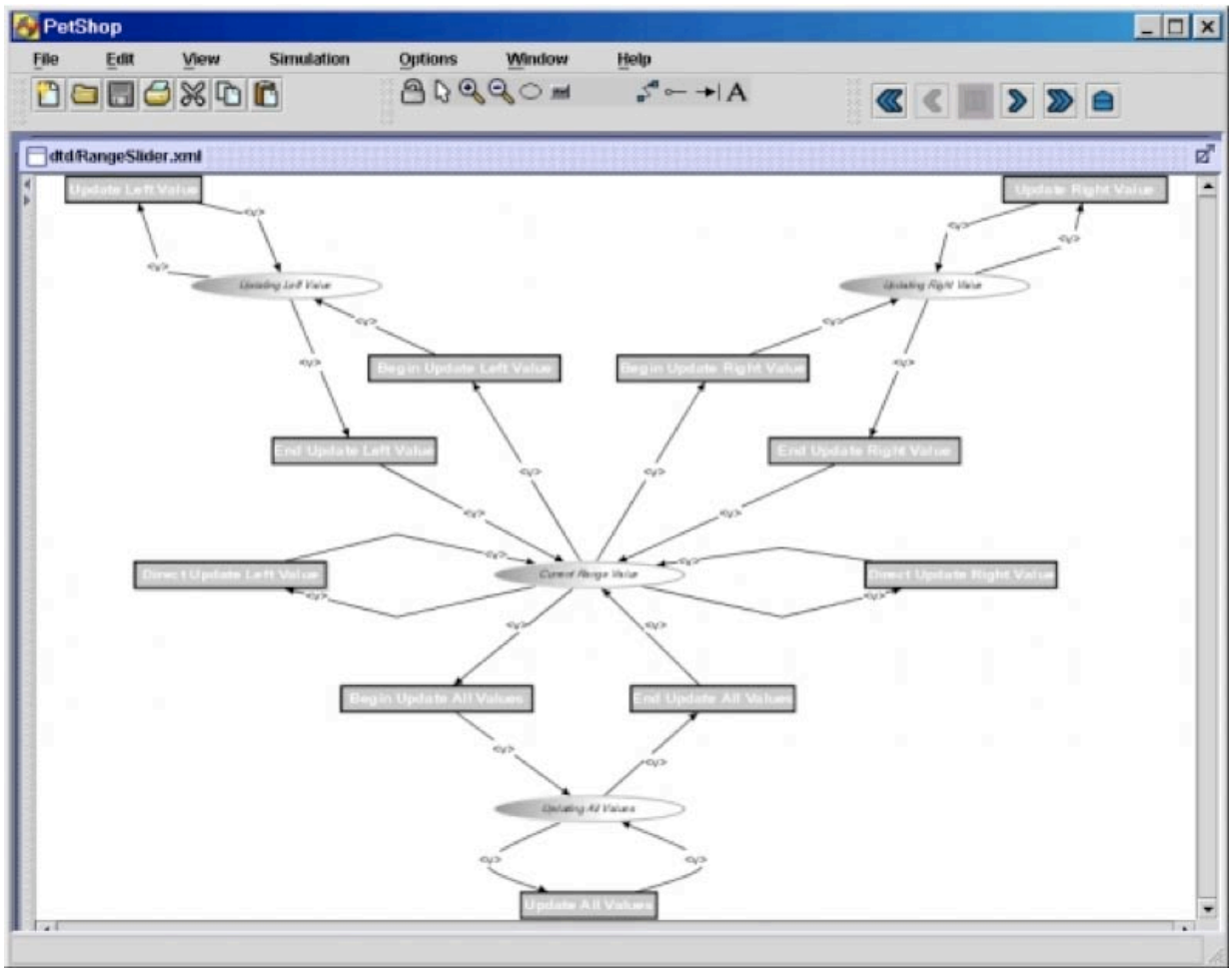


FIG. 2.5 : L'environnement PetShop (*Illustration extraite de [130]*)

Le formalisme Interactive Cooperative Objects (ICOs) [137] repose sur la combinaison d'une approche orientée objet et du formalisme des réseaux de Petri [139], un formalisme doté d'un pouvoir d'expression plus grand que les statecharts. L'état du système est explicitement représenté par la position

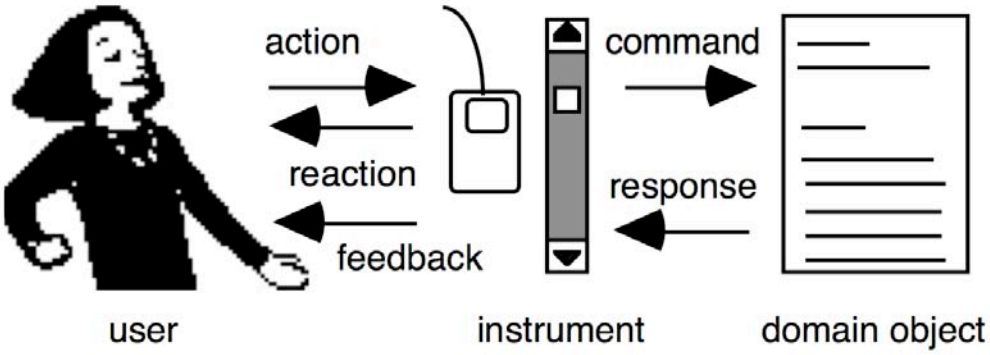
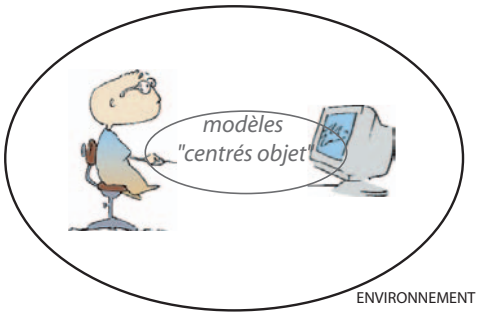


FIG. 2.6 : Le modèle de l'interaction instrumentale (*Illustration extraite de [24]*)

de jetons dans les places du réseau. Les jetons peuvent changer de place en passant des transitions sous certaines conditions comme le nombre de jetons actuellement dans une place. Les ICOs modélisent un système interactif comme un ensemble d'objets communiquant ensemble. La partie statique des objets est exprimée dans un langage orienté objet tandis que la communication entre objets et le comportement dynamique d'un objet sont décrits avec des réseaux de Petri. La figure 2.5 montre le réseau de Petri d'un range slider [6] édité dans Petshop [21], un environnement pour exécuter un modèle ICO. En effet, ICO est un formalisme dont le niveau d'abstraction est celui d'un langage de programmation et permet donc d'obtenir des prototypes exécutables et auxquels on peut appliquer des méthodes de vérification formelles.

La description d'une interface en utilisant des modèles très formels comme les ICOs ou les IOGs est très détaillée et demande donc beaucoup de temps. De plus, leur niveau d'abstraction est faible et il est donc très difficile d'apprécier les propriétés qualitatives d'une technique. Le modèle de l'interaction instrumentale [24] offre un niveau d'abstraction plus élevé et propose de penser l'interface en termes d'objets du domaine et d'instruments d'interaction pour agir sur les objets du domaine. L'instrument transforme les actions de l'utilisateur en commandes qui agissent sur les objets du domaine. L'instrument réagit aux actions de l'utilisateur et fournit un feedback quant à l'état de l'objet du domaine qu'il contrôle (figure 2.6). L'interaction instrumentale est un modèle peu formel qui permet de comparer des instruments selon un ensemble de propriétés faciles à utiliser. Par exemple, le *degré d'indirection* mesure la distance entre objet et instrument et ainsi que l'intervalle de temps entre l'action de l'utilisateur et l'effet sur l'objet. Par exemple, la scrollbar a un degré d'indirection plus fort que la "main Adobe" qui permet de faire glisser un document. Le niveau d'abstraction de l'interaction instrumentale permet de laisser des espaces de conception ouverts et fournit des principes pour l'explorer. Par exemple, le principe de réification consiste à transformer des concepts en objets concrets. C'est notamment en utilisant ce principe qu'a été proposé un outil performant pour rechercher et remplacer des chaînes de caractères dans un éditeur de texte [24].

En résumé...



Tous ces modèles "centrés objet" permettent de décrire comment les utilisateurs manipulent les objets de l'interface et la réaction de ces objets à ces manipulations. Cependant, les trois modèles que nous avons présentés diffèrent grandement dans leur niveau d'abstraction. En effet, le vocabulaire d'entrée des ICOs correspond aux événements captés par les périphériques d'entrée et est donc très proche du niveau de l'implémentation. Le vocabulaire d'entrée des IOGs est exprimé par la notation User Action Notation que nous allons voir à la section suivante et qui est également bas niveau puisque que c'est un vocabulaire destiné à communiquer avec une équipe de développement. L'interaction instrumentale est, au contraire, peu formalisée et de haut niveau pour décrire les interactions en termes d'objets ayant un rôle dans l'interaction et comparer les instruments. Ces trois modèles décrivent principalement les objets qui sont au coeur de l'interaction entre l'humain et la machine.

2.1.3 Au niveau de la tâche

Nous passons maintenant en revue les modèles qui permettent de décrire l'interaction au niveau de la tâche ou de l'activité. Ils peuvent être regroupés selon trois grandes familles : les modèles "comportementalistes", les modèles "cognitivistes" et ceux basés sur la théorie de l'activité.

Les modèles "comportementalistes"

Les modèles "comportementalistes", tels que UAN [87] ou Keystroke-level model [48], utilisent comme briques de base principales les actions motrices. La notion de tâche apparaît lors de la composition de ces briques de base selon les buts que l'utilisateur cherche à atteindre.

User Action Notation, UAN [160, 87] est un langage pour capturer le processus de conception d'une interface. Le but principal de cet outil est de faciliter la communication entre les concepteurs et développeurs selon une séquentialisation des outils utilisés : d'abord UAN pendant la conception puis des outils spécifiquement destinés à la programmation pendant l'implémentation.

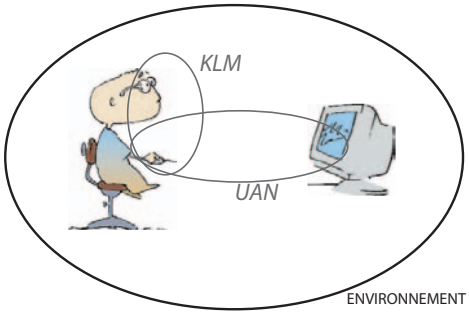
La figure 2.7 donne un exemple d'une description UAN pour la tâche qui consiste à déplacer l'icone d'un fichier. La tâche est modélisée par l'ensemble du tableau, la colonne de gauche décrit la séquence d'actions motrices permettant de remplir cette tâche, la seconde colonne décrit les feed-backs (réactions aux actions à l'utilisateur), tandis que les deux dernières colonnes listent les effets des actions sur l'état de l'application décrite. Un des aspects particulièrement intéressant d'UAN est l'expressivité et la simplicité de la description des actions (figure 2.7, première colonne). Nous verrons dans la section 2.2.1 une comparaison entre ces opérateurs et les opérateurs de notre modèle CIS. Les autres opérateurs d'UAN, qui ne décrivent pas le comportement de l'utilisateur, sont des opérateurs relatifs au feed-back et à l'état de l'interface. Ces informations sont particulièrement importantes dès lors qu'UAN est destiné à être facilement traduisible en langage de programmation. D'autres modèles comme Keystroke-Level Model (KLM) adoptent un niveau de description similaire pour leurs briques de base mais leur syntaxe est moins riche et moins expressive.

TASK: move a file_icon			
USER ACTIONS	**INTERFACE FEEDBACK**	**INTERFACE STATE**	**CONNECTION TO COMPUTATION**
~[file_icon] Mv	file_icon-!: file_icon!, ∀file_icon'!: file_icon'-!	selected = file	
~[x,y]*~[x',y']	outline(file_icon) > ~		
Mˆ	@x',y' display(file_icon)		location(file_icon) = x', y'

FIG. 2.7 : Description UAN de l'interaction pour déplacer un icone (*Illustration extraite de* [87])

Alors que les actions d'UAN incluent des opérateurs pour décrire plus finement l'interface, KLM ajoute aux opérateurs moteurs (K : frappe au clavier ou appui de bouton, P : pointage et D : dessin) des opérateurs (M et R) destinés à modéliser des aspects relatifs au dialogue humain-machine en temps réel : M représente la "préparation mentale" de l'utilisateur et R représente le temps de réponse de la machine, par exemple pour exécuter une commande. L'introduction de ces opérateurs est motivée par la propriété prédictive de KLM sur laquelle nous allons revenir à la section suivante. KLM compose ces opérateurs selon une *méthode*, c'est-à-dire un ordonnancement des opérateurs pour accomplir une tâche. Il propose de plus un ensemble de règles heuristiques pour le placement des opérateurs M. Avec KLM, la séquence correspondante à la tâche de la figure 2.7 est MPKMPK. M : préparer la prochaine opération - P : déplacer le curseur vers l'icone à déplacer - K : presser le bouton de la souris - M : préparer la prochaine opération - P : déplacer le curseur vers la position désirée - K : relâcher le bouton de la souris. Pour placer les opérateurs M, nous avons utilisé les règles (1) placer un opérateur M avant chaque opérateur et (2) supprimer les opérateurs M qui sont inclus dans un schéma PMK.

En résumé...



À l'exception de l'opérateur M de KLM, ces modèles décrivent l'interaction d'un point de vue "comportementaliste" et font émerger l'analyse de l'interaction graphique à deux niveaux : au premier niveau, les actions motrices, qui sont composées pour décrire un deuxième niveau, celui de la tâche. La comparaison de la figure 2.7 et de la séquence KLM correspondante (MPKMPK) illustre bien le fait que KLM est plus abstrait qu'UAN. De plus, les opérateurs de KLM décrivent principalement l'interaction du côté utilisateur alors qu'UAN couvre une plus grande partie de l'interaction en permettant de décrire des informations relatives au feed-back.

Les modèles "cognitivistes"

L'opérateur M du modèle KLM marque l'avènement de toute une famille de modèles dont le but est de décrire l'interaction sous un angle cognitif. Les modèles "cognitivistes" sont également centrés sur la notion de tâche mais, contrairement aux modèles "comportementalistes" qui sont centrés sur le comportement articulatoire des actions motrices de l'utilisateur, ils cherchent à décrire les processus cognitifs.

Les premiers modèles cognitifs pour décrire l'interaction reposent sur le formalisme des grammaires. En 1981, Reisner propose d'utiliser des grammaires formelles pour décrire des systèmes graphiques interactifs [153]. Une application interactive est décrite en utilisant des symboles non terminaux, qui correspondent aux commandes de l'application, qui peuvent être réécrits sous forme de symboles terminaux, appelés *cognitive terminals*, qui correspondent aux actions que l'utilisateur doit mémoriser (enclencher le mode dessin, désigner cette forme avec le curseur, etc.). En 1986, Payne et Green introduisent la notion de tâche avec le modèle Task-Action Grammar (TAG) [138] qui comprend des grammaires génératives pour réécrire des tâches simples sous forme d'actions. La famille de modèles qui a succédé à TAG et qui est la plus représentée dans la littérature est la famille GOMS (Goals, Operators, Methods and Selection rules) [99, 98] qui comprend quatre modèles : KLM, CMN-GOMS, NGOMSL et CPM-GOMS. KLM est considéré comme membre de cette famille mais ne contient ni la notion de buts (Goals), ni celle de méthodes (Methods), ni celles de règles de sélection (Selection rules). KLM ne décrit que des séquences linéaires d'opérateurs (Operators), c'est pourquoi nous l'avons traité comme modèle comportementaliste. C'est le modèle de Card, Moran et Newell, CMN-GOMS [46], qui introduit formellement la définition des composants d'un modèle GOMS. L'utilisateur cherche à atteindre un *but* qui est une décomposition hiérarchique de en sous-buts. Pour cela, il utilise des *méthodes* qui sont des séquences "bien connues" de sous-buts et d'*opérateurs* qui lui permettent d'atteindre des buts de haut niveau, les opérateurs étant les actions rendues disponibles par le logiciel. Les *règles de sélection* sont des règles ad hoc pour choisir une méthode lorsqu'un but peut-être atteint par différentes méthodes (par exemple, *si* le but est "sélectionner du texte" *et* que le texte est composé d'un seul mot *alors* double-cliquer sur le texte *sinon* presser le bouton de la souris au début du texte *puis* bouger jusqu'au bout du texte). NGOMSL introduit une syntaxe plus formelle pour la description d'un modèle GOMS, très proche de l'écriture d'un programme. Enfin CPM-GOMS permet de décrire l'utilisation parallèle d'opérateurs Cognitifs, Perceptuels et Moteurs.

Tous ces modèles cognitivistes plus avancés que KLM décrivent ce que l'utilisateur doit savoir pour utiliser une interface et reposent sur des architectures cognitives qui décrivent l'acquisition et l'utilisation de ces connaissances (la partie supérieure de la figure 2.8). La première architecture cognitive est le Model Human Processor (MHP) de Card et al. [49] qui regroupe de nombreuses recherches en psychologie cognitive. Il existe d'autres architectures cognitives comme ACT [9] ou EPIC [102]. ACT définit la cognition complexe comme une interaction entre les connaissances procédurales et declaratives. Les connaissances procédurales sont représentées par un ensemble de règles de production tandis que les connaissances déclaratives sont des ensembles d'unités appelées "chunk". Les règles de production sont issues d'encodages de transformations dans l'environnement et les "chunks" sont des objets de l'environnement. L'humain extrait alors de cette base les connaissances appropriées face à un contexte donné. L'architecture EPIC va plus loin et se focalise sur la modélisation de l'interaction en proposant de représenter l'être humain comme une machine. Une architecture EPIC est composée d'un proces-

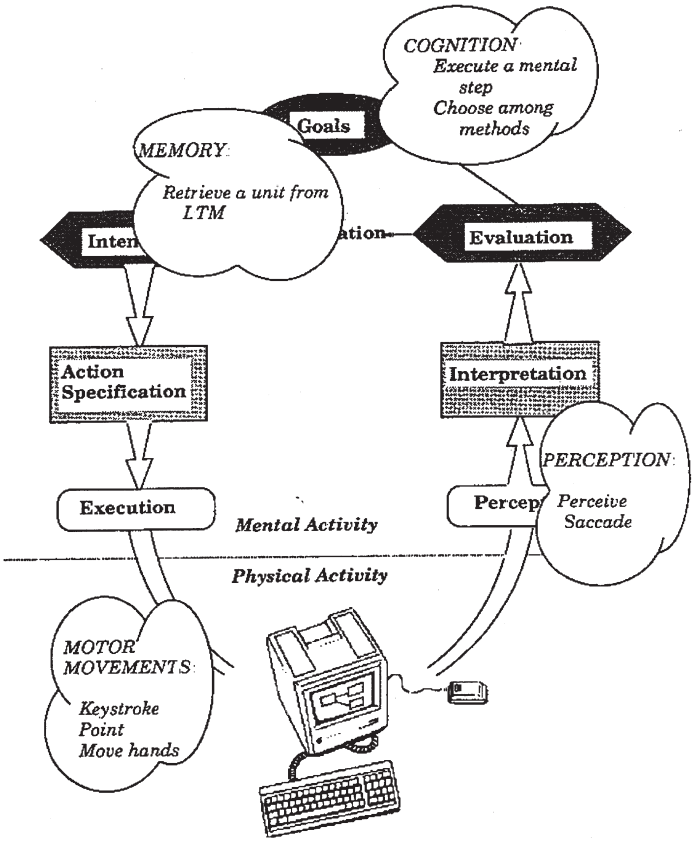


FIG. 2.8 : Les éléments modélisés par une architecture cognitive (*Illustration extraite de* [135])

seur cognitif, d'un interpréteur de règles de production et d'une mémoire de travail, le tout entouré de capteurs périphériques, processeurs perceptuels et moteurs qui opèrent en parallèle. L'interaction d'un utilisateur est alors une composition de règles de production lui permettant d'atteindre un but donné. Un changement de but correspond, par exemple, à l'effacement du but courant de la mémoire de travail suivi de l'écriture du nouveau but dans celle-ci. Dans un tel cadre, la modélisation de l'interaction demande l'écriture d'un programme destiné à être interprété par l'"humain". En quelque sorte, les modèles cognitivistes décrivent l'interaction homme-machine comme une interaction machine-machine. L'utilisation d'un modèle GOMS suppose de connaître ces architectures afin d'être capable de décrire l'interface en termes de méthodes et règles de sélection que l'humain peut utiliser.

Parmi les modèles que nous avons décrits jusqu'ici, les modèles GOMS sont les seuls à avoir été conçus dans le but de prédire des mesures de performance de l'interaction décrite. Idéalement, les méthodes d'analyse fondées sur ces modèles autorisent différents niveaux d'abstraction : plus l'effort de modélisation est grand, plus les prédictions sont précises. Elles peuvent ainsi être utilisées dans les étapes préliminaires de conception, avant le prototypage et les études d'utilisabilité, pour obtenir des prédictions quantitatives et aider dans les premiers choix sans investir trop de temps dans la modélisation, les pré-

dictions pouvant être raffinées au fur et à mesure de la conception. Ces méthodes d'analyse pour prédire ne visent pas à remplacer les études auprès des utilisateurs mais à réduire la quantité nécessaire de ces études pour développer un système "hautement utilisable". Elles permettent d'obtenir différentes mesures : le temps pour exécuter une tâche, le temps d'apprentissage d'une méthode ou encore le nombre moyen d'erreurs faites par l'utilisateur.

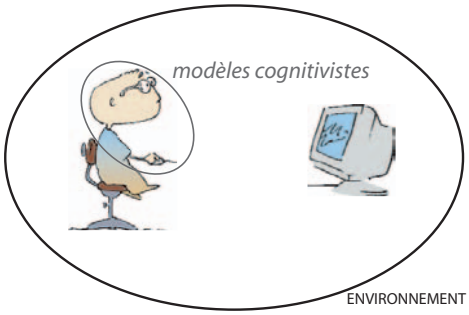
La principale mesure de performance que l'on peut prédire avec une modélisation GOMS est le *temps d'exécution* d'une méthode pour atteindre un but. A l'exception de CPM-GOMS qui propose une modélisation complexe incluant l'exécution parallèle de certains opérateurs, les modèles GOMS utilisent un mécanisme de grammaire pour traduire une méthode en une séquence d'opérateurs. Le temps requis par la méthode est alors la somme des temps estimés pour chaque opérateur de la séquence. Dans le cas le plus simple, on obtient une séquence de *keystrokes* de KLM pour lesquels on dispose de temps moyens reportés dans la littérature (K : 280 ms, M : 1,35 s, P : 1,1 s et H : 400 ms dans [46]). CPM-GOMS est plus réaliste et plus précis mais la quantité et la complexité de la modélisation en sont d'autant plus grandes.

Reisner [153] proposait de profiter du formalisme des grammaires pour comparer la *facilité d'apprentissage* de deux applications en regardant les profondeurs d'un même symbole terminal dans les deux modèles. NGOMSL est la seule méthode qui permet aussi de prédire le *temps d'apprentissage* d'une méthode pour atteindre un but. Contrairement au temps d'exécution, la mesure du temps d'apprentissage d'une méthode obtenue par une analyse NGOMSL n'est pas une mesure absolue mais elle prend son sens dans la comparaison de différentes méthodes. Il est aussi très important de rappeler que cette mesure est appelée "temps d'apprentissage pur pour les connaissances procédurales", elle est estimée comme une fonction linéaire du nombre de routines NGOMSL. Le caractère relatif de cette mesure s'explique donc par l'introduction d'un coefficient à déterminer empiriquement qui dépend des "conditions d'apprentissage". Pour fournir des prédictions, les modèles GOMS se restreignent au cadre où l'utilisateur connaît le système : il a simplement à reconnaître des situations et à effectuer les actions appropriées. C'est une hypothèse dont il est difficile de s'écarter dès lors que l'on veut modéliser formellement l'interaction avec un humain et tenter d'obtenir des mesures les plus objectives possibles. D'autres méthodes non automatiques, comme le Cognitive Walkthrough [145], laissent une plus grande liberté d'interprétation à l'analyste pour évaluer une interface à partir d'une simple maquette détaillée. La description d'une tâche est ici encore une structure hiérarchique ayant des actions pour feuilles. La méthode de Cognitive Walkthrough exige de l'évaluateur qu'il considère les actions une à une pour répondre systématiquement à un ensemble de questions pour donner une note à l'interface. Cette note, obtenue par une simulation du comportement exploratoire des utilisateurs dits "novice" par les expérimentateurs eux-mêmes, mesure la difficulté à découvrir le système évalué et constitue donc, d'une certaine façon, une mesure de l'apprentissage du système.

Le dernier type de mesure que ces modèles prédisent est le nombre d'*erreurs* faites par l'utilisateur. Card et al. [46] avaient mentionné que l'une des causes des erreurs des utilisateurs était la surcharge de la mémoire de travail. Quelques études empiriques ont observé cette relation entre charge de la mémoire de travail et nombre d'erreurs. Par exemple, Lerch et al. [112] ont utilisé GOMS pour modéliser différentes tâches qui consistaient à entrer des formules dans un tableur en variant la charge de mémoire qu'elles

occasionnaient et ont effectivement observé que le nombre d'erreurs pour entrer des formules augmentait avec cette charge.

En résumé...



Rappelons qu'idéalement toutes ces méthodes d'analyse prédictives doivent permettre de gagner du temps par rapport aux études auprès des utilisateurs et qu'elles doivent donc pouvoir être utilisées rapidement et facilement dans les premiers stades de conception. Or l'idée d'atteindre des prédictions très fines a motivé l'introduction d'architectures cognitives très complexes comme EPIC qui décrivent les processus cognitifs à un niveau très fin et qui sont difficiles à comprendre et à utiliser. D'une manière générale, les méthodes d'analyse GOMS souffrent de critiques sur la quantité de temps nécessaire à leur utilisation (voir, par exemple, [155]).

La théorie de l'activité

La théorie de l'activité a été introduite dans les années 1920 par des psychologues soviétiques comme Leontiev ou Vygotski [111, 171]. Selon cette théorie, l'humain n'est pas une composition de raisonnements et actions à l'échelle de l'individu mais il se définit par ce qu'il fait et ce qu'il fait s'inscrit dans un environnement social constitué de personnes et d'*artefacts*. La théorie de l'activité n'est pas une théorie au sens strict du terme, c'est en fait un ensemble de principes qui constitue un concept généraliste destiné à supporter des théories plus spécifiques. Le principe de l'orientation objet (*object-orientedness*) stipule que les êtres humains vivent dans une réalité qui est faite d'objets qui n'ont pas seulement des propriétés considérées comme objectives mais aussi des propriétés sociales et culturelles. Les activités sont hiérarchiques et font l'objet d'un développement continu. Il y a deux types d'activité : les activités *internes*, qui correspondent au classique processus mental individuel, et les activités *externes*, qui regroupent le reste des activités. La frontière entre les deux types n'est pas imperméable et les processus d'*internalisation* et *externalisation* permettent le passage d'une activité d'un type vers l'autre. Un dernier principe important porte sur le rôle de *médiation des outils* pour l'activité : les outils sont les moyens utilisés par les humains pour interagir avec l'environnement et ils sont le reflet d'expériences passées de personnes confrontées au même problème. Les propriétés structurelles (forme, matériaux, etc.) qu'un outil a acquis au cours du développement continu d'une activité sont une accumulation et une transmission du savoir social.

L'utilisation de cette théorie dans le domaine de l'interaction homme-machine a été proposée au début des années 1990 par Bannon et Bødker [20, 35, 19] pour compléter les descriptions comportementalistes et cognitivistes d'un système. La figure 2.9 illustre les trois niveaux hiérarchiques d'une activité dans cette théorie qui ont été identifiés par Kuutti [129] et la figure 2.10 donne des exemples pour chacun de ces niveaux. Les activités sont des processus de long terme motivés par des objets qui ne peuvent être transformés immédiatement en résultats mais en utilisant des chaînes d'actions ayant des buts immédiats. Une action, individuelle ou coopérative, est une chaîne d'opérations. Une opération est une routine utilisée inconsciemment pour répondre aux conditions auxquelles les agents sont confrontés au moment

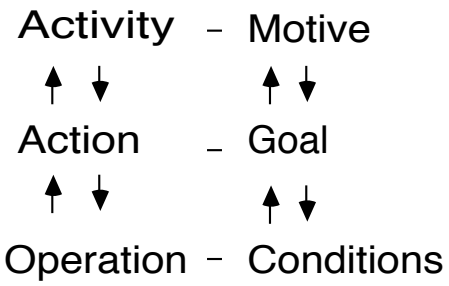


FIG. 2.9 : Les niveaux hiérarchiques d'une activité (*Illustration extraite de* [129])

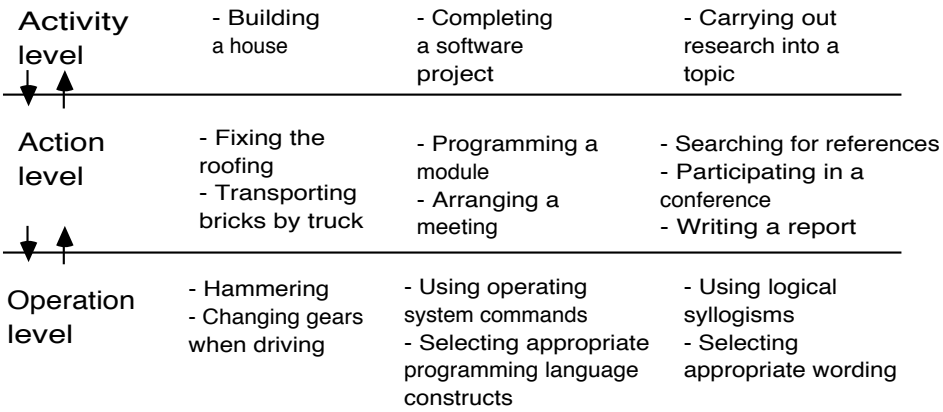


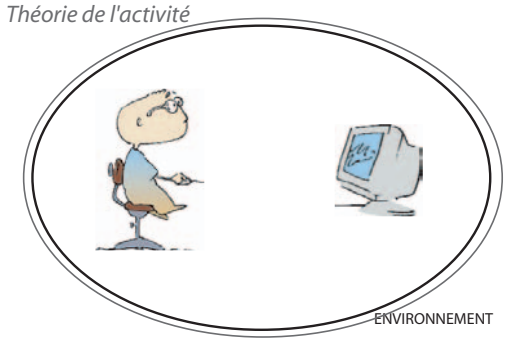
FIG. 2.10 : Exemples d'activités, actions et opérateurs (*Illustration extraite de* [129])

d'une action. La définition et la structure hiérarchique d'une activité ne sont pas sans rappeler les tâches des modèles cognitifs, la différence se situant plutôt dans la description des mouvements dynamiques qui permettent le passage d'un niveau hiérarchique à l'autre [129].

Dans le cadre de la théorie de l'activité, l'humain n'acquiert pas seulement des informations sur les caractéristiques d'un objet par une stimulation directe des sens provoquée par l'objet mais aussi par une activité cognitive et perceptuelle commandant des contacts avec l'objet doté de réaction et c'est cette action-réaction qui contient toute l'information. En s'appuyant sur ce principe complété par l'approche écologique de Gibson [70], Bærentsen propose un des rares travaux qui ait étudié les interfaces graphiques en utilisant la théorie de l'activité. Il expose des principes de conception qui restent très généraux pour développer des "Intuitive User Interfaces" [17] comme, par exemple, l'utilisation de métaphores issues de l'expérience passée des utilisateurs avec la technologie afin de faciliter l'apprentissage de fonctions complexes qui ne peuvent être perçues directement. Son étude est en grande partie fondée

sur l'interprétation du concept d'*affordance*[1], c'est-à-dire ce qu'un objet "dit" à l'humain sur son mode d'utilisation.

En résumé...



Les modèles basés sur la théorie de l'activité sont des modèles purement descriptifs [129] de très haut niveau qui prennent encore plus de facteurs en compte (sociaux, environnementaux, etc.) que les modèles cognitivistes. Or ces derniers souffraient déjà de critiques sur la quantité d'efforts nécessaire à l'apprentissage du modèle et à la description d'un système. De plus, nous avons vu que leur utilisation dans le domaine de l'interaction homme-machine n'a produit que des principes très généraux qui restent assez flous et peu utilisables en pratique.

Synthèse

Tous ces modèles sont principalement descriptifs. Les modèles qui se situent au niveau de la tâche / activité peuvent être utilisés pour vérifier la couverture et la consistance de l'interface d'un système en décrivant toutes les tâches correspondant aux fonctionnalités offertes par le système. Ils peuvent également être utilisés pour analyser la fonctionnalité d'une interface en regardant, par exemple, s'il existe une méthode pour réparer chaque type d'erreur (si l'utilisateur a exécuté par erreur une commande, on peut analyser si la suite d'actions pour revenir à l'état précédent existe et si elle est acceptable). Cependant, dans le cadre de notre approche générative, nous cherchons un modèle qui permette non seulement de décrire mais également d'évaluer la performance d'une technique pour faire des choix et à proposer des guides de conception qui permettent la génération de nouvelles techniques d'interaction. Seuls les modèles cognitivistes et KLM sont prédictifs et permettent d'évaluer des interactions mais, à l'exception de KLM, ils requièrent un coût de modélisation très grand car ils reposent sur des architectures cognitives complexes. Enfin, seule l'interaction instrumentale a des propriétés génératives car elle permet de regarder l'interaction à un niveau d'abstraction assez élevé et ouvre un espace de conception avec des guides (réification, etc.) pour créer des points dans cet espace.

La figure 2.11 classe les principaux modèles par rapport à leur niveau d'abstraction. Alors que notre concepteur cherche toujours à "esquisser", il a besoin d'un modèle dont le niveau d'abstraction est assez élevé pour ne pas aborder immédiatement les détails de l'implémentation. Il n'est probablement pas enclin ou capable de décrire à ce stade les processus cognitifs de l'utilisateur ou encore les détails de l'implémentation (comme le font les ICOs) et aurait donc tendance à s'orienter vers des modèles comme UAN ou KLM en laissant de côté les processus cognitifs complexes impliqués dans l'interaction, bien que ceux-ci restent importants. De l'autre côté, les modèles basés sur la théorie de l'activité sont, eux, trop abstraits et donc très difficiles à exploiter en pratique. Enfin, l'interaction instrumentale présente des propriétés génératives grâce à son niveau d'abstraction intermédiaire mais ce modèle reste peu formel

[1]Le terme d'*affordance* ayant fait l'objet de nombreuses controverses dans le domaine de l'interaction homme-machine, nous préférons ici parler d'interprétation plutôt que de définition.

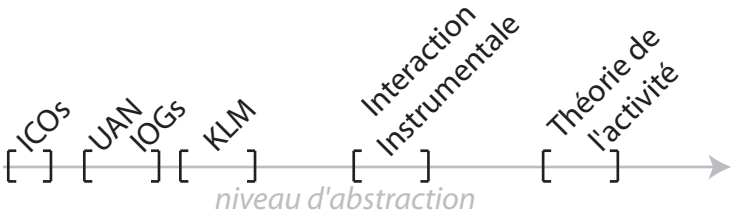


FIG. 2.11 : Les modèles candidats par niveau d'abstraction

et ne permet aucun traitement automatique pour, par exemple, obtenir des prédictions sur l'efficacité d'une technique en contexte. Nous cherchons donc un modèle qui permette de regarder l'interaction à un niveau similaire à celui de l'interaction instrumentale mais avec un degré de formalisation supérieur sans descendre en-dessous du niveau d'abstraction d'UAN.

Aussi, aucun des modèles présentés ne permet d'apprécier l'interaction dans sa totalité. Certains sont centrés uniquement au niveau du périphérique d'entrée quand d'autres sont complètement centrés sur l'utilisateur. Les modèles objet et les modèles basés sur la théorie de l'activité incluent le cœur de l'interaction cependant les premiers sont partiels et ignorent complètement le contexte d'utilisation alors que les seconds sont très vastes et incluent l'environnement au sens large. À l'exception de la théorie de l'activité, aucun de ces modèles ne permet donc de modéliser l'interaction dans son ensemble. Bien que la section suivante démontre l'importance du contexte d'utilisation dans l'interaction, nous cherchons un modèle plus simple que ceux basés sur la théorie de l'activité qui semblent inutilisables car trop vastes et trop abstraits.

2.1.4 CIS : au niveau de la technique et du contexte

Complexity of Interaction Sequences (CIS) est un modèle dont les briques de base sont du même niveau d'abstraction que les modèles UAN et KLM mais qui propose une composition de ces briques selon deux niveaux d'abstraction supplémentaires : celui de la *technique* pour mettre en avant les propriétés et les caractéristiques articulatoires de celle-ci et le niveau du *contexte* afin de prendre en compte certains aspects des processus cognitifs impliqués dans l'interaction. Nous verrons que CIS aborde ces aspects d'une manière simplifiée de façon à conserver un bon compromis entre précision et simplicité.

Au niveau de la technique

Alors qu'UAN est plus centré sur la communication dans l'équipe de conception et les autres modèles précédemment vus sont plus orientés vers la description de la planification d'une tâche ou activité, l'aspect descriptif de notre modèle, Complexity of Interaction Sequences (CIS), a pour but de mettre en avant des propriétés des techniques d'interaction graphique. Dès lors, le niveau de description qui semble approprié est celui de la technique. Au niveau de la technique, CIS décrit les actions que l'utilisateur peut faire dans un état donné de l'interface et comment une technique d'interaction lui permet d'activer une commande de l'application par un enchaînement d'actions. Il ne décrit donc pas comment l'utilisateur

accomplit une tâche mais ce qu'il peut faire. Le paragraphe suivant introduit les différents éléments sur lesquels reposent une description CIS.

CIS décrit l'*état de l'interface* comme un ensemble d'objets que les utilisateurs peuvent manipuler. Certains d'entre eux sont les *objets de travail* [2], des formes géométriques par exemple, tandis que d'autres sont des outils, des items de menus ou des barres d'outils par exemple, ou des instruments dans le sens de l'interaction instrumentale [24], des barres de défilement par exemple. L'*espace des commandes* est l'ensemble des commandes de l'application disponibles à l'utilisateur dans un état donné de l'interface. Une *étape d'interaction* est une séquence d'*actions* qui réduit progressivement l'espace des commandes à une unique commande et qui l'exécute. Ceci mène à un nouvel état de l'interface et donc à un nouvel espace des commandes (par exemple, si un utilisateur vient de créer un fichier alors l'espace des commandes inclut désormais la commande pour effacer ce fichier). Une *technique d'interaction* est un ensemble d'étapes d'interaction que CIS décrit avec un graphe orienté appelé *graphe d'interaction*. Par exemple, la technique d'interaction pour effacer un fichier qui consiste à sélectionner l'icone de ce fichier puis le glisser par drag n'drop vers la corbeille, réduit dans un premier temps l'ensemble des commandes à celles qui sont possibles sur le fichier de l'icone sélectionné (par exemple, les commandes qui consistent à ouvrir des fichiers d'un autre type sont exclues), puis à l'unique commande qui efface le fichier (par exemple, les commandes qui consistent à changer le répertoire parent du fichier sont maintenant exclues). Une toolglass, par exemple, réduit par deux pointages conjoints l'espace des commandes à l'unique commande qui correspond à l'outil sélectionné dans la toolglass et à l'objet sur lequel est appliqué cet outil.

Pour les briques de base, les *actions*, nous avons adopté un niveau similaire à celui des actions UAN ou des keystrokes du modèle KLM. L'originalité de CIS tient à sa façon de composer ces actions au sein d'un *graphe d'interaction*. Une modélisation CIS n'est pas une simple séquence d'actions, elle a pour but d'exprimer la composition articulatoire des actions, principalement motrices, d'une technique d'interaction. CIS a été conçu dans un souci de pouvoir faire des prédictions, nous verrons à la section 3.1.1 que c'est aussi cette propriété qui a motivé le choix des briques de base. Une technique d'interaction fait, bien entendu, entrer en compte plus de paramètres que la simple dimension articulatoire motrice comme nous l'avons montré lors de l'exposé des modèles cognitivistes et de ceux centrés autour de la théorie de l'activité mais celle-ci est néanmoins primordiale et a l'avantage de pouvoir être exprimée simplement. La section 2.2.1 présentera les propriétés des techniques que ce niveau d'abstraction permet d'évaluer.

Au niveau du contexte

Afin de ne pas rester dans un modèle trop simpliste complètement décorrélé d'un contexte réel d'utilisation, nous avons choisi d'opérationnaliser la notion de *contexte* sous forme de *séquence d'interaction*. CIS place l'utilisation d'une technique au sein d'un ensemble ordonné d'interactions. Cet ensemble ordonné est exprimé sous la forme d'une séquence de commandes, elle se situe donc à un niveau d'abstraction supérieur à la séquence de keystrokes de KLM : une séquence KLM décrit une séquence d'actions

[2] Les *objets de travail* correspondent aux *objets d'intérêt* dans les principes de Shneiderman sur la manipulation directe [159]

en dehors de tout contexte plus global alors qu'une séquence CIS décrit l'interaction au niveau des objets de l'interface dans le contexte d'une activité telle que créer un document. Les actions qui permettent de compléter une séquence d'interaction CIS se trouvent dans les graphes des techniques qui permettent d'activer les commandes de la séquence.

La notion de séquence d'interaction introduit une dimension cognitive. Cependant, ce qui est traduit ici ne correspond pas aux mécanismes cognitifs et aux différentes influences de l'environnement qui interviennent dans l'interaction mais au *résultat* de ceux-ci. Une séquence d'interaction peut être vue comme la *trace* d'un processus cognitif qui s'est déroulé, cette trace étant le reflet d'un contexte d'utilisation. Contrairement aux méthodes GOMS qui décrivent l'*organisation* des actions cognitives mises en oeuvre, CIS ne considère que le *résultat* de celles-ci dans une séquence d'interaction. En effet, nous verrons à la fin de cette section que des études empiriques ont montré qu'une partie des processus cognitifs sont capturés par l'organisation des interactions dans une séquence. En ne considérant que le *résultat de cette organisation* plutôt que *les raisonnements qui mènent à cette organisation*, CIS ne modélise pas toute la complexité des processus cognitifs de l'utilisateur mais ne les néglige pas tout en restant simple à comprendre et utiliser.

2.2 Le modèle CIS

Dans cette section, nous introduisons les différents éléments d'une modélisation CIS en nous appuyant sur le cas d'école suivant : l'interface d'un éditeur graphique simple permettant, notamment, de créer des triangles, rectangles et ellipses de taille prédéfinie. L'espace des commandes de notre exemple contient donc des commandes de la forme :

– $(t \in \{\text{Create_Triangle, Create_Rectangle, Create_Ellipse}\}, p \in \text{position})$

2.2.1 Décrire une technique d'interaction

Les briques de base : Les actions CIS

Les éléments, arcs et noeuds, d'un graphe d'interaction CIS sont des *actions*. Nous distinguons deux types d'action, les *acquisitions* et les *validations*.

– Une acquisition, représentée par un arc, identifie un sous-ensemble de l'espace des commandes courant. Il s'agit typiquement d'un déplacement du curseur vers un outil, un objet de travail ou une position. La figure 2.12 montre la représentation graphique d'une action d'acquisition : le déplacement du curseur vers l'outil de création de triangle identifie les commandes de la forme $(\text{Create_Triangle}, p \in \text{position})$. La tête de la flèche est étiqueté par le ou les objets graphiques déplacés et son autre extrémité par la position ou l'objet cible vers lequel les objets sont déplacés.
– Une validation, représentée par un noeud, confirme le sous-ensemble identifié par une acquisition qui devient maintenant l'espace des commandes courant. Il s'agit typiquement d'un clic de souris ou de l'utilisation d'une touche du clavier. La figure 2.13 correspond à un clic de souris sur l'outil de création de triangle réduisant effectivement l'espace des commandes à l'ensemble des commandes de la forme $(\text{Create_Triangle}, p \in \text{position})$.

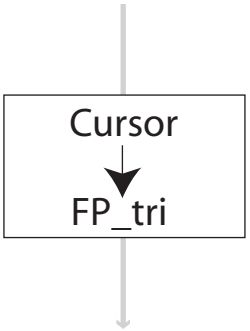


FIG. 2.12 : Une acquisition CIS : le pointage d'un outil de création de triangle à l'aide du curseur

FIG. 2.13 : Une validation CIS : un clic sur l'outil de création de triangle

Les acquisitions qui décrivent des mouvements d'objets de l'interface sont très proches des actions M (mouvement) d'UAN ou des opérateurs P (pointage) de KLM. Notons la différence entre ces deux notations : le P de KLM est un simple symbole désignant un déplacement générique du curseur alors que l'action M d'UAN peut être enrichie par la cible vers laquelle le curseur se déplace ou le fait qu'un objet suit les déplacements du curseur (cf figure 2.7). Comme UAN, CIS opte pour un niveau de description un peu plus informatif, les informations supplémentaires étant également utiles pour prédire la performance d'une technique.

Pour les validations, CIS est, au contraire, moins détaillé qu'UAN et se situe plus du côté des *keystrokes* de KLM qui décrivent génériquement les actions que nous qualifierons de ponctuelles (qui prennent un temps moyen constant) telles qu'une frappe au clavier ou un appui de bouton. Par exemple, avec UAN, un appui de bouton et un clic de bouton auront deux modélisations différentes, Mv et MvM^, alors qu'avec CIS, ces deux actions seront représentées par un même nœud correspondant au K (keystroke) de KLM. Nous avons jugé les descriptions UAN trop ad hoc, spécifiquement définies pour un style d'interaction parfois qualifié de "point-and-click" alors que des techniques d'interaction récentes comme celles basées sur le franchissement [10] ne sont justement pas de ce type. Nous avons donc fait le choix de les laisser ouverts à des espaces de conception plus larges, ce que nous exposerons dans la partie 3.2.

La composition des briques : Les graphes d'interaction CIS

Abordons maintenant la composition de ces différentes actions en étudiant la construction des graphes d'interaction. La racine d'un graphe est étiquetée par le nom de la technique. Un chemin de la racine à un noeud modélise une *étape d'interaction* (notion introduite dans la section 2.1.4) correspondant à une séquence d'actions. Par exemple, le chemin le plus à gauche dans le graphe de la figure 2.14 décrit l'étape d'interaction suivante : déplacer le curseur vers l'outil de création de triangles - cliquer - déplacer le curseur vers la position à laquelle le triangle doit être créé - cliquer.

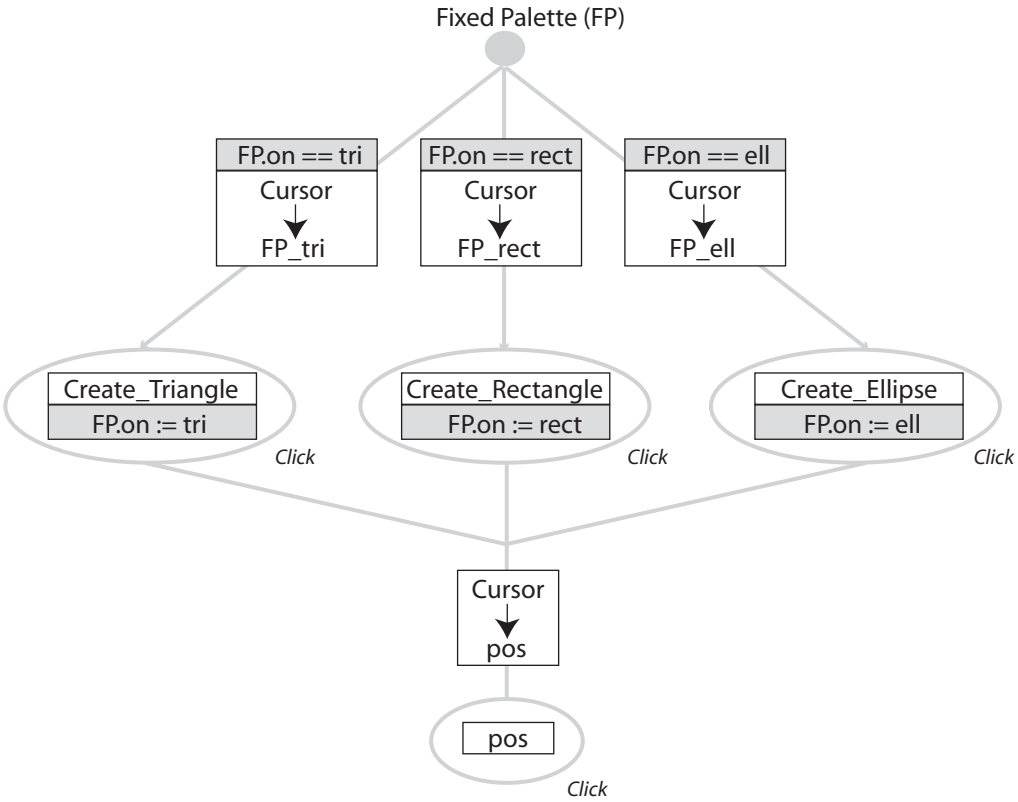


FIG. 2.14 : Le graphe d'interaction d'une palette fixe

Dans le graphe d'interaction, chaque nœud (à l'exception de la racine) est une action de validation étiquetée par l'action correspondante (clic par exemple) et les champs de la manipulation qu'il instancie. Si nous reprenons le chemin le plus à gauche comme exemple, il instancie en premier lieu le champ t (le type), réduisant l'espace d'interaction aux commandes de la forme $(\text{Create_Triangle}, p \in \text{position})$. Puis, en second lieu, il instancie le champ pos lorsque l'utilisateur clique à une position, réduisant ainsi l'espace d'interaction à la seule commande (Create_Triangle, pos). L'espace des commandes de l'application graphique est ainsi réduit à une unique commande, ce qui a pour conséquence de l'exécuter : un triangle est créé à la position pos et l'état de l'interface s'en trouve. Un graphe modélise donc la façon

dont une technique propose différentes étapes d'interaction à l'utilisateur.

Les nœuds peuvent avoir des *effets de bord* qui permettent de décrire les changements d'état de l'interface autres que les positions des objets qui sont modifiées par les arcs (acquisitions). Par exemple, cliquer sur un outil dans une palette sélectionne cet outil pour les futures actions. Un effet de bord se traduit par une étiquette supplémentaire sur le noeud en question (par exemple, "FP.on :=tri" du noeud Create_Triangle dans la figure 2.14). Les arcs peuvent avoir des *préconditions* qui, lorsqu'elles sont vérifiées, rendent optionnelles les actions d'acquisition et de validation associées. Par exemple, si l'outil pour créer des triangles est déjà sélectionné, les actions de pointage de l'outil et le clic associé ne sont pas nécessaires : un unique clic à la position désirée crée un nouveau triangle. Une précondition se traduit par une étiquette supplémentaire sur l'arc, qui devient optionnel (par exemple, "FP.on==tri" de l'arc $Cursor \rightarrow FP_tri$ dans la figure 2.14). Il est enfin possible de lier deux arcs pour représenter l'exécution parallèle de deux actions afin de pouvoir décrire des techniques bimanuelles comme la toolglass évoquée dans l'introduction et décrite par le graphe de la figure 2.16.

Propriétés morphologiques d'une technique

La description de techniques d'interaction par des graphes CIS permet de mettre en évidence les propriétés morphologiques de ces techniques. Ici, nous présentons un ensemble de propriétés faciles à appliquer pour comparer rapidement, de façon qualitative, des techniques d'interaction à partir de leurs graphes.

- *Organisation temporelle* :
  Une technique d'interaction impose une organisation séquentielle ou parallèle de ses actions constituantes qui est visualisée par la forme des graphes d'interaction et l'utilisation de la construction parallèle. En conservant le même exemple de création de formes graphiques, nous pouvons décrire une technique qui spécifie d'abord la position, puis l'outil à appliquer, par exemple avec un menu contextuel contenant les trois outils de création invoqué par un clic de souris à la position désirée. La figure 2.15 montre les graphes d'interaction de ces deux techniques.
  Toujours sur le même exemple de l'éditeur graphique simple, la toolglass se caractérise par deux acquisitions parallèles et une seule validation. La figure 2.16 montre le graphe d'interaction de la toolglass : les arcs correspondant aux acquisitions parallèles sont reliés par une double ligne.
- *Persistance* :
  Nous avons vu à la section précédente que les techniques d'interaction peuvent avoir des effets de bord comme, par exemple, changer la valeur des attributs des objets de l'interface. Ces effets de bord peuvent affecter comment la technique d'interaction sera utilisée les prochaines fois. L'utilisation conjointe des effets de bord et des préconditions traduit la propriété de persistance de certaines actions. Par exemple, l'outil sélectionné dans une palette est persistant et donc, par exemple, créer deux triangles en série ne requiert qu'une unique sélection de l'outil pour créer des triangles. D'autres techniques d'interaction proposent de la persistance sur l'objet de travail plutôt que sur l'outil. Ainsi, dans la plupart des éditeurs graphiques de dessin vectoriel, il est possible de sélectionner une forme et de lui appliquer successivement un outil pour changer la couleur de fond

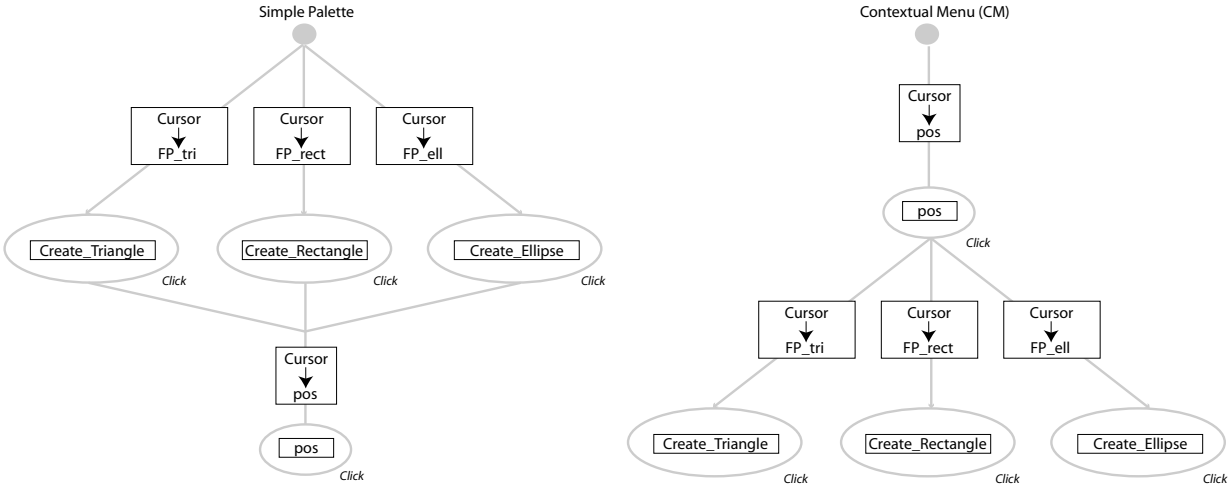


FIG. 2.15 : Ordre des actions dans une technique. À gauche : une palette d'outils ; à droite : un menu contextuel

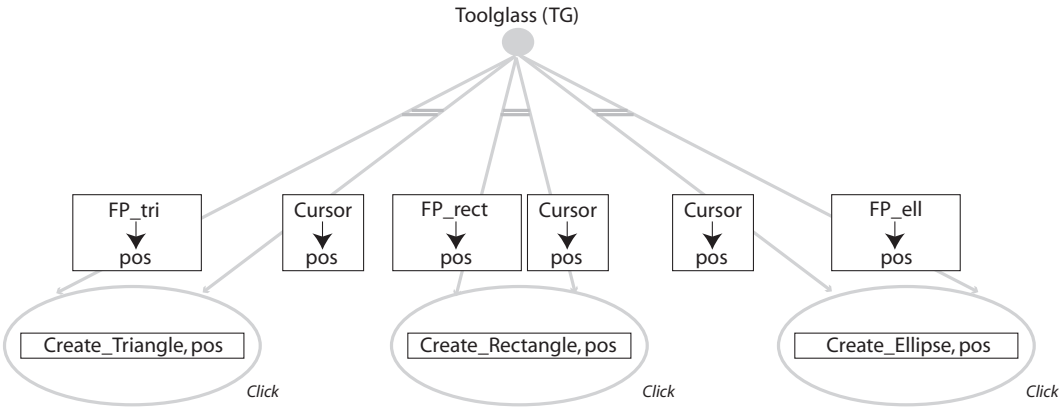


FIG. 2.16 : Parallélisme des actions dans une technique : la toolglass

puis un autre pour changer la couleur de son contour. A l'inverse, les toolglasses telles qu'elles ont été décrites dans [101], n'ont pas de propriété de persistance.

– *Fusion* :
Certains outils utilisent les principes d'intégralité [97] pour manipuler plusieurs attributs en même temps. Les principes d'intégralité/séparabilité de Jacob et al. encouragent le contrôle combiné de différentes dimensions dès lors que celles-ci peuvent être perceptuellement intégrées comme une seule dimension par l'utilisateur. La partie droite de la figure 2.17 montre le graphe d'interaction d'une technique que l'on rencontre dans les applications du pack Microsoft Office : un outil qui permet d'appliquer deux commandes simultanément pour changer l'épaisseur et le style d'une forme graphique simultanément. Dans un tel graphe, un nœud porte une étiquette faite de plusieurs commandes (ici, $width$ et $style$).

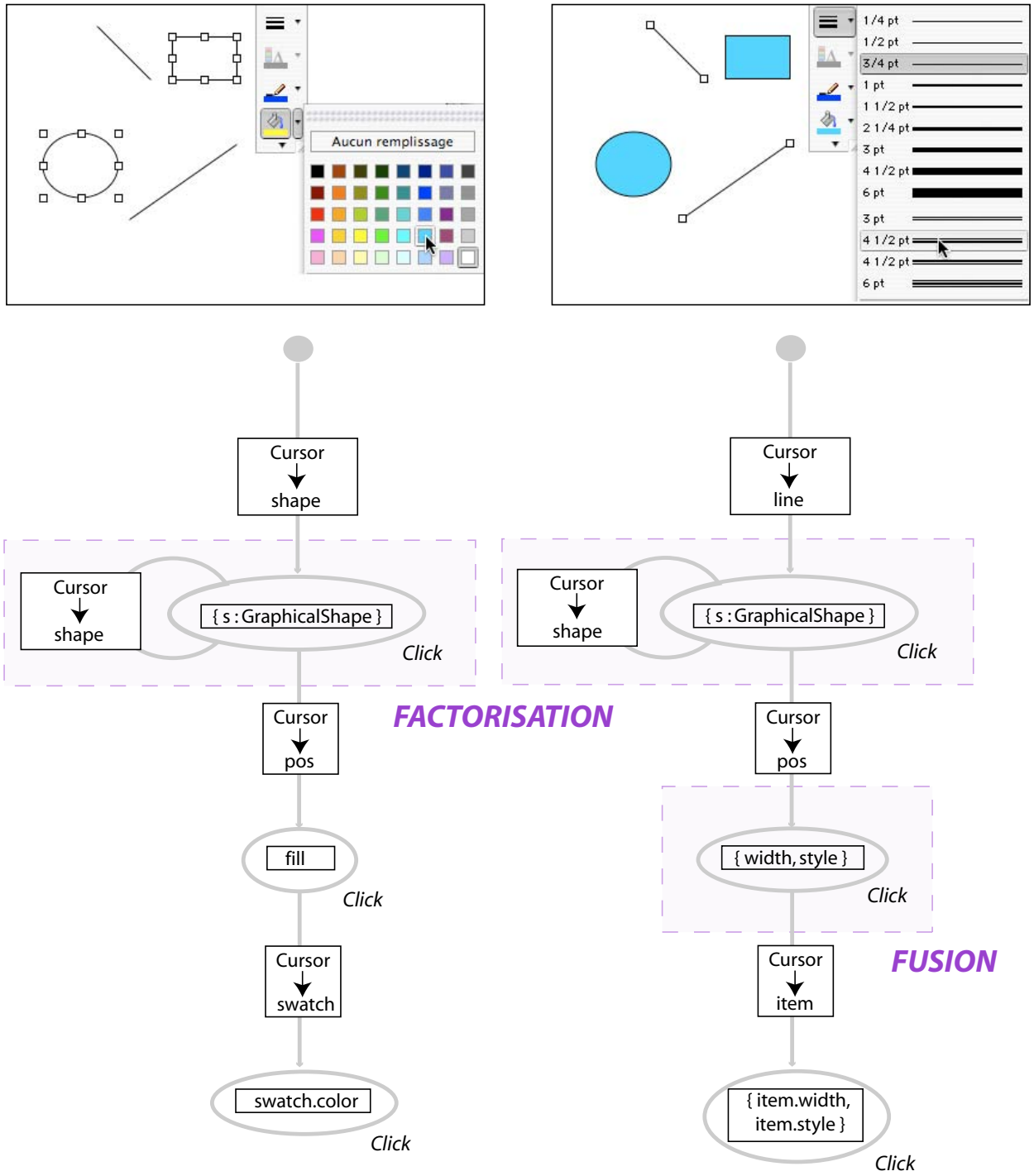


FIG. 2.17 : Propriétés de factorisation et de fusion. À gauche : appliquer la commande $(fill, s : GraphicalShape, c : Color)$ à plusieurs objets de travail à la fois ; à droite : appliquer deux commandes $(width, s : GraphicalShape, w : Integer)$ et $(style, s : GraphicalShape, s : Style)$ à plusieurs objets de travail.

– *Développement* et *Factorisation* :
La plupart des outils du pack Microsoft Office permettent également de modifier plusieurs objets de travail simultanément. Par exemple, beaucoup d'outils de dessin permettent l'acquisition en série de plusieurs formes en maintenant la touche $SHIFT$ enfoncée puis d'appliquer une commande à toutes ces formes simultanément comme illustré par la figure 2.17. Cette factorisation est traduite par une boucle sur le nœud qui instancie la forme graphique à modifier. D'autres interfaces permettent à l'utilisateur de créer plusieurs copies d'un outil "instancié". Par exemple, l'interface HabilisDraw [8] permet de disposer des pots de peinture de différentes couleurs un peu partout sur la surface de dessin. Ceci permet d'économiser des actions en ayant des outils "tout prêt" sous la main mais il ne faut pas négliger l'espace écran supplémentaire ainsi consommé. De telles techniques se caractérisent des graphes très larges et peu profonds.

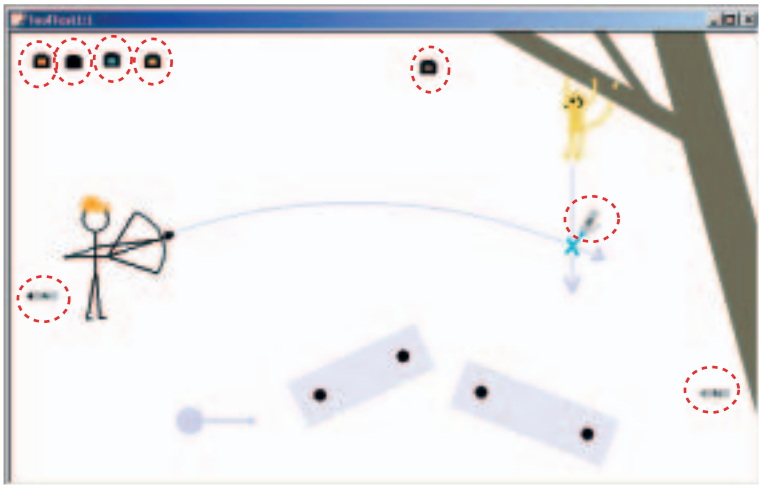


FIG. 2.18 : Propriété de développement. (*Illustration extraite de [8]*, les outils ont été entourés par nous)

2.2.2 Opérationnaliser un contexte d'utilisation

Contexte cognitif : théories et observations empiriques

Ici, nous présentons les travaux qui ont directement influencé l'introduction de la notion de contexte d'utilisation dans CIS. Il s'agit des dimensions cognitives de Green [72] qui ont été initialement introduites dans le cadre du développement de programmes et dont certaines ont été étudiées par Mackay [116] dans le cadre des interfaces graphiques.

Le "Cognitive Dimensions Framework" de Green est une approche générale pour évaluer l'utilisabilité d'un artefact d'information, c'est-à-dire un système qui stocke, affiche et manipule de l'information. L'idée est d'introduire un vocabulaire bien défini pour parler du concept d'utilisabilité. Pour cela, il définit six types d'activité (*incrémentation*, *transcription*, *modification*, *conception exploratoire*, *recherche* et *compréhension exploratoire*) et un ensemble de dimensions telles que la *viscosité* (résistance d'un système au changement) ou la *visibilité* de l'information. Evaluer l'utilisabilité d'un système de fait en

deux étapes. D'abord, il faut décider quel type d'activité le système doit supporter. Chaque activité ayant ses propres besoins en termes de dimensions cognitives (voir le tutorial [73] pour une vue d'ensemble), la seconde étape consiste donc à regarder le système sous chacune de ces dimensions. Par exemple, il n'est pas grave que la dimension d'*engagement prématuré* qui traduit le nombre de choix prématurés qu'impose le système soit grande pour l'activité de *transcription* (recopie d'un document par exemple) alors qu'il est primordial qu'elle soit réduite au maximum pour une activité de *conception exploratoire*. En effet, dans le cas d'une activité de transcription, la charge cognitive demandée par l'activité elle-même est faible, ce qui permet à l'utilisateur de planifier loin et de juger l'intérêt de choix faits en amont. Les dimensions cognitives ont été initialement introduites dans le cadre de l'évaluation des langages de programmation mais, dans [72], leur applicabilité a été démontrée sur d'autres systèmes comme un téléphone pour lequel, par exemple, l'activité de transcription (composer un numéro de téléphone lu) est très fréquente. Cette approche peut être appliquée dès lors que nous sommes en présence d'un langage d'interaction ou notation, d'un environnement pour éditer cette notation et d'un "medium" d'interaction (un dispositif d'affichage par exemple). Ces trois éléments étant toujours présents dans les interfaces graphiques, les dimensions cognitives sont une méthode d'évaluation pertinente. Par exemple, si on traduit ces termes dans le cadre d'un éditeur de dessin standard : le langage d'interaction comporte les commandes d'édition graphique dans un environnement disposant des mediums d'interaction que sont typiquement souris, clavier et écran. Un des messages très important sur lequel insiste Green est qu'il n'existe pas d'approche qui satisfasse tous les types d'activité et donc qu'un système ne peut être évalué que sous certaines dimensions, les dimensions d'intérêt.

Alors que les techniques de visualisation sont de plus en plus fréquemment évaluées pour différentes tâches au sein d'une même expérience et que les recommandations sur la prise en compte du contexte sont nombreuses en IHM ([56] par exemple), il existe, malheureusement, très peu d'études empiriques qui tiennent compte du type d'activité lors de l'évaluation des techniques d'interaction. Pourtant, il est de plus en plus courant, en visualisation d'informations, de rencontrer des évaluations sur des tâches de type "benchmark" créées de façon ad hoc et les résultats de ces évaluations rapportent que les performances qui dépendent de la tâche (voir [142], [103], etc.). Pour les techniques d'interaction, une des seules études empiriques, à notre connaissance, qui se soit directement inspirée du cadre des dimensions cognitives pour évaluer des interfaces graphiques est celle menée par Mackay [116]. Cette étude a comparé l'efficacité de trois techniques d'interaction (toolglasses [101], palettes flottantes et marking menus [104]) utilisées dans l'interface de CPN2000 [26], un éditeur graphique de réseaux de Petri colorés. Les utilisateurs devaient utiliser ces techniques pour accomplir deux tâches : une tâche de recopie (*copy*) dans laquelle ils devaient recopier un réseau qui leur était fourni et une tâche de résolution de problème (*problem solving*) dans laquelle ils devaient trouver les modifications correctes à apporter à un réseau donné pour qu'il réponde à certaines spécifications. Ces deux tâches correspondent aux activités de transcription et d'incrémentation de Green. Les résultats de cette expérience montrent que les performances de chaque technique dépendent du contexte cognitif. Dans un contexte de recopie, les utilisateurs sont plus efficaces avec les palettes flottantes alors que, dans un contexte de résolution de problème, ils sont plus efficaces avec les toolglasses et les marking menus. De plus, leur préférence qualitative est en accord avec les résultats quantitatifs. En d'autres termes, ce travail démontre qu'on ne peut affirmer qu'une technique est "meilleure" qu'une autre de façon absolue. Les comparaisons doivent être faites de façon relative à un contexte d'utilisation afin que les résultats puissent être appliqués à des problèmes réels.

Contexte de recopie	Contexte de résolution de problème
1. $(Create_Triangle(100, 100))$ 2. $(Create_Triangle(700, 300))$ 3. $(Create_Triangle(100, 700))$ 4. $(Create_Ellipse(300, 100))$ 5. $(Create_Ellipse(300, 300))$ 6. $(Create_Ellipse(300, 700))$ 7. $(Create_Rectangle(700, 100))$ 8. $(Create_Rectangle(100, 300))$ 9. $(Create_Rectangle(700, 700))$	1. $(Create_Triangle(100, 100))$ 2. $(Create_Ellipse(300, 100))$ 3. $(Create_Rectangle(700, 100))$ 4. $(Create_Triangle(700, 300))$ 5. $(Create_Ellipse(300, 300))$ 6. $(Create_Rectangle(100, 300))$ 7. $(Create_Triangle(100, 700))$ 8. $(Create_Ellipse(300, 700))$ 9. $(Create_Rectangle(700, 700))$

FIG. 2.19 : Deux séquences d'interaction. À gauche : recopie ; à droite : résolution de problème.

Nous avons été influencés par ce cadre d'étude qui lie intimement utilisabilité et contexte d'utilisation afin que CIS aide à comprendre les relations entre l'efficacité d'une technique, son contexte d'utilisation et ses propriétés.

Contexte cognitif : Les séquences d'interaction CIS

Nous introduisons la notion de *séquence d'interaction* qui permet de modéliser le contexte d'utilisation avec CIS. Cette notion peut-être vue comme l'instanciation d'un type d'activité pour l'application graphique que l'on cherche à étudier.

Dans l'idée d'atteindre un but, que nous définissons comme un état de l'interface que l'utilisateur cherche à atteindre, l'utilisateur peut choisir différentes techniques d'interaction et différentes séquences d'interaction. Ces choix sont guidés par l'état courant de l'interface, les techniques qui sont à sa disposition, la connaissance qu'il a de cette interface et la quantité de commandes qu'il est en mesure de planifier. Certaines activités (au sens de Green), comme la recopie, demandent une faible charge cognitive et permettent donc aux utilisateurs de planifier bien à l'avance leurs futures interactions alors que d'autres, comme la résolution de problème, sont plus difficiles et les utilisateurs les abordent de manière plus incrémentale [116]. Nous définissons le contexte d'utilisation comme la combinaison de l'état courant de l'interface et la quantité de planification que l'utilisateur peut faire. En termes CIS, ceci se traduit par une séquence de commandes appelée *séquence d'interaction* qui est plus ou moins longue. L'état courant de l'interface dépend des interactions précédentes alors que la planification des interactions futures dépend de l'activité courante de l'utilisateur. Le contexte d'utilisation se trouve *opérationnalisé* par une séquence d'interaction dès lors que nous pouvons donner une séquence d'interaction représentative d'un type d'activité pour une interface graphique donnée.

Nous illustrons la notion de séquence d'interaction dans le cadre de l'étude empirique de Mackay mentionnée à la section précédente. Cette étude a notamment rapporté que, dans un contexte de recopie, les utilisateurs ont tendance à créer les objets de même type en séquence alors que, dans un problème de résolution de problème, ils organisent leurs interactions spatialement en progressant incrémentalement à partir d'un point d'attention. Bien que l'organisation des interactions soit différente, ces deux contextes peuvent mener au même état final de l'interface. Nous pouvons donc opérationnaliser ces deux contextes par les séquences Seq_{copy} et $Seq_{problemsolving}$ de la table 2.19. Seq_{copy} est obtenue en regroupant les créations avec des outils de même type alors que $Seq_{problemsolving}$ est obtenue en organisant les commandes qui minimisent les distances à partir d'un point d'attention (ici, en haut à gauche).

Les dimensions cognitives de Green n'ont pas été conçues pour fournir des analyses détaillées mais comme une façon de décrire la structure de l'information. Elles peuvent bien entendu être la source d'inspiration pour des analyses détaillées mais un des buts de ce cadre d'étude est d'être simple en offrant des concepts clairs et nommés afin de disposer d'un vocabulaire analytique. La notion de séquence d'interaction permet d'instancier ce cadre d'étude pour les GUI, les activités d'intérêt pouvant être représentées par des séquences d'interaction type. Dans le chapitre suivant, nous abordons l'aspect prédictif de CIS et notamment comment nous sous servons de ces séquences d'interaction pour mesurer l'efficacité des techniques d'interaction.

2.3 Conclusion

Rappelons que nous cherchons à mettre en place un modèle qui soit utile pendant la phase de conception. Dans le cadre de notre approche générative, ce modèle doit permettre de décrire, évaluer pour faire des choix et favoriser la génération techniques alternatives. Nous voulons donc un modèle qui permette à la fois de décrire l'interaction assez finement afin de pouvoir prédire l'efficacité d'une technique pour faire des choix et, en même temps, nous voulons garder un niveau d'abstraction assez haut pour laisser des espaces de conception ouverts. Nous avons également démontré que l'efficacité d'une technique d'interaction ne peut être décorrélée de son contexte d'utilisation et que notre modèle doit donc décrire l'interaction assez largement pour inclure le contexte d'utilisation.

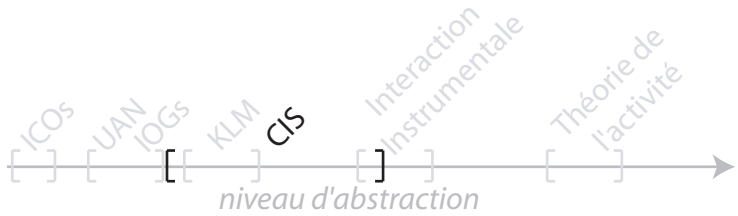


FIG. 2.20 : Le niveau d'abstraction de CIS

CIS, le modèle que nous proposons dans cette thèse, a été conçu pour répondre à ces besoins. CIS propose de modéliser l'interaction à 3 niveaux différents et complémentaires :

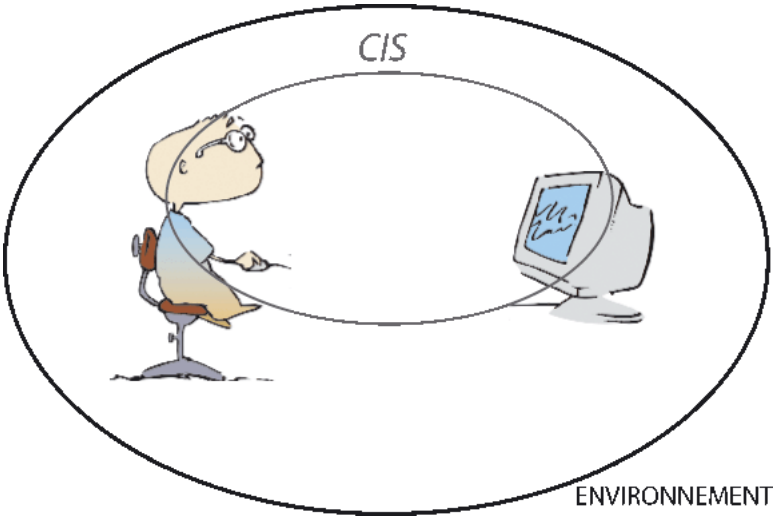


FIG. 2.21 : Sous-ensemble de l'interaction décrite avec CIS

- Au niveau de l'*action*, brique de base de CIS. Ce niveau d'abstraction est du même ordre que celui des actions UAN et des keystrokes KLM. En effet, une action CIS est plus détaillée qu'un keystroke KLM mais moins détaillée que les actions UAN. Par exemple, l'opérateur P de KLM désigne un pointage générique alors qu'un arc CIS décrit les changements sur les objets de l'interface, qui représentent un sous-ensemble du feed-back spécifié par une action UAN. Ce niveau de détail permet de faire de CIS un modèle prédictif (comme KLM) mais CIS ne spécifie pas l'instanciation de ces actions afin que le concepteur puisse envisager diverses possibilités (contrairement à UAN). Le chapitre suivant (3) détaille ces deux points.
- Au niveau de la *technique*, graphe d'interaction. Les graphes d'interaction de CIS décrivent l'articulation de ces actions. Ils permettent de regarder l'interaction à un niveau d'abstraction similaire à celui de l'interaction instrumentale puisqu'ils décrivent l'enchaînement des actions sur les objets au sein d'un cadre sémantique, celui de la technique. Ce niveau d'abstraction permet de mettre en avant des propriétés comparatives à la manière de l'interaction instrumentale pour comparer et faciliter l'exploration d'un espace de conception. Nous reviendrons également sur l'exploration de cet espace au chapitre suivant 3.
- Au niveau du *contexte*, séquence d'interactions. CIS permet d'opérationnaliser un contexte d'utilisation sous la forme d'une séquence d'interactions qui peut être vue comme la trace d'un processus cognitif lors d'une activité. Ce niveau de description nous sera utile pour prédire les performances des techniques relativement à un contexte d'utilisation.

Les figures 2.21 et 2.20 résument la position de CIS sur les critères que nous avons utilisés tout au long de ce chapitre : (1) le niveau d'abstraction de CIS et (2) la partie de l'interaction qui peut être décrite avec CIS.

Explorer l’espace de conception avec SimCIS et CIS

Alors que, jusqu'ici, nous nous sommes concentrés sur la description de techniques d'interaction, nous allons maintenant nous intéresser aux outils dont dispose notre concepteur pour juger de la qualité de ce qu'il a décrit. En effet, les modèles descriptifs ont des buts de communication et d'analyse des caractéristiques des techniques et des tâches mais ils ne nous renseignent pas précisément sur la performance d'une technique dans un contexte d'utilisation donné. Les deux grandes approches, non exclusives, pour évaluer quantitativement une application graphique sont les expérimentations contrôlées et les modèles prédictifs. Dans le premier cas, les concepteurs prototypent leur interface et réalisent directement des expériences auprès des utilisateurs. Ce processus est bien évidemment itératif : les résultats des expériences sont utilisés pour améliorer, valider ou remettre en question des choix de l'interface évaluée. Cependant, les expérimentations contrôlées sont très gourmandes à la fois en termes de compétences et de temps et interviennent dans les phases avancées de conception puisque la technique à évaluer. Afin de répondre aux besoins des premières phases de conception, les concepteurs ont besoin de modèles pour prédire l'efficacité d'une technique. Ces modèles ne sont pas destinés à se substituer aux expériences auprès des utilisateurs mais à gagner du temps sur celles-ci en réalisant un travail en amont pour faire des choix et des appréciations préliminaires. Pour que ces outils soient adoptés, il faut bien sûr que le temps de modélisation soit largement inférieur au temps qui serait passé à faire des études empiriques tout en produisant des prédictions fiables, ce qui n'est malheureusement pas le cas avec les modèles existants. Nous proposons le simulateur SimCIS qui, à partir d'une séquence d'interaction et d'un graphe d'interaction, simule l'utilisation de la technique décrite par le graphe pour la séquence afin de fournir des prédictions sur l'efficacité d'une technique en contexte. Comme tous les autres modèles prédictifs, CIS et SimCIS ne remplacent pas les observations empiriques mais sont des outils pour choisir des techniques et des tâches expérimentales pertinentes.

Nous abordons également dans ce chapitre comment CIS et SimCIS sont des outils qui favorisent la *génération* de techniques d'interaction. L'approche proposée diffère considérablement des recherches sur la génération automatique de techniques d'interaction. En effet, les traitements *automatiques* excluent le concepteur du processus de génération et présentent donc un degré d'innovativité limité. Au contraire, CIS et SimCIS sont des outils pour susciter l'imagination des concepteurs d'interface en ouvrant un espace de conception muni de mesures. D'une part, le concepteur peut, à partir d'un graphe d'interaction existant, identifier les différentes actions sur lesquelles agir pour optimiser une technique. D'autre part, il peut choisir de réfléchir sur les propriétés que doit avoir une technique en travaillant au niveau de la structure du graphe pour ensuite instancier cette structure en une technique d'interaction.

3.1 SimCIS : prédire l'efficacité d'une technique en contexte

3.1.1 Les lois empiriques

SimCIS s'appuie sur les lois empiriques sous-jacentes à l'interaction des applications graphiques. Les trois lois les plus connues et les plus utilisées en Interaction Homme-Machine sont respectivement les lois de Fitts, de Hick-Hyman et de "steering-path". La loi de Fitts [64] évalue la difficulté et le temps de mouvement pour le pointage d'une cible. La loi de Hick-Hyman [88, 94] évalue le temps de choix d'un élément dans un ensemble en fonction du nombre d'éléments de l'ensemble. La loi de steering-path [2], plus récente, évalue le temps de suivi d'une trajectoire. Ces trois lois réunies permettent de couvrir la

majorité des interactions dans une application graphique et motivent le niveau d'abstraction des briques de base du modèle CIS.

La loi de Fitts

La loi de Fitts [64] est de loin la plus célèbre des lois empiriques utilisées en IHM. Elle permet de prédire le temps de mouvement pour atteindre une cible d'une taille donnée à une distance donnée. Elle s'exprime sous la forme $MT = a + b \times log_2(1 + \frac{D}{W})$ où D et W sont respectivement la distance à la cible et la taille de la cible (figure 3.1) et MT est le temps de mouvement. a et b sont des constantes à déterminer empiriquement et qui dépendent de différents facteurs comme le périphérique d'entrée utilisé. Le terme $log_2(1+\frac{D}{W})$ représente l'Indice de Difficulté (ID) de la tâche de pointage. Il peut être interprété en termes de quantité d'information que le mouvement doit fournir au système afin que celui-ci soit en mesure d'identifier une cible parmi l'ensemble des cibles affichées.

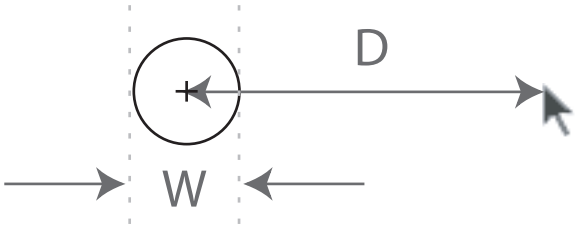


FIG. 3.1 : Pointage d'une cible

Initialement observée par Fitts [64] dans le monde physique pour un participant qui atteint un objet avec son doigt selon une dimension, elle a été introduite dans le monde virtuel pour évaluer les techniques de pointage dans le cadre des interfaces homme-machine par Card et al. [45]. Elle a été ensuite étudiée intensivement pour évaluer des périphériques d'entrée pour le pointage [118, 120]. Puis, les recherches les plus récentes se sont focalisées principalement autour de deux grands thèmes : le pointage en deux dimensions et le pointage dans des espaces multi-échelles.

La volonté de modéliser finement le pointage en deux dimensions de cibles non circulaires a fait émerger des modèles plus complexes qui incluent plus de variables. Ainsi, MacKenzie et Buxton [119] ont proposé deux modèles pour le pointage d'une cible rectangulaire : l'un prend en compte la "largeur visible" de la cible (basée sur l'angle du mouvement) alors que l'autre modèle est plus simple et prend pour taille effective le minimum de la largeur et de la hauteur. Le modèle ayant le plus haut coefficient de corrélation est le second et il a, de plus, l'avantage de rester simple : $MT = a+b\times log_2(1+\frac{D}{min(W,H)})$. Accot et Zhai [5] proposent un modèle un peu plus compliqué, $MT = a+b\times log_2(1+\sqrt{(\frac{D}{W})^2 + \eta(\frac{D}{H})^2})$, dans lequel les influences de la hauteur et de la largeur sont pondérées indépendamment. Ainsi, contrairement au modèle de MacKenzie et Buxton, l'ID ne reste pas le même si on intervertit largeur et hauteur mais au

prix d'une constante supplémentaire à déterminer empiriquement (η). Enfin, Grossman et Balakrishnan [75] adoptent une approche probabiliste pour proposer un modèle qui inclut l'angle du mouvement et qui peut s'appliquer à n'importe quelle forme en deux dimensions. Cependant, l'expression de ce modèle, une double intégrale sur la surface, est nettement plus compliquée et les constantes à déterminer empiriquement plus nombreuses.

D'autres études ont montré que la loi de Fitts, en restant dans des formulations simples, est également pertinente dans le cadre la navigation multi-échelles [82, 77, 78]. Il s'agit d'étudier le passage à l'échelle de la loi de Fitts pour des mondes virtuels dans lesquels l'utilisateur peut naviguer selon la dimension d'échelle (au moyen de la molette de la souris ou avec un second périphérique d'entrée par exemple). Dans ce type d'interface, Guiard et al. [78] ont ainsi identifié deux types de pointage : le "cursor pointing" pour lequel l'utilisateur a simplement besoin de déplacer son curseur vers un objet visible dans la vue courante, sans manipuler l'échelle, et le "view pointing" qui consiste à modifier la vue par "pan" et "zoom" afin de rendre visible une partie donnée du monde virtuel. Les auteurs ont observé que ce second type de pointage peut être modélisé en utilisant la loi de Fitts appliquée à la taille de la vue.

La loi de Hick-Hyman

La loi de Hick-Hyman [88, 94] modélise le choix d'un élément parmi un ensemble d'éléments représentant différents stimuli. Le temps de réaction, RT, est une fonction linéaire du logarithme du nombre d'éléments n dans l'ensemble : $RT = a + b \times log_2(n)$. Ce modèle a notamment été utilisé en interaction homme-machine pour expliquer le temps de choix d'une méthode parmi plusieurs dans le cadre d'applications pour des tableurs [134] ou encore pour expliquer le temps de choix d'un item dans un menu [107]. Cependant, son utilisation en interaction homme-machine est restée plus limitée que celle de la loi de Fitts. Seow [158] explique ce constat par la complexité des stimuli dans les interfaces graphiques et le fait que l'utilisateur n'atteint pas un niveau de familiarité avec les interfaces tel qu'il connaît la mise en page graphique de tête.

La loi de suivi de trajectoire et de franchissement de but

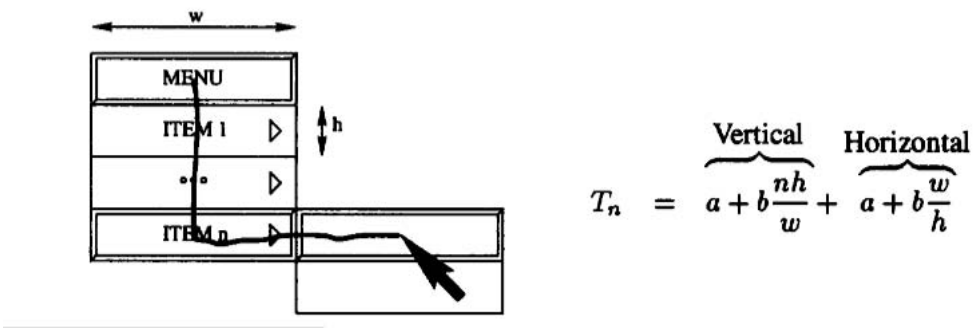


FIG. 3.2 : Suivi de chemin et sélection du n^e item dans un menu (*Illustration extraite de [2]*)

Certaines techniques d'interaction obligent l'utilisateur à suivre une trajectoire avec leur curseur. Pour modéliser ces interactions, Accot et Zhai ont introduit la loi de "steering path". Dans leur protocole expérimental [2], ils commencent par établir un modèle pour décrire un simple franchissement de but, et montrent qu'il suit la même loi que le pointage. Franchir un but de largeur W à une distance D prend un temps linéaire à l'indice de difficulté $ID = log_2(1 + \frac{D}{W})$. Dans un second temps, ils valident le fait de modéliser un suivi de trajectoire comme une succession de franchissements. Par exemple, suivre un couloir rectiligne de longueur D et de largeur uniforme W prend un temps $T = a + b \times \frac{D}{W}$. Ce modèle peut donc être utilisé pour modéliser les actions élémentaires de l'interaction avec un menu comme l'illustre la figure 3.2.

La première expérience sur le franchissement a été menée par Accot et Zhai [4] afin de comparer les valeurs des constantes des modèles de pointage et de franchissement. Ils ont identifié que la valeur des constantes était fonction de la direction du but à franchir ou de la cible à atteindre par rapport à celle du mouvement (colinéaire ou orthogonale) et du type des interactions (discrètes ou continues). Leur résultat est que, selon la direction et le type des interactions, le franchissement est plus rapide que le pointage (Figure 3.3). En s'appuyant sur ces résultats encourageants, Apitz et Guimbretière ont conçu crossY [10], un éditeur de dessin dans lequel chacun des widgets classiques a été remplacé par sa version "franchissable" (Figure 3.4).

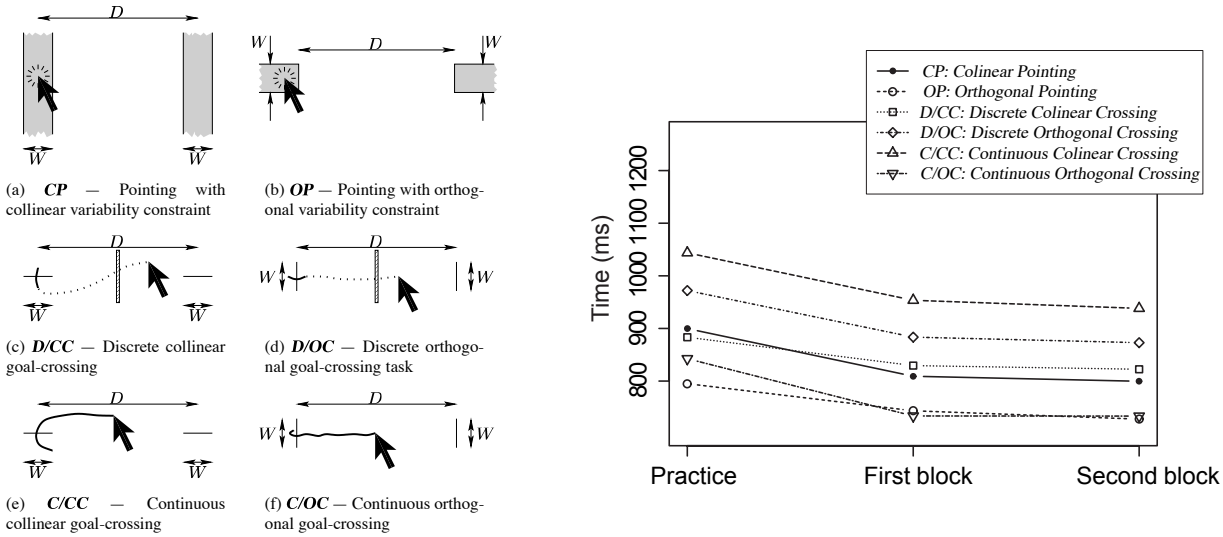


FIG. 3.3 : Pointage (P) VS. Franchissement (C) en fonction du type d'interaction (D : Discrète / C : Continue) et de la direction (O : Orthogonale / C : Colinéaire) (*Illustration extraite de [4]*)

Ces différentes lois ont été utilisées de façon concluante en interaction-homme machine et présentent toutes au moins une expression simple facilement utilisable. Les informations nécessaires à l'utilisation de ces lois sont présentes dans la modélisation CIS de l'état de l'interface et leur niveau d'abstraction correspond à celui des briques de base des graphes d'interaction CIS. Le paragraphe suivant décrit comment

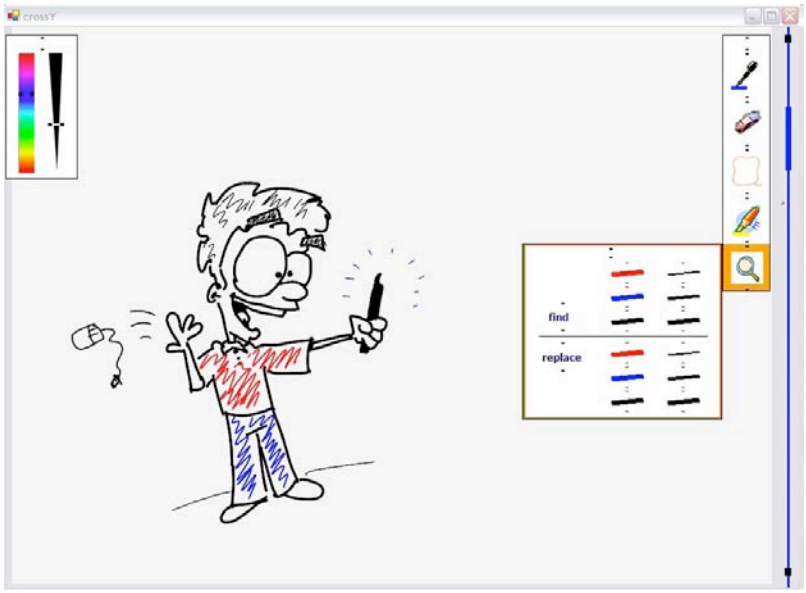


FIG. 3.4 : CrossY : Un éditeur graphique basé sur le franchissement (*Illustration extraite de [10]*)

le simulateur SimCIS tire parti de ces lois pour prédire le temps nécessaire à une séquence d'interaction.

3.1.2 Complexité d'une technique d'interaction

Nous avons déjà mis l'accent sur l'importance de lier la performance d'une technique d'interaction à son contexte d'utilisation, c'est pourquoi nous nous sommes inspirés de la notion de complexité utilisée en algorithmique dans laquelle le temps et l'espace pris par un algorithme donné n'est jamais décorrélé de la taille et de la nature des données en entrée. Ainsi, on parle de complexité dans le pire cas, dans le meilleur cas ou en moyenne correspondant à des situations où les données en entrée sont plus ou moins "favorables" à l'algorithme évalué. Ici, nous introduisons la *complexité en temps* et la *complexité en nombre d'actions* d'une technique d'interaction pour une séquence d'interaction. Une séquence d'interaction est une séquence de commandes qui change l'état de l'interface. Nous définissons un *problème* comme un état de l'interface à atteindre selon une séquence d'interaction donnée. La taille du problème correspond à la longueur de la séquence (c'est-à-dire le nombre de commandes). Les actions sont les acquisitions et validations utilisées dans la séquence d'interaction qui résout le problème, c'est-à-dire les actions qui permettent d'activer les bonnes commandes dans l'ordre de la séquence. La complexité d'une technique d'interaction pour la séquence donnée mesure le coût de ces actions lorsque l'on utilise cette technique d'interaction. Nous utilisons deux mesures différentes : le *nombre d'actions* nécessaire à la résolution du problème et le *temps* d'exécution de ces actions. La figure 2.19 (répétée à la figure 3.5 pour faciliter la lecture) illustre comment deux séquences d'interaction peuvent permettre d'atteindre le même état de l'interface. Comme avec les algorithmes, on peut donc envisager d'étudier les meilleurs et pires cas, c'est-à-dire les séquences les plus et les moins coûteuses qui résolvent le même problème.

Contexte de recopie Seq_{copy}	Contexte de résolution de problème $Seq_{problemsolving}$
1. $(Create_Triangle(100, 100))$	1. $(Create_Triangle(100, 100))$
2. $(Create_Triangle(700, 300))$	2. $(Create_Ellipse(300, 100))$
3. $(Create_Triangle(100, 700))$	3. $(Create_Rectangle(700, 100))$
4. $(Create_Ellipse(300, 100))$	4. $(Create_Triangle(700, 300))$
5. $(Create_Ellipse(300, 300))$	5. $(Create_Ellipse(300, 300))$
6. $(Create_Ellipse(300, 700))$	6. $(Create_Rectangle(100, 300))$
7. $(Create_Rectangle(700, 100))$	7. $(Create_Triangle(100, 700))$
8. $(Create_Rectangle(100, 300))$	8. $(Create_Ellipse(300, 700))$
9. $(Create_Rectangle(700, 700))$	9. $(Create_Rectangle(700, 700))$

FIG. 3.5 : Deux séquences d'interaction. À gauche : recopie ; à droite : résolution de problème.

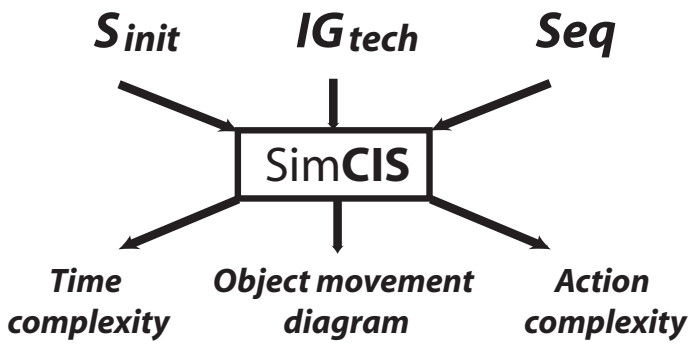


FIG. 3.6 : SimCIS : entrées et sorties

Nous avons développé une application, SimCIS, qui simule l'utilisation d'une technique d'interaction et calcule sa complexité. SimCIS prend en entrée :

- l'état initial de l'interface, S_{init} ;
- le graphe d'interaction de la technique IG_{tech} ;
- la séquence d'interaction, Seq.

A partir de ces entrées, SimCIS construit le *graphe de séquence* qui décrit l'enchaînement des utilisations d'une technique en fusionnant ensemble la ou les racines d'un graphe aux feuilles du graphe précédent. Chaque chemin partant de la racine à une feuille du graphe de séquence instancie une séquence d'interaction. SimCIS construit dans ce graphe le chemin P permettant d'instancier la séquence d'interaction Seq et évalue les complexités en temps et en action. La figure 3.7 montre, par exemple, le chemin permettant

d'instancier la séquence $(Create_Triangle(100, 100))$ - $(Create_Ellipse(300, 100))$ et qui contient huit actions (quatre nœuds et quatre arcs).

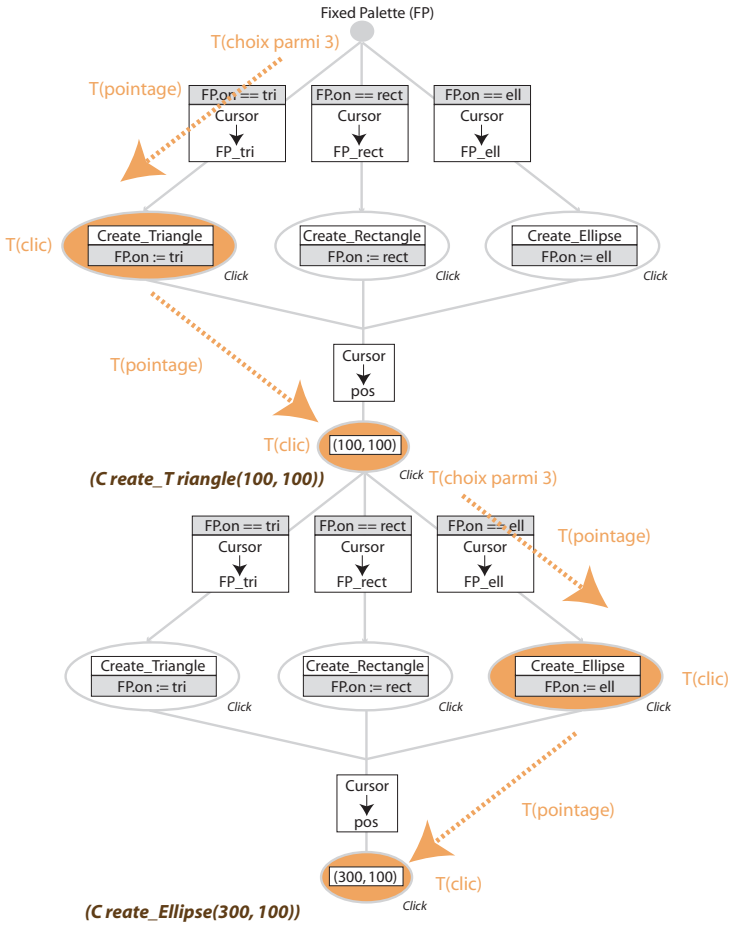


FIG. 3.7 : SimCIS : calcul des complexités

La complexité en actions correspond au nombre de noeuds et d'arcs (acquisitions et validations) qui constituent le chemin P. Quand une précondition est vérifiée, l'arc et le noeud correspondants ne sont pas comptés. La complexité en temps est obtenue en sommant le temps de chaque noeud et arc (comme indiqué par la figuer 3.7). Le temps d'un noeud est donné par son étiquette (Un clic prend 200 ms en moyenne par exemple) tandis que le temps d'un arc correspond à la somme du temps pour choisir cet arc et du temps pour exécuter l'action associée à cet arc. Pour ce faire, nous utilisons les lois empiriques que nous avons introduit à la section précédente : le temps de choix est prédit en utilisant la loi de Hick-Hyman et le temps d'un arc est prédit en utilisant la loi de Fitts. Quand deux actions sont parallèles, nous estimons le temps total de ces deux actions par une somme pondérée des temps des deux actions $T = T_{max} + 1.5 \times T_{min}$. En effet, Bourgeois [36] a introduit l'indice de coordination qui évalue le degré

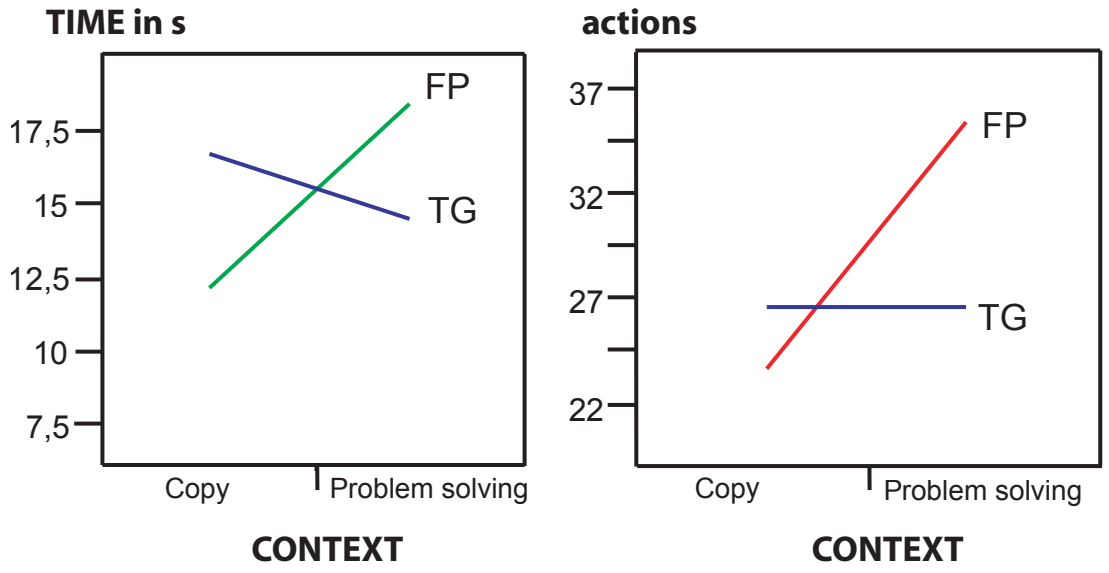


FIG. 3.8 : Complexités en temps (à gauche) et en actions (à droite) de la palette fixe et de la toolglass dans des contextes de recopie et de résolution de problème

de parallélisme entre les contrôles de deux dimensions avec les deux mains et a notamment montré que l'interaction bi-manuelle montre un fort parallélisme. Ici encore, quand une précondition est vérifiée, les temps de l'arc et du noeud correspondants ne sont pas comptés.

Outre les mesures de complexités, SimCIS fournit également en sortie un diagramme appelé Object Movement Diagram (OMD) illustrant les différents mouvements d'objets qui ont eu lieu pendant la séquence. Ces diagrammes sont utiles pour aider à comprendre pourquoi une technique est plus performante qu'une autre en fonction de son contexte d'utilisation. L'axe vertical représente le temps (chronologique du haut vers le bas) et l'axe horizontal représente une approximation des distances entre objets. Les objets et les positions d'intérêt, c'est-à-dire ceux qui interviennent dans la séquence, sont représentés par des lignes brisées : un sous-segment oblique correspond à un mouvement de cet objet. Les objets statiques sont représentés par une double ligne verticale (voir par exemple les trois outils de la palette sur le diagramme en haut de la figure 3.9). Lorsque que deux objets sont déplacés ensemble, les outils d'une toolglass ou lorsque un objet est déplacé par drag'n drop avec le curseur, ces objets sont liés par une simple barre horizontale.

Nous avons utilisé SimCIS pour comparer différentes techniques sur différents contextes et notamment les deux techniques, Palette Fixe (FP) et ToolGlass (TG), sur les deux séquences opérationnalisant les contextes de recopie et de résolution de problèmes, Seq_{copy} et $Seq_{problemsolving}$ (Figure 3.5). La figure 3.8 reporte les complexités en temps et en nombre d'actions des quatre combinaisons technique $\times$ contexte et la figure 3.9 montre les OMDs correspondants. Ces prédictions montrent que la palette fixe est très sensible au contexte à la fois en terme de temps et en terme d'actions, les OMDs mettant

	Contexte de recopie (Seq_{copy})	Contexte de résolution de problème ($Seq_{problemsolving}$)
Palette Fixe (FP)		
ToolGlass (TG)		

FIG. 3.9 : Exemples d'Object Movement Diagrams (OMDs)

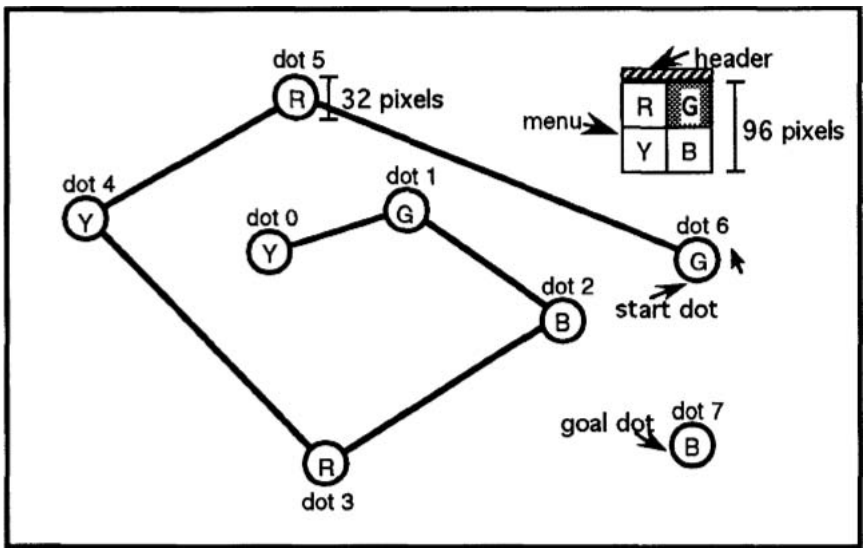


FIG. 3.10 : La tâche "connect the dots". (*Illustration extraite de [101]*)

en évidence le coût ajouté des différents aller-retours vers la palette dans un contexte de résolution de problèmes qui maximise le nombre de changements d'outil (comparer les deux OMDs du haut dans la figure 3.9). La toolglass est, elle, beaucoup moins sensible au contexte. Ce n'est pas surprenant étant donné que la toolglass n'est pas persistante et exige de sélectionner l'outil à chaque fois que l'on veut s'en servir. Enfin, le résultat, très intéressant, est le croisement entre les courbes des toolglass et palette fixe qui traduit que la "meilleure technique" dépend bien du contexte : la toolglass est plus performante que la palette fixe dans un contexte de résolution de problèmes alors que le résultat s'inverse dans un contexte de recopie.

Ces résultats sont en accord avec ceux rapportés lors de l'étude expérimentale de Mackay [116] qui nous a fortement inspirés pour cet exemple. Nous avons construit les séquences étudiées afin d'opérationnaliser les contextes qui ont été testés dans l'expérience de Mackay : recopie et résolution de problèmes. Rappelons que les résultats quantitatifs et qualitatifs ont rapporté que les utilisateurs sont plus efficaces avec les toolglasses qu'avec les palettes dans un contexte de résolution de problème alors qu'ils sont plus efficaces avec les palettes qu'avec les toolglasses dans un contexte de recopie. Nos prédictions sont donc consistantes avec ces résultats. Une autre étude expérimentale menée par Kabbash et al. [101] compare les toolglasses aux palettes et rapporte que les toolglass sont plus efficaces que les palettes. Ce résultat n'est pas surprenant si on regarde de plus près la tâche qui a été utilisée pour comparer ces techniques. Dans la tâche de Kabbash et al., "connect the dots" (figure 3.10), des points colorés apparaissent l'un après l'autre et l'utilisateur doit sélectionner la couleur du nouveau point avant de le relier par drag'n drop au point précédent. Dans l'étude de Kabbash et al., deux points successifs ont toujours deux couleurs différentes, ce qui est proche de l'opérationnalisation d'un contexte de résolution de problèmes que nous avons proposée puisque l'utilisateur doit changer d'outil à chaque fois. Les prédictions que nous obtenons nous amènent à penser que cette tâche est avantageuse pour les toolglasses précisément pour

cette raison. La cohérence observée entre les prédictions de CIS et les résultats déjà rapportés dans la littérature constitue une validation préliminaire du modèle. Une validation plus formelle de CIS et SimCIS est présentée au chapitre suivant.

CIS et SimCIS sont des outils destinés à faciliter l'analyse et la comparaison des techniques d'interaction. Ils ne sont pas destinés à se passer d'expérimentations contrôlées mais à améliorer la qualité des expérimentations contrôlées et à réduire le temps nécessaire à leur conception. En effet, en testant différentes séquences d'interaction, l'expérimentateur peut sélectionner une séquence représentative d'un contexte d'utilisation donné pour la tâche expérimentale et rapporter des résultats valides dans ce contexte ou, au contraire, inclure différentes séquences dans son expérimentation pour rapporter des résultats ayant une plus large validité. SimCIS permet également de se faire une idée a priori des résultats de l'expérimentation et donc de réduire le risque de réaliser une expérimentation sans résultats exploitables.

Les sections suivantes illustrent que CIS et SimCIS sont également des outils pour comprendre comment réaliser une technique efficace. D'une part, ils permettent d'identifier les actions à optimiser à partir d'un graphe existant. D'autre part, ils aident à comprendre les propriétés qui avantagent ou désavantagent une technique dans un contexte d'utilisation et constituent donc une base de réflexion d'un niveau d'abstraction supérieur à l'implémentation et plus propice à l'innovation.

3.2 Explorer l'espace de conception : Optimiser au niveau de l'action

Cette section introduit deux exemples de techniques que nous avons réalisées pour optimiser un arc d'un graphe d'interaction CIS et/ou un noeud : OrthoZoom et ControlTree. OrthoZoom [15] est une nouvelle technique de pointage multi-échelles efficace alors que ControlTree [14] combine pointage et franchissement pour naviguer dans de grandes hiérarchies.

3.2.1 Agir au niveau de l'action grâce aux lois empiriques

Les lois empiriques sur lesquelles repose les actions des graphes d'interaction CIS sont une source d'inspiration importante en IHM. L'état de l'art sur la loi de Fitts révèle qu'elle a servi non seulement de source d'inspiration pour créer de nouvelles techniques d'interaction plus efficace mais aussi de cadre expérimental pour de nombreux travaux. Le protocole expérimental basé sur la loi de Fitts est devenu un standard d'évaluation [118], créant ainsi un pan de la littérature plus facile à unifier. En effet, la tâche de pointage utilisée dans ces expérimentations ne fait intervenir que des actions motrices et est donc décorrélée d'un domaine d'application particulier. De plus, pour ces expérimentations, les performances des techniques sont toujours rapportées de la même manière, c'est-à-dire le temps d'exécution en fonction du facteur *indice de difficulté*.

La loi de Fitts, sous-jacente à une famille d'arcs CIS, s'exprime en fonction de l'indice de difficulté, c'est-à-dire du rapport de deux variables : la distance de la position courante à la cible (D) et la largeur de la cible (W).Cette formulation donne des indications pour améliorer le temps de pointage : réduire les distances ou augmenter la taille des cibles. La littérature montre que ces deux guides de conception

ont inspirés nombre de chercheurs en IHM. Ainsi, Baudisch et al. agissent sur D dans leur *drag-and-pop* [23] qui amène temporairement les cibles potentielles autour du curseur (partie droite de la figure 3.11). Collomb et al. permettent de "lancer à l'arc" un objet vers une cible avec leur *drag-and-throw* [55] (partie gauche de la figure 3.11). *Object pointing* [81] est le travail le plus "extrême" dans cette voie puisqu'il s'agit de prédire quelle est la cible dès les tous premiers instants du mouvement et de déplacer automatiquement le curseur sur cette cible ($D \simeq 0$).

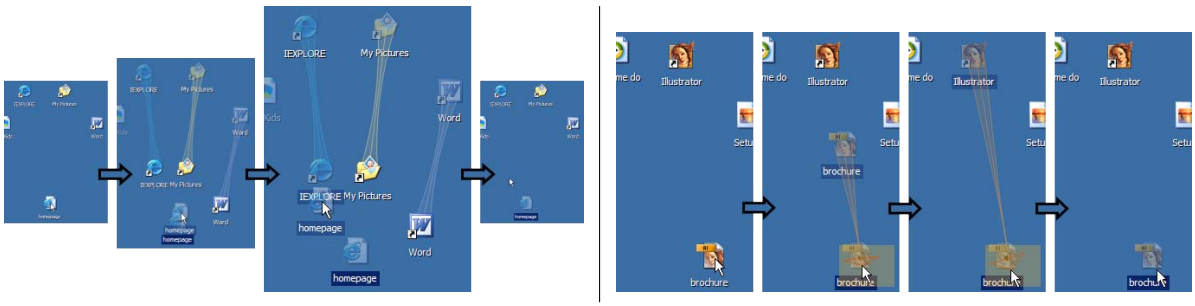


FIG. 3.11 : Diminuer D : à gauche, le drag-and-pop ; à droite, le drag-and-throw (Illustrations extraites de http ://www.lirmm.fr/edel/dragging/)

Le but d'"augmenter W" n'a pas été moins prolifique. Ainsi, McGuffin et Balakrishnan [123] ont montré qu'augmenter la taille de la cible même tardivement dans une tâche de pointage réduit significativement le temps de mouvement. Cette observation reste encore vraie dans le cas où l'expansion de la cible n'est pas prévisible par l'utilisateur [173]. Le Bubble Cursor [74] est un curseur dont la taille de la zone d'activation change au cours du mouvement pour englober à chaque instant la cible la plus proche (changer la taille de l'objet déplacé ou celle de la cible ayant le même effet sur le temps de mouvement). Enfin, le pointage sémantique [33] agit à la fois sur D et W en distordant les cibles uniquement dans l'espace moteur : les pointages sont alors plus faciles sans qu'il n'y ait le moindre changement visuel pour l'utilisateur. Il ne s'agit ici que de quelques exemples, l'article [18] de Balakrishnan fournit un état de l'art plus exhaustif des travaux dont le but est de "battre la loi de Fitts".

Plus récemment, les études se sont concentrées autour de la loi de Fitts pour des pointages dans des interfaces mutli-échelles. Le grand défi pour ces interfaces est que le contrôle du facteur de zoom et du déplacement est fait par l'utilisateur au fur et à mesure de son déplacement. Les solutions proposées sont donc motivées par la volonté d'introduire le contrôle le plus efficace permettant d'ajuster le facteur d'échelle qui agit conjointement sur D et W. La loi de Fitts s'est avérée rester un cadre expérimental approprié à ces pointages [82], ce qui est assez remarquable car les types de mouvement mis en œuvre dans une interface zoomable sont très différents d'un pointage "classique". C'est dans ce cadre que nous avons imaginé OrthoZoom [15], une nouvelle technique de pointage que nous présentons et évaluons à la section suivante.

Les lois empiriques ont donc non seulement l'avantage de servir de guides de conception pour l'interaction mais aussi de cadre expérimental bien défini pour évaluer les différentes alternatives. Ainsi,

toutes les techniques qui viennent d'être mentionnées ont été évaluées sur des tâches de pointage ayant différents indices de difficulté et les résultats de ces évaluations sont fournies d'une manière unifiée : le temps de mouvement en fonction de l'indice de difficulté. La loi de Fitts est même devenue un standard pour l'évaluation des dispositifs de pointage [57]. La loi de steering, plus jeune, commence à suivre le même chemin. Ainsi, Accot et al. [3] ont ouvert la voie avec une première évaluation de cinq dispositifs d'entrée pour accomplir des tâches de suivi de tunnel.

Décrire des techniques d'interaction avec des graphes d'interaction CIS découpe l'interaction en étapes et rend visible l'ensemble des lois empiriques qui gouvernent la technique et donc les variables sur lesquelles agir pour optimiser l'interaction. La littérature contient nombre d'expérimentations contrôlées rapportant les performances des différentes alternatives qui permettent de réaliser ces actions. Le concepteur d'interfaces peut donc plus facilement lire les résultats de cette littérature ciblée pour faire des choix qui optimisent les actions de sa technique ou, comme présenté ci-dessus, se servir des variables de ces lois pour imaginer de nouvelles interactions en agissant sur les valeurs de ces variables. C'est ainsi que la technique de pointage OrthoZoom, décrite à la section suivante, a été imaginée.

3.2.2 Optimiser un arc : OrthoZoom

Les tâches de navigation et de sélection en une dimension (1D) (avec un glisseur (*slider*) ou une barre de défilement (*scrollbar*) par exemple) consistent à sélectionner par pointage une valeur dans un intervalle borné. Cette tâche de pointage devient un pointage multi-échelle dès lors que l'intervalle à représenter est trop grand relativement à la taille ou la résolution du dispositif d'affichage. Par exemple, représenter un intervalle de $[1, 10000]$ sur seulement 1000 pixels ne permet pas de sélectionner chacune des valeurs si l'on n'envisage pas une représentation multi-échelle dans laquelle l'utilisateur peut naviguer. De même, il est impossible de faire défiler continûment un très grand document avec une scrollbar traditionnelle, un déplacement d'un pixel pouvant provoquer un saut de plusieurs pages.

OrthoZoom [15] est une technique de pointage multi-échelle 1D qui utilise un périphérique de pointage 2D standard (souris, stylet, trackpad, etc.). Elle permet de contrôler le facteur de zoom selon la dimension orthogonale (l'abscisse de la souris par exemple) à celle utilisée pour la navigation (l'ordonnée de la souris par exemple) (Figure 3.12). Après une revue des travaux antérieurs, nous présentons le principe d'OrthoZoom et une évaluation comparant OrthoZoom (OZ) et Speed Dependant Automatic Zooming (SDAZ) [95] pour des tâches de pointage dont l'indice de difficulté varie de 10 à 30. L'évaluation révèle qu'OrthoZoom permet de réaliser des pointages très difficiles deux fois plus rapidement que la seule technique de pointage multi-échelle n'utilisant qu'une souris. Nous terminons par un exemple d'application pour naviguer dans de grands documents textuels qui tire avantage d'OrthoZoom.

Travaux antérieurs

Il existe deux grandes familles de techniques pour sélectionner une valeur précise dans un intervalle : les techniques discrètes et les techniques continues.

Les techniques discrètes Les techniques discrètes utilisent des mécanismes non continus comme le filtrage pour retirer progressivement des valeurs et réduire la taille de l'intervalle. Par exemple, BinScroll

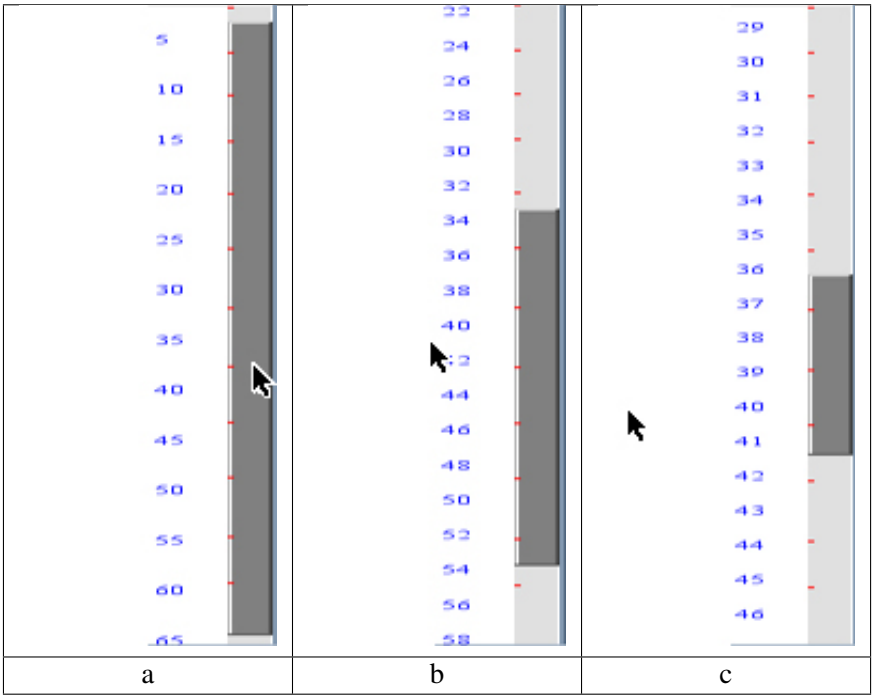


FIG. 3.12 : Principe d'OrthoZoom : (a) précision faible (b) précision moyenne (c) précision haute

[110] est une technique qui utilise quatre boutons pour faire une recherche dichotomique se limitant ainsi aux listes ordonnées (deux boutons permettent de sélectionner le haut ou le bas de cette liste et deux autres boutons permettent de valider ou annuler). LensBar [121] est également une *listbox* augmentée d'un *slider* et d'une entrée textuelle pour faire une sélection dans une grande liste de données. Cliquer sur l'élément courant et se déplacer vers la gauche permet d'afficher la liste avec une granularité moins fine. LensBar contrôle la visibilité des différents éléments selon leur degré d'intérêt (DOI). LensBar est une technique pour des interfaces équipées d'un clavier et requiert un pré-traitement pour assigner un DOI à chaque élément.

L'Alphaslider [6] est un slider constitué de trois sous-sliders, chacun agissant à une granularité différente. Bien que chaque sous-slider soit manipulé par une interaction continue, le passage d'un sous-slider à l'autre brise cette continuité et va finalement à l'encontre des principes de la manipulation directe de Shneiderman [159]. De plus, le fait que l'utilisateur ne peut agir qu'à trois niveaux de granularité différents fait de l'AlphaSlider une technique qui ne passe pas à l'échelle pour de très grands intervalles. Le FineSlider [122] étend l'idée de l'AlphaSlider : cliquer en dehors de la poignée du slider permet d'ajuster la valeur à une granularité proportionnelle à la distance entre la position de clic et le slider. Une fois de plus, le passage d'une granularité à l'autre reste discontinu puisque la granularité est dépendante du point de départ de l'interaction.

Les techniques continues Comme nous l'avons déjà mentionné précédemment, Guiard et al. [77] ont démontré que les tâches de sélection précise peuvent être vues comme des tâches de pointage multi-échelle. Les techniques d'interaction continues utilisent effectivement cette dimension d'échelle pour permettre d'agir à la fois sur la position (ou pan) et sur l'échelle (ou zoom). Cette famille de techniques peut être découpée en deux sous-familles : les techniques utilisant une entrée non standard et les techniques utilisant une entrée standard.

Parmi les techniques utilisant une entrée non standard, les techniques de pointage bimanuel ont fait l'objet de nombreux travaux [174, 77, 80]. Par exemple, dans [80, 82, 77], les utilisateurs de déplacent en utilisant un stylet dans la main dominante et changent d'échelle en utilisant un joystick dans la main non-dominante. Ces techniques sont très efficaces mais sont bien souvent impossibles à mettre en œuvre dans les configurations d'entrée habituelles. De même, Zliding [151] requiert un capteur de pression pour ajuster le niveau de zoom.

Deux techniques récentes utilisent un contrôle circulaire pour ajuster le niveau de zoom : le Radial Scroll Tool [162] et le Virtual Scroll Ring [124]. Elles utilisent la rotation dans les deux sens pour se déplacer. Le Radial Scroll Tool utilise le rayon du cercle pour déterminer la vitesse de défilement alors que le Virtual Scroll Ring utilise la distance parcourue sur la circonférence du cercle. Cependant, les mouvements circulaires peuvent être inadaptés pour certains périphériques d'entrée comme la souris.

Le Position+Velocity Slider est une technique pour les interfaces à stylet proposée dans le système LEAN [150], un prototype pour gérer des flux vidéo. Comme les sliders multi-résolution de la boîte à outils Infovis [62], cette technique ajuste la position par une interaction de drag durant laquelle le niveau de zoom est fonction de la distance entre le point courant et le point initial. Ces deux techniques prometteuses n'ont pas fait l'objet d'une évaluation formelle et il est donc difficile d'apprécier leur efficacité. Enfin, Speed Dependant Automatic Zooming (SDAZ) [95] ne laisse pas le contrôle direct du niveau de zoom à l'utilisateur puisque celui-ci est fonction de la vitesse de défilement. Une fois de plus, il s'agit d'une interaction de drag dans laquelle la distance entre le point courant et le point initial définit la vitesse de défilement, le facteur de zoom étant ajusté automatiquement pour garder un flux visuel continu. Cockburn et al. [54] ont montré que SDAZ est une technique qui nécessite une calibration fine mais qui s'avère être la plus efficace jusqu'à maintenant pour se déplacer dans de grands documents textuels.

OrthoZoom : le principe

OrthoZoom étend un slider 1D ou une barre de défilement en une technique de navigation multi-échelle. Il se comporte exactement comme ces widgets standard lorsque l'utilisateur interagit dans les bornes de la représentation du widget. Lorsque le curseur sort de ces bornes, le facteur de zoom est changé continûment (Figure 3.12). Le facteur de zoom décroît au fur et à mesure que le curseur s'éloigne des bornes perpendiculairement à la dimension de déplacement. En d'autres termes, déplacer la souris le long de la dimension de direction provoque effectivement un déplacement alors que déplacer la souris le long de la direction **Ortho**gonale ajuste le facteur de **zoom**. D'où le nom de cette technique : OrthoZoom.

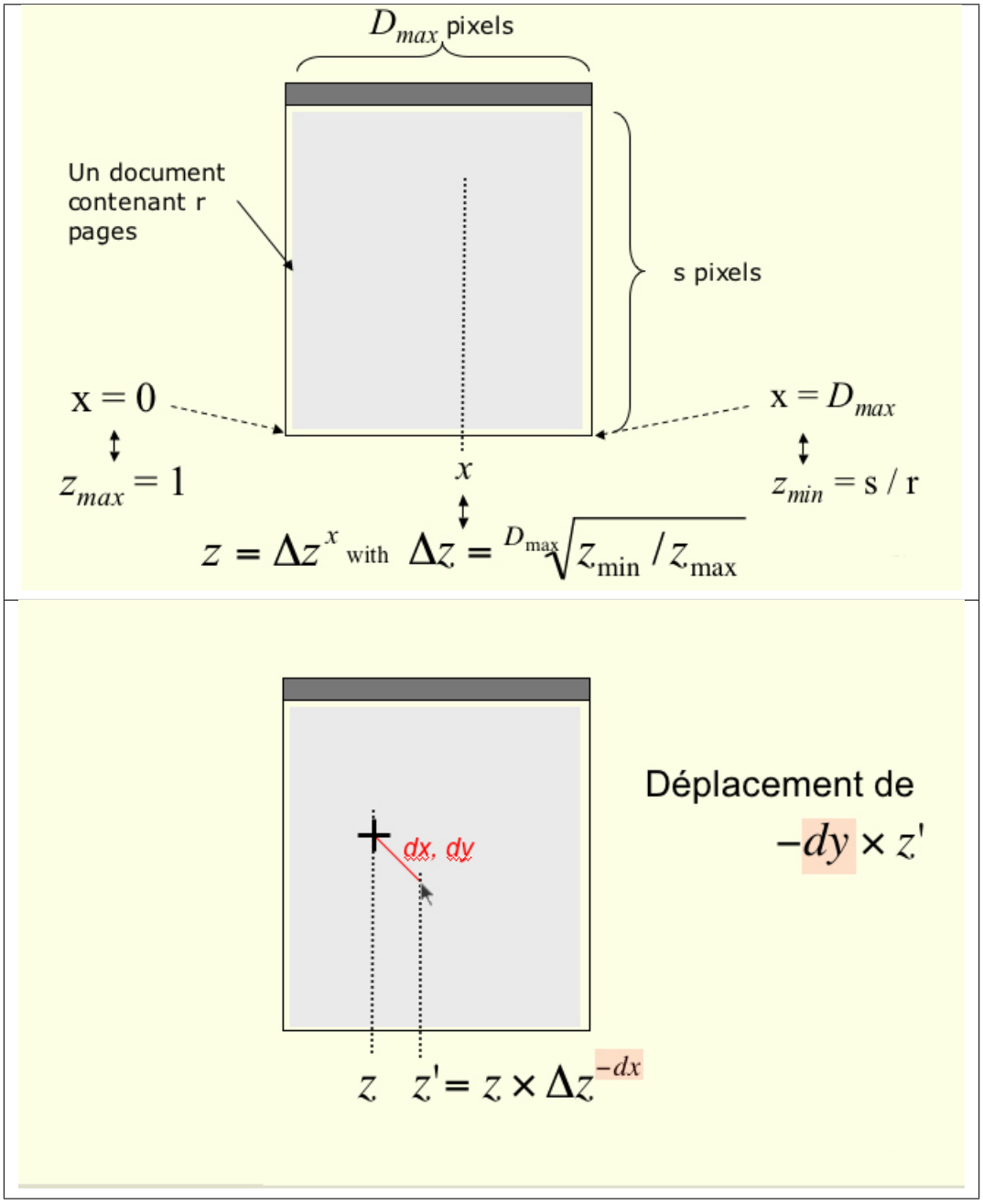


FIG. 3.13 : Utiliser la dimension orthogonale pour contrôler le facteur de zoom

Aire de contrôle : la dimension orthogonale Considérons la sélection d'une valeur dans un intervalle R contenant r valeurs représenté par un slider S de s pixels de long. Une valeur peut être sélectionnée avec une précision r/s, ce qui est équivalent à représenter le slider avec un facteur de zoom de

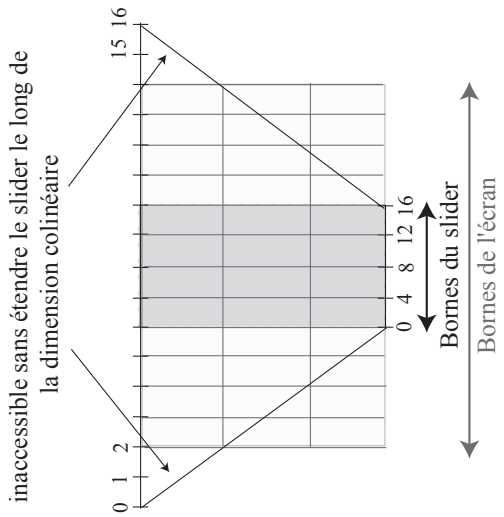


FIG. 3.14 : Valeurs qui peuvent être atteintes en restant dans les bornes du slider et dans les bornes de l'écran

$s/r = Z_{min}$. Donc, lorsque $r > s$, certaines valeurs deviennent inaccessibles avec un slider traditionnel. OrthoZoom ajuste la précision en utilisant la direction orthogonale à celle du slider. OrthoZoom a donc une aire de contrôle supérieure à son aire de représentation visuelle. Plus la distance orthogonale entre le slider et le curseur est grande, plus le facteur de zoom est grand (Figure 3.13). Ainsi, le facteur de zoom initial peut être choisi en démarrant l'interaction à une distance orthogonale de son choix (cette dernière remarque n'est valable que si la totalité de la fenêtre est disponible pour OrthoZoom, sinon l'interaction commence à la précision maximale dans les bornes du slider). L'implémentation d'OrthoZoom est simple, il suffit :

- d'assigner un facteur de zoom maximal, Z_{max} à la distance orthogonale minimale, (typiquement, $Z_{max} = 1$).
- d'assigner le facteur de zoom Z_{min} à la distance orthogonale maximale, D_{max}.
- interpoler l'intervalle $[Z_{min}, Z_{max}]$ sur l'intervalle $[0, D_{max}]$ afin d'assigner un facteur de zoom. Pour une position x comprise entre 0 et D_{max}, le facteur de zoom z est :

$$z = (\Delta_z)^x \text{ avec } \Delta_z = \sqrt[D_{max}]{Z_{min}/Z_{max}}$$

Ainsi, lors d'un déplacement (d_x, d_y), le nouveau facteur de zoom z' et le déplacement correspondant sont mis à jour selon les formules de la partie basse de la figure 3.13.

Aire de contrôle : la dimension colinéaire L'aire de contrôle d'OrthoZoom n'est pas limitée aux bornes de sa représentation graphique. Cependant, autoriser l'utilisateur à contrôler le facteur de zoom pose un problème lorsque $r/z > s$: l'utilisateur ne peut atteindre les extrémités de l'intervalle comme l'illustre la figure 3.14 si les déplacements du curseur restent contraints aux bornes du slider dans la direction colinéaire (c'est-à-dire celle de déplacement). Pour résoudre ce problème, l'aire de contrôle est donc non seulement élargie dans la direction orthogonale mais aussi dans la direction colinéaire. Lorsque le curseur atteint les bornes physiques du dispositif d'affichage, un défilement automatique à la précision

courante est enclenché (*rate-based scrolling*).

Évaluation

Cette section présente l'expérimentation contrôlée que nous avons menée pour comparer OrthoZoom (OZ) avec Speed Dependant Automatic Zooming (SDAZ) pour des tâches de pointage de différents indices de difficulté, SDAZ étant la seule technique continue et qui n'a besoin que d'un périphérique standard. Nous avons conçu cette expérience non seulement pour comparer ces deux techniques mais aussi pour mesurer les limites de ces techniques en les soumettant à des tâches de pointage très difficiles.

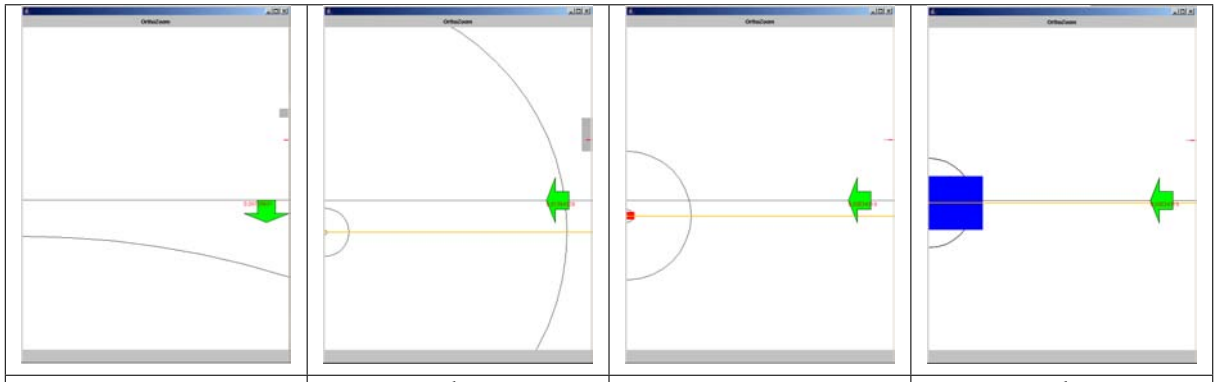
Sujets Douze participants volontaires, 11 hommes et 1 femme, ont accepté de participer à l'expérimentation. L'âge moyen était de 26 ans.

Cadre expérimental L'expérimentation était exécutée sur une station de travail HP équipée d'un processeur 2 GHz Pentium 4, d'un écran LCD de dimension 1280×1024 LCD et d'une souris optique. Le programme a été réalisé avec Java 1.4 en utilisant la boîte à outils Piccolo Toolkit [28]. La fenêtre de l'application était fixée aux dimensions 600×800 pixels et la longueur du document 2^{30} pixels.

Tâche Les participants doivent atteindre le plus vite possible des cibles apparaissant les unes après les autres dans un document trop grand pour être visualisé à sa taille naturelle sans défilement. Les participants doivent donc faire défiler la vue verticalement pour amener la cible au centre de la vue, celui-ci étant constamment indiqué par une ligne noire horizontale. Une flèche montre également à chaque instant dans quelle direction aller pour trouver la cible (figure 3.15). En effet, nous évaluons une tâche de pointage et, dans ce type de tâche, l'utilisateur sait où la cible se trouve. Afin d'éviter que l'utilisateur soit perdu, une ligne horizontale orange insensible au facteur de zoom est toujours affichée sur la cible afin que celle-ci reste localisable à n'importe quel facteur de zoom. De plus, la cible est entourée de cercles concentriques sensibles au facteur de zoom pour donner un feedback au contrôle du participant. La tâche de pointage est considérée terminée lorsque la cible est restée une seconde au centre de la vue à un facteur de zoom égal à 1. La cible, initialement rouge, devient bleue, le temps de mouvement est alors enregistré et les instructions pour une nouvelle tâche de pointage apparaissent.

Hypothèses

- H_1 : **OZ est plus efficace que SDAZ pour des tâches de pointage**
  Nous prédisons qu'OrthoZoom est plus efficace dans tous les cas. Nous pensons que les utilisateurs préfèrent contrôler le facteur de zoom eux-mêmes et que les changements de direction sont plus faciles avec OZ qu'avec SDAZ. Pour contrôler le facteur de zoom, SDAZ et OZ utilisent toutes les deux des déplacements de souris. SDAZ utilise la distance entre l'ordonnée courante de la souris y_t et l'ordonnée du point initial y_0 alors qu'OZ utilise uniquement la coordonnée de la souris x_t. Pour contrôler la direction de déplacement, SDAZ utilise le signe du vecteur d'interaction V alors qu'OZ utilise le vecteur de mouvement v (figure 3.16). Donc, avec SDAZ, l'utilisateur doit d'abord annuler son vecteur d'interaction (rejoindre l'ordonnée initiale y_0) avant de pouvoir se déplacer dans la direction opposée (le vecteur d'interaction étant non nul, le document continue de défiler dans la mauvaise direction). OZ, pour lequel il suffit de changer de direction directement



a	b	c	d

FIG. 3.15 : La tâche expérimentale : (a) la cible est sous la vue (b) la cible est dans la vue (ligne orange) mais pas visible au facteur de zoom courant (c) la cible est dans la vue et est visible au facteur de zoom courant (d) la cible a été pointée durant une seconde

avec la souris, devrait donc être plus efficace puisque les changements de direction peuvent être fréquents au cours d'un pointage [164].

	Facteur de zoom (z)	Direction de déplacement (d)
SDAZ	$z = f(y_t, y_0)$	$d =$signe$(y_t - y_0)$
OZ	$z = f(x_t)$	$d =$signe$(y_{t-1} - y_t)$

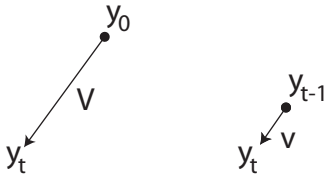


FIG. 3.16 : Facteur de zoom et direction de déplacement pour les deux techniques

– H_2 : **OZ est une technique efficace pour les pointages difficiles**
Les tâches de pointage dans de grands documents sont facilitées par un contrôle facile du facteur de zoom. Nous pensons que la dimension orthogonale permet de contrôler facilement le facteur de zoom et que, dès lors, OZ est une technique adaptée aux pointages difficiles. Les expérimentations ayant manipulé de très grands indices de difficulté ne sont jamais allées au-delà de 30 bits [80, 78, 82], nous avons donc soumis OZ et SDAZ aux mêmes conditions extrêmes.

Procédure Les facteurs de cette expérimentation sont :
- 2 $Techniques$ (OZ et SDAZ)
- 5 IDs (10, 15, 20, 25 et 30)
- 3 Ws (60, 120 et 300)
- = 30 conditions possibles.

Le facteur W, la taille de la cible, est prise en compte pour suivre les recommandations de Hinckley [89] sur une potentielle interaction entre la technique et la taille de la cible pour les techniques incluant des défilements de document.

Chaque participant exécute une série d'essais avec OZ et une série d'essais avec SDAZ. L'ordre de présentation est contrebalancé entre les participants afin d'éviter de perturber les participants avec des changements successifs de technique : un groupe commence avec OZ et l'autre commence avec SDAZ. Chaque bloc contient 45 essais (5 IDs x 3 Ws répétés 3 fois) avec une même technique et chaque participant doit exécuter deux blocs en série par technique. L'ordre de présentation des essais au sein d'un bloc est aléatoire.

Pour chaque essai, le temps de mouvement (MT), le nombre de fois où le participant relâche le bouton de la souris ($Releases$) et le nombre de fois où le participant passe sur la cible sans s'arrêter ($Overshoots$) sont enregistrés. À la fin de l'expérimentation, chaque participant doit donner son ordre de préférence entre les deux techniques et peut donner des commentaires.

Résultats L'analyse de variance révèle un effet significatif de la $Technique$ sur le temps de mouvement MT ($F_{1,11} = 393,094$ et $p < .0001$). Ce résultat illustré par la figure 3.17 supporte notre hypothèse H_1 : OZ est environ deux fois plus rapide que SDAZ.

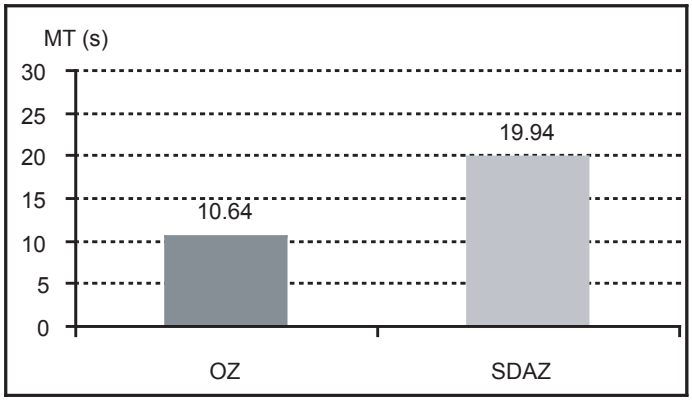


FIG. 3.17 : Analyse du temps de mouvement moyen

La figure 3.18 montre qu'OZ et SDAZ suivent bien la loi de Fitts et nos résultats révèlent un effet significatif de l'indice de difficulté sur le temps de mouvement ($F_{4,44} = 62.657$ et $p < .0001$). Cependant, l'effet de l'ID provient de la distance à la cible puisqu'il n'y a pas eu d'effet significatif de la largeur W sur le temps de mouvement ($F_{2,22} = 1.004$ et $p = 0.367$). Ceci est probablement dû à l'effet négligeable de W sur ID puisque la valeur de W était contrainte par la taille de la fenêtre. L'analyse de variance a également révélé un effet d'interaction $Technique \times ID$ ($F_{4,44} = 6.291$ et $p < .0001$). La figure 3.18 supporte l'hypothèse H_2 : la courbe de OZ présente une pente environ deux fois plus petite que la courbe de SDAZ, ce qui montre qu'OZ est une technique efficace pour les pointages très difficiles.

La figure 3.19 montre l'évolution du temps de mouvement en fonction du numéro d'essai (*trial*) pour chacune des deux techniques. Ici encore, la courbe de OZ a une pente plus petite que celle de SDAZ et

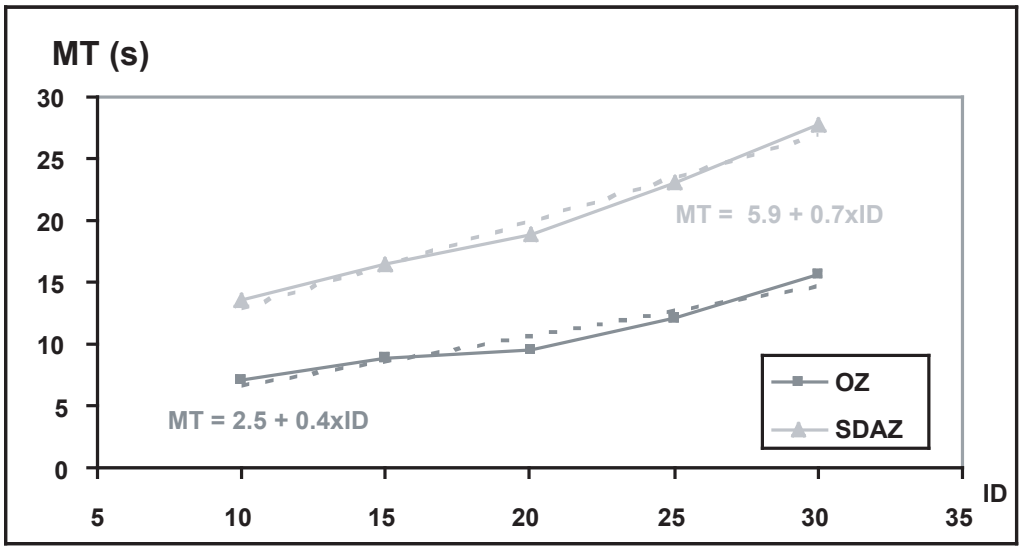


FIG. 3.18 : Analyse du temps de mouvement en fonction de l'indice de difficulté

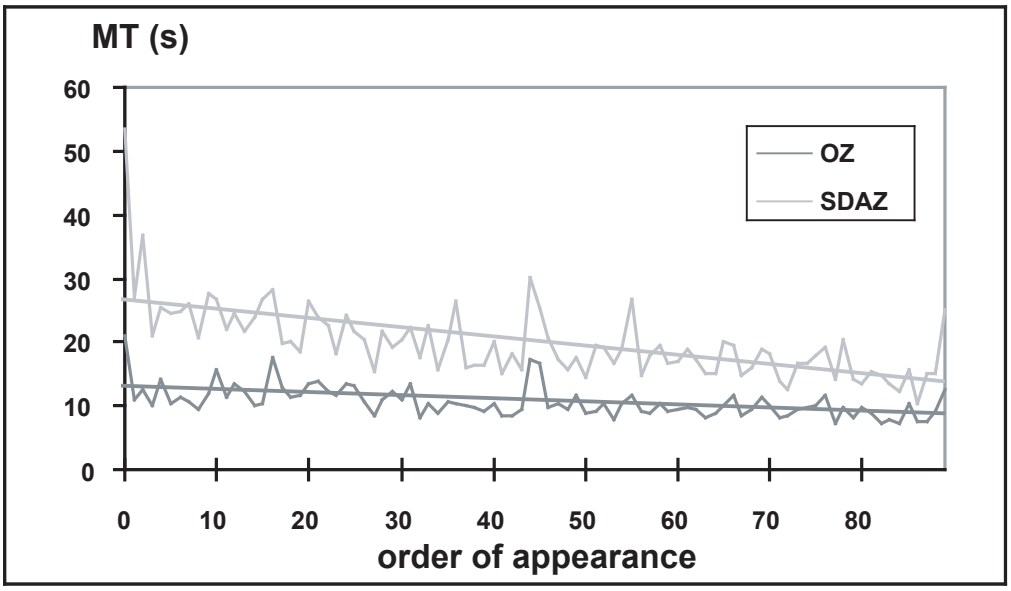


FIG. 3.19 : Apprentissage des deux techniques : temps moyen en fonction de l'ordre de présentation d'un essai

montre donc que les participants ont plus vite appris comment se servir d'OZ que de SDAZ.

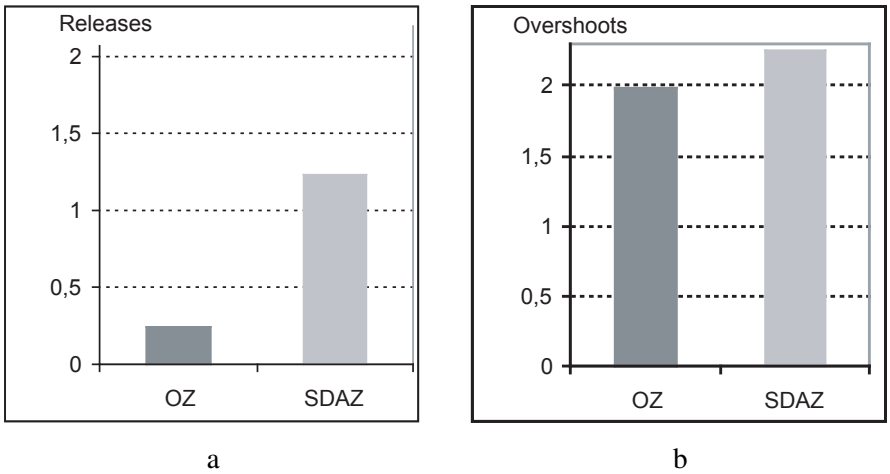


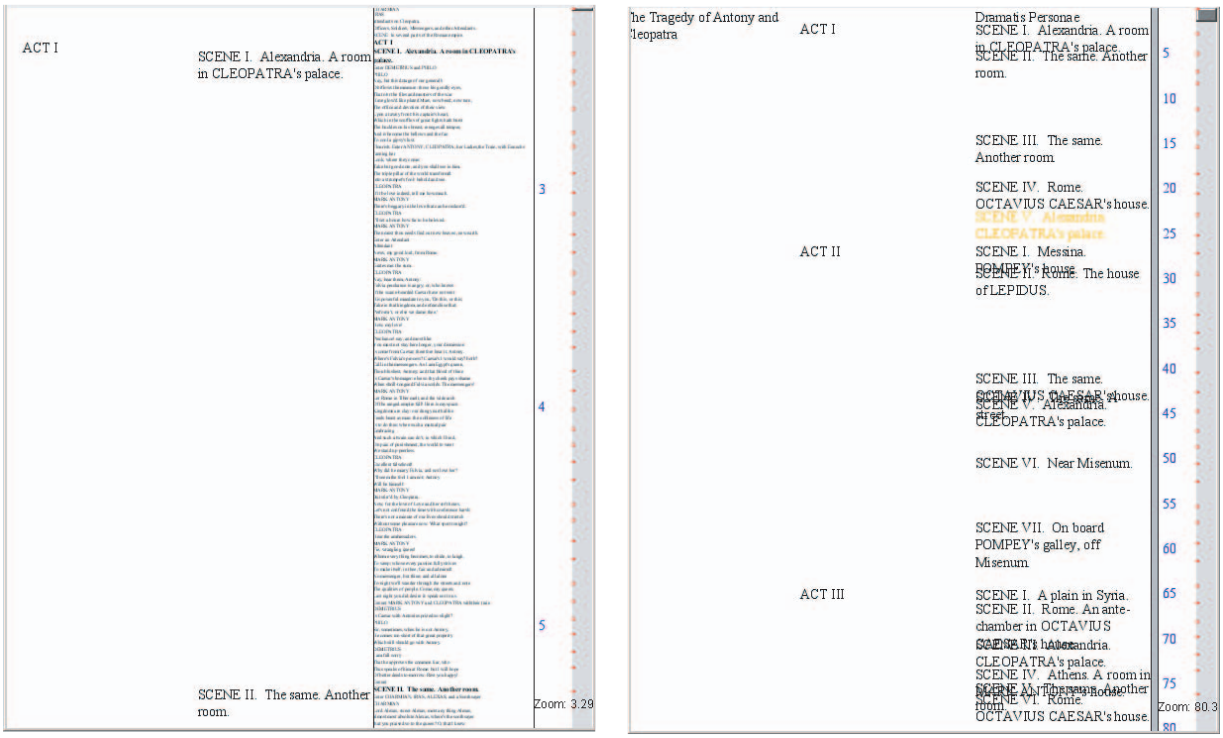
FIG. 3.20 : Analyse des erreurs : (a) nombre moyen d'erreurs de type *Release* (b) nombre moyen d'erreurs de type *Overshoot*

L'analyse de variance révèle un effet significatif de la $Technique$ sur le nombre de "Releases" ($F_{1,11} = 317.918$ et $p < .0001$), mais pas sur le nombre d'"Overshoots" (Figure 3.20). Les résultats de la figure 3.20 illustrent la stratégie utilisée par les participants durant l'expérimentation : pour changer de direction avec la technique SDAZ, ils relâchent le bouton de la souris et recommencent une interaction plutôt que de s'exposer au problème du changement de direction que nous avons expliqué précédemment. Enfin, l'ID n'a pas d'effet significatif sur ces deux mesures.

MSTOC : naviguer dans de grands documents textuels

Utiliser OrthoZoom pour naviguer dans de grands documents nécessite des adaptations du feedback puisqu'un pixel de déplacement peut provoquer un très grand déplacement dans le document. Considérons, par exemple, un document contenant les 37 pièces de Shakespeare, ce qui correspond environ à 150000 lignes de texte. En supposant qu'une fenêtre peut contenir 40 lignes de texte en moyenne à une taille raisonnablement lisible et une scrollbar de 1000 pixels de long, un déplacement d'un pixel dans la scrollbar peut provoquer un déplacement de 4 pages du document. Par exemple, les techniques multi-échelle pour faire défiler un document ajustent souvent le facteur de zoom en fonction de la vitesse de défilement pour maintenir un flot optique continu. Ajuster le facteur de zoom fonctionne bien pour des documents qui sont eux-mêmes multi-échelle comme une carte géographique, par exemple, qui garde du sens à différents niveaux d'échelle. Cependant, passer à l'échelle inférieure la totalité d'un document textuel ne révèle qu'une bande noirâtre illisible. Par exemple, naviguer à l'échelle des pièces dans les œuvres de Shakespeare nécessite de passer à l'échelle géométrique 1/150, échelle à laquelle le texte est illisible.

Pour résoudre ce problème, nous avons développé la Multi-Scale Table Of Contents (MSTOC) qui est affichée à gauche du texte qui a été réduit. La figure 11 illustre le comportement de la MSTOC.



a b

FIG. 3.21 : Multi-Scale Table Of Contents (MSTOC) pour les œuvres complètes de Shakespeare : la table du contenu apparaît sur la gauche du texte lorsque celui-ci est affiché à un facteur de zoom inférieur. Sur la droite, seule la structure reste visible.

Chaque entrée dans la table de contenu est affichée à une taille constante et est alignée verticalement avec sa position dans le document quel que soit le facteur de zoom pendant la navigation. Lorsque le facteur de zoom diminue, la MSTOC se réduit verticalement pour laisser place aux entrées de plus haut niveau. Lorsque le facteur de zoom diminue, des entrées peuvent éventuellement entrer en collision, comme sur la figure 3.21-b. Lorsque ceci se produit, la totalité du niveau est réduite et, lorsqu'il devient illisible, il est enlevé de la fenêtre, laissant ainsi de la place au niveau d'échelle supérieur qui commence à apparaître à gauche. De plus, nous avons implémenté un mécanisme de magnétisme ("snapping") dans la MSTOC : lorsqu'OrthoZoom est actif, l'entrée de plus haut niveau qui est alignée avec le curseur est mise en surbrillance. Si l'utilisateur relâche le bouton de la souris à ce moment-là, il arrivera exactement à cet endroit lorsque l'animation qui ramène au facteur de zoom 1 sera terminée. Ce comportement est très important étant donné qu'avec des grands documents, un pixel d'écart a de lourdes conséquences. La MSTOC affiche également les numéros de page (ce sont les entrées par défaut toujours disponibles). Le nombre de numéros de page dépend du niveau de zoom : toutes les pages, toutes les 2 pages, 5 pages, etc. Les numéros de page répondent également au "snapping" mais avec une priorité inférieure à celle des autres entrées. Enfin, cliquer sur la scrollbar déclenche une animation rapide montrant une vue générale de la MSTOC.

Ces détails d'implémentation font de la MSTOC une application dans laquelle une seule technique d'interaction permet des fonctions de navigation similaires à celle des logiciels qui permettent de se déplacer dans de grands documents en une dimension. Par exemple, Adobe Acrobat Reader fournit cinq moyens de navigation qui pourrait être remplacés par OrthoZoom : l'outil "main", les boutons suivant/précédent, la navigation par sélection de miniatures, la navigation par les "favoris" et l'entrée d'un numéro de page dans une zone textuelle. Selon l'abscisse du curseur, OrthoZoom permet de faire défiler le document au niveau de l'outil "main", au niveau de la miniature, des favoris, de la structure ou des numéros de page, chacun de ces éléments pouvant être une entrée dans la MSTOC. Toutes les interactions se trouvent alors intégrées à un endroit : la partie visible du document, qui est l'objet d'intérêt de l'utilisateur, respectant ainsi les principes de la manipulation directe [159]. Le système Metisse [52] intègre OrthoZoom dans le système de fenêtrage d'une façon globale afin que chaque barre de défilement traditionnelle peut être transformée en une barre de défilement OrthoZoom.

3.2.3 Optimiser un nœud : ControlTree

L'objectif de cette section est de concevoir une technique efficace et originale pour sélectionner un nœud dans une grande structure hiérarchique représentée sous forme d'un arbre "nœud-lien", qui est la représentation la plus familière pour les hiérarchies [142]. Alors qu'OrthoZoom permet d'optimiser des arcs CIS, ControlTree [14], présenté ici, a été obtenu en optimisant non seulement les pointages (les arcs CIS) mais aussi les sélections (les nœuds CIS) dans un affichage dynamique.

ControlTree : les motivations

Naviguer pour sélectionner un nœud dans une hiérarchie est une tâche fréquente. Tout utilisateur est régulièrement amené à sélectionner un fichier dans son système de fichiers ou à sélectionner une commande dans un menu. De même, beaucoup de spécialistes manipulent des données organisées hiérarchiquement. Par exemple, un biologiste cherche régulièrement une espèce dans des taxonomies ou un bibliothécaire un livre dans une classification.

Les travaux antérieurs Fournir une vue d'ensemble d'une hiérarchie sous forme d'arbres nœud-lien est très coûteux en surface d'affichage et ne passe pas à l'échelle, la limite maximale étant atteinte lorsque le nombre de nœuds excède le nombre de pixels (sans compter que la représentation devient illisible bien avant cette limite physique). Dès lors, la majorité des solutions reposent sur une technique de navigation dans une interface dont l'affichage est dynamiquement ajusté en fonction de la navigation.

Les techniques de visualisation dynamique d'une grande hiérarchie comme Windows Explorer, Hyperbolic Tree Browser [106] ou SpaceTree [143] proposent des visualisations nœud-lien intéressantes mais reposent toutes sur une interaction classique "point-and-click" pour naviguer et sélectionner. Ces techniques ont été conçues dans un but plus large que la sélection d'un nœud et, en particulier, pour des tâches cognitives plus complexes comme la comparaison de plusieurs nœuds, la recherche d'un ancêtre commun, etc. Les évaluations qui comparent ces visualisations [142, 103] ont toutes révélé qu'il n'existe pas de meilleure visualisation pour toutes les tâches.

Au niveau de l'interaction pour les tâches de sélection d'un nœud, seuls les travaux autour des menus hiérarchiques, qui sont bien souvent des hiérarchies relativement petites, ont réellement innové. Les deux travaux les plus innovants sont les *pie menus* [42], qui peuvent être augmentés par un mécanisme de reconnaissance gestuelle pour obtenir des *marking menus* [105], et les menus de crossY [10] qui utilisent le franchissement. Les *pie menus*, déjà évoqués, sont des menus circulaires qui permettent d'obtenir un temps de sélection constant quel que soit l'élément sélectionné. Cependant, ces menus ne passent pas à l'échelle : ils sont efficaces lorsque le nombre d'éléments est pair et n'excède pas 8 par niveau. CrossY n'aborde que les listes textuelles et change la structure de la hiérarchie sous-jacente puisqu'il réorganise les éléments d'un même niveau selon une hiérarchie basée sur l'ordre lexicographique afin de pouvoir sélectionner un nœud lorsque la taille de la liste dépasse la taille de l'affichage disponible.

Le but de ControlTree Dans les interactions courantes, il est fréquent qu'un utilisateur ait besoin de sélectionner un fichier dans son système de fichiers ou de sélectionner une commande fréquente dans un grand menu hiérarchique (par exemple, l'application Adobe Illustrator contient un menu intitulé "texte" divisé en trois niveaux hiérarchiques et dont les sous-menus peuvent contenir jusqu'à 150 items). Sur ces deux exemples, l'utilisateur cherche à sélectionner le plus rapidement possible un nœud dans une grande hiérarchie dont certaines parties lui sont familières. Les techniques courantes (Microsoft Windows Explorer et les menus hiérarchiques linéaires) privent l'utilisateur de son contexte de travail car elles peuvent utiliser une grande partie de la surface de travail et sont donc ainsi source d'oublis et d'erreurs. ControlTree a pour but d'optimiser le temps nécessaire à la sélection d'un nœud mais également de minimiser la surface d'affichage utilisée afin de préserver la visibilité de la surface de travail. À chaque instant, ControlTree n'affiche que les nœuds nécessaires à la navigation vers un nœud "connu" : ceux déjà traversés qui constituent le chemin jusqu'au nœud courant et les nœuds qui sont les fils directs du nœud courant afin de naviguer vers le niveau inférieur. Il s'agit donc d'une succession d'acquisitions CIS (choix du prochain nœud puis l'atteindre) et de validations (valider ce prochain nœud pour passer au niveau inférieur). Nous cherchons donc à optimiser des arcs CIS et des nœuds CIS dont le degré dépend du nombre de fils du nœud graphique courant. La figure 3.22 montre la structure CIS pour la sélection d'un nœud au niveau suivant. La section suivante décrit comment ControlTree est le résultat de l'optimisation de cette structure.

ControlTree : les optimisations

Lors de la conception de ControlTree, nous nous sommes attachés à réaliser une technique pour une représentation dynamique nœud-lien. À partir du nœud courant, ControlTree affiche non seulement les fils directs et le chemin mais aussi les frères directs du nœud courant pour offrir un peu plus de contexte à l'utilisateur. L'affichage est constamment mis à jour en fonction du nœud courant et offre la possibilité à l'utilisateur de naviguer dans trois zones : la zone des parents (*parent zone*), la zone des frères (*sibling zone*) et la zone des fils (*child zone*).

Disposer les étiquettes des nœuds pour une représentation nœud-lien est un problème difficile. La figure 3.23 montre l'affichage que nous avons utilisé. L'arbre est affiché horizontalement et les nœuds d'un même niveau sont alignés verticalement pour améliorer la lisibilité des relations. Cet affichage permet d'afficher horizontalement les étiquettes des fils directs du nœud courant (sur la figure 3.23, le nœud courant est $Dental$) et d'offrir ainsi une très bonne lisibilité pour ceux-ci. Cependant, afficher

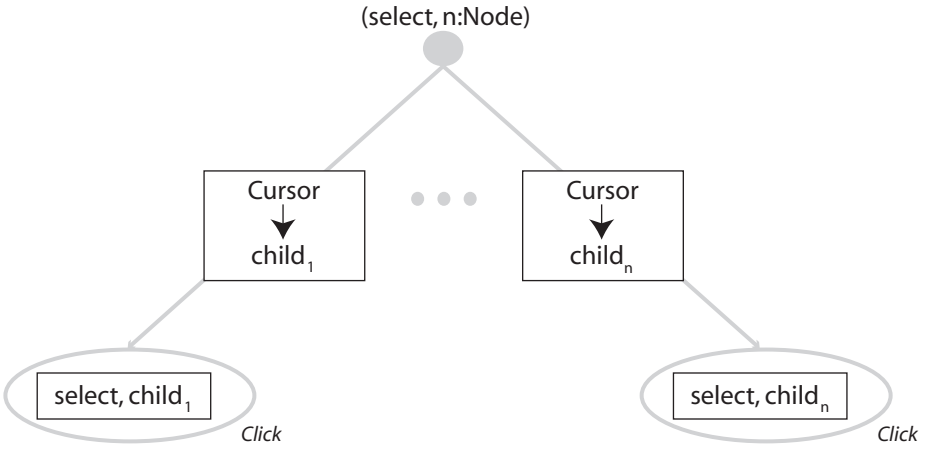


FIG. 3.22 : Structure CIS pour la sélection d'un nœud au niveau suivant avec une technique type "point-and-click"

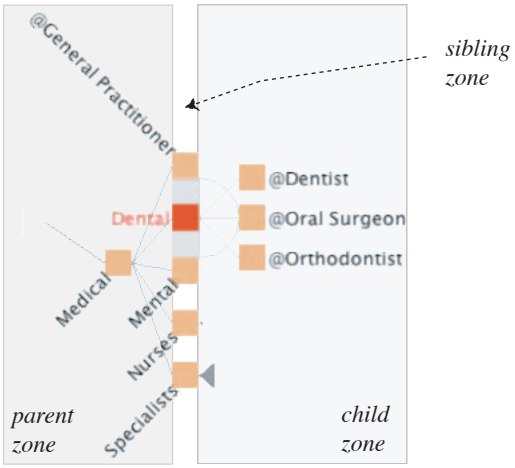


FIG. 3.23 : L'affichage nœud-lien dynamique de ControlTree

des étiquettes horizontales pour les frères et les nœuds du chemin sans occlusion demande d'utiliser des algorithmes de placement avancés et, bien souvent, l'affichage produit occupe beaucoup de place à l'écran. Pour résoudre ce problème, ControlTree affiche les étiquettes des nœuds des zones des parents et des frères avec un angle de 45 degrés. Cet affichage a également la propriété de ne pas obstruer la zone d'interaction la plus courante : celle qui va du nœud courant à un de ces fils. En effet, alors que les étiquettes des fils sont alignées à gauche, les étiquettes des parents et frères sont alignées à droite.

Pour optimiser les nœuds, nous avions la possibilité d'agir à deux niveaux :

1. Réduire le temps de choix en agissant sur le nombre de fils des nœuds appartenant à la séquence d'interaction. En effet, en changeant la structure de l'arbre, il est possible de réduire le degré des nœuds appartenant aux chemins souvent empruntés en augmentant celui des nœuds qui sont, au contraire, rarement visités. Ainsi, le temps prédit par la loi de Hick-Hyman qui est proportionnel au degré des nœuds se trouve réduit. Cependant la structure hiérarchique a bien souvent une sémantique liée à l'application sous-jacente et il peut être gênant, voire impossible, de la remettre en question.
2. Réduire le temps constant qui étiquette chaque nœud (les 200 ms nécessaires pour un clic de souris par exemple). Accot et Zhai ont montré que franchir un but de hauteur W peut être plus rapide [4] que pointer une cible de largeur W (dans le cas où le mouvement est continu et le but orthogonal à la direction du mouvement). Dans ControlTree, l'utilisateur navigue dans l'arbre par un geste continu en franchissant les nœuds un par un pour les ouvrir. En termes CIS, ceci se traduit par l'économie du coût constant de chaque nœud sur le chemin.

Pour optimiser les arcs, nous avons décidé de diminuer le temps de pointage en utilisant les principes d'OrthoZoom. En effet, bien que nous utilisons un affichage dynamique, il est au minimum nécessaire de rendre tous les nœuds du niveau suivant et le nœud du parent visibles afin de pouvoir naviguer partout. Cependant, les nœuds du niveau suivant peuvent être très nombreux, ce qui implique une réduction de la taille de ceux-ci pour les rendre visibles dans le cadre d'une représentation nœud-lien. Atteindre un nœud devient alors une tâche de pointage de haute précision. Nous avons donc décidé d'utiliser le principe d'OrthoZoom, présenté à la section précédente, qui est une technique efficace pour des tâches de sélection de haute précision et qui ne requiert qu'un périphérique de pointage standard.

La sélection d'un nœud avec ControlTree se fait donc par un geste continu en franchissant les nœuds du chemin un par un. Chaque fois qu'un nœud est franchi, celui-ci devient le nœud courant et l'affichage est alors dynamiquement actualisé.

ControlTree : l'interaction en détails

Ici, nous décrivons en détails l'interaction avec ControlTree illustrée par la figure 3.24 et, en particulier, les optimisations des actions de pointage :

- le mécanisme de magnétisme (*snapping*)
- l'adaptation d'OrthoZoom à la représentation nœud-lien

Le snapping : les sélections faciles sont triviales ControlTree prend en compte la position relative du curseur par rapport au nœud courant pendant l'interaction pour faciliter l'acquisition des nœuds environnants. En effet, le nœud courant est entouré de zones actives qui déclenchent différentes actions. La figure 3.25 liste les actions possibles en fonction des différentes zones actives. La taille de ces zones dépend de trois seuils :

1. Dans la zone des parents, si l'utilisateur va au-delà du seuil vertical t_{parent}, le nœud parent est sélectionné par un mécanisme de *snapping* (le nœud se déplace pour venir sous le curseur) ; sinon le nœud parent est mis en surbrillance.

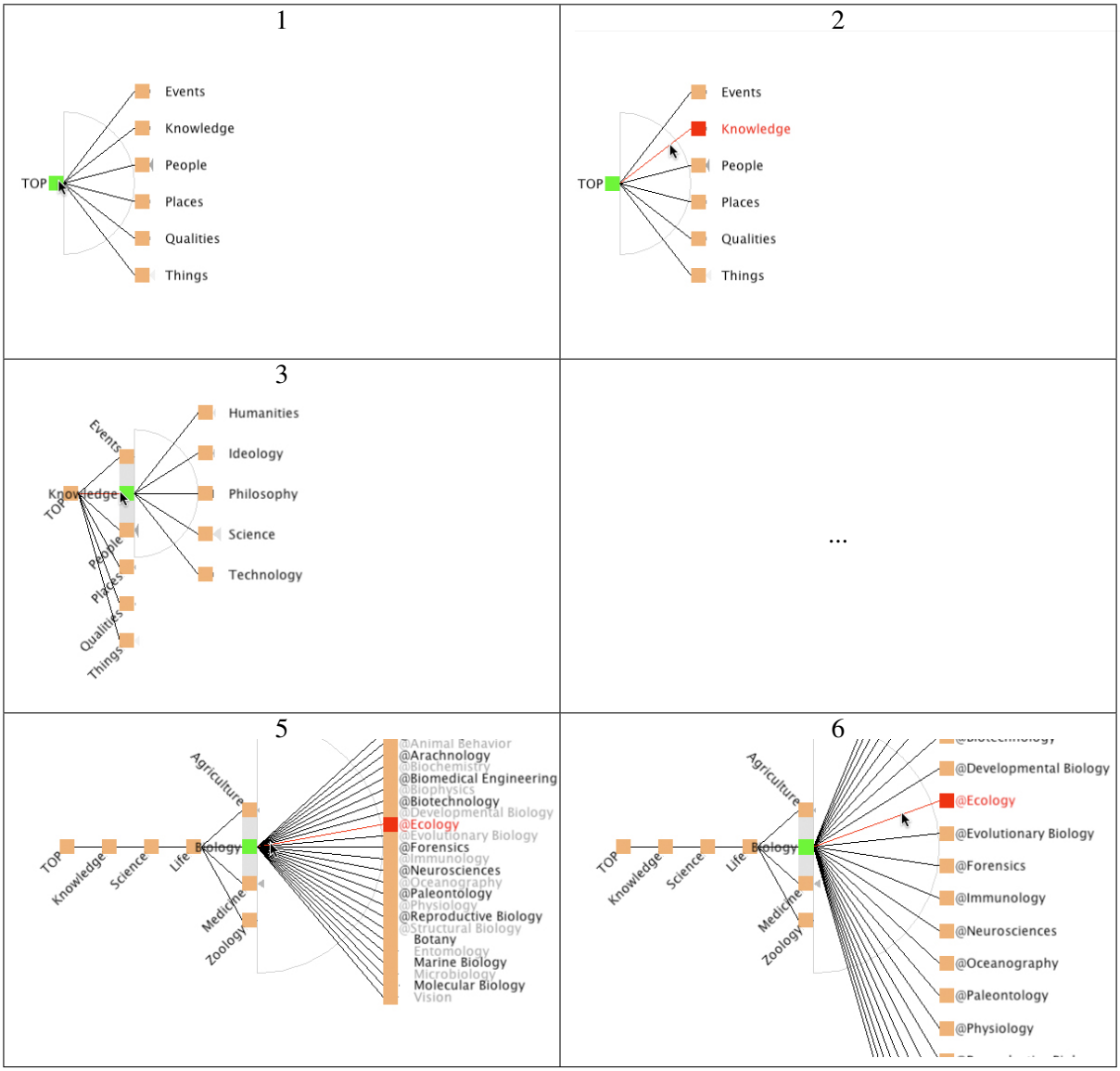


FIG. 3.24 : Une interaction typique avec ControlTree

2. Dans la zone des frères, si l'utilisateur va au-delà du seuil horizontal $t_{sibling}$ du haut (resp. bas), le frère du haut (resp. bas) est sélectionné par un mécanisme de *snapping* ; sinon le frère du haut (resp. bas) est mis en surbrillance.
3. Dans la zone des fils, si l'utilisateur va au-delà du seuil semi-circulaire t_{child}, le fils le plus proche est sélectionné par un mécanisme de *snapping* ; sinon le fils le plus proche est mis en surbrillance.

La *distance à un fils* c est la distance entre le curseur et la ligne qui représente le lien du nœud courant vers c. Le *fils le plus proche* est donc celui qui minimise cette distance par rapport à la position courante du curseur.

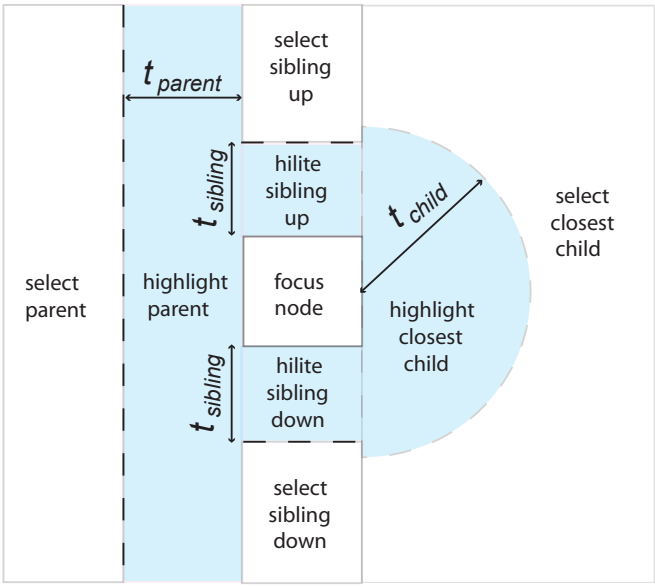


FIG. 3.25 : Zones d'interaction autour du nœud courant

L'affichage horizontal de ControlTree peut entraîner des distances élevées entre le nœud courant et les fils qui sont aux extrémités. Le mécanisme de *snapping* permet de résoudre ce problème en "apportant" le nœud sous le curseur au lieu que le curseur aille au nœud. Cependant, le *snapping* n'est approprié que si la probabilité de faire une mauvaise anticipation est faible (c'est-à-dire lorsque la difficulté de sélection est assez faible) afin d'éviter les erreurs. La prochaine section décrit comment une adaptation d'OrthoZoom nous permet d'adapter l'affichage afin de n'avoir que des sélections dans un intervalle de difficulté raisonnable.

OrthoZoom : les sélections difficiles sont possibles ControlTree prend également en compte la trajectoire de l'utilisateur pour faciliter l'acquisition d'un nœud dans les cas difficiles. En effet, il n'existe pas de valeur unique pour t_{child} qui facilite la sélection dans tous les cas. La figure 3.26 illustre le problème : la précision angulaire requise pour sélectionner un nœud augmente avec le degré de son parent.

Pour contourner ce problème, la figure 3.27 illustre deux options :

1. augmenter l'espacement entre les nœuds d'un même niveau.
2. augmenter la valeur du seuil t_{child}.

Prises indépendamment, ces solutions sont assez simplistes puisque dans les deux cas, l'espace d'affichage disponible est une limite rapidement atteinte. Dans le premier cas, il y a plus d'espace consommé verticalement. De plus, dans ce cas, certaines sélections sont plus faciles que d'autres : la précision angulaire requise pour sélectionner un nœud dépend de sa position d'affichage. Dans le second cas, il y a plus d'espace consommé horizontalement, limitant ainsi le nombre de niveaux hiérarchiques qui peuvent être affichés dans la vue.

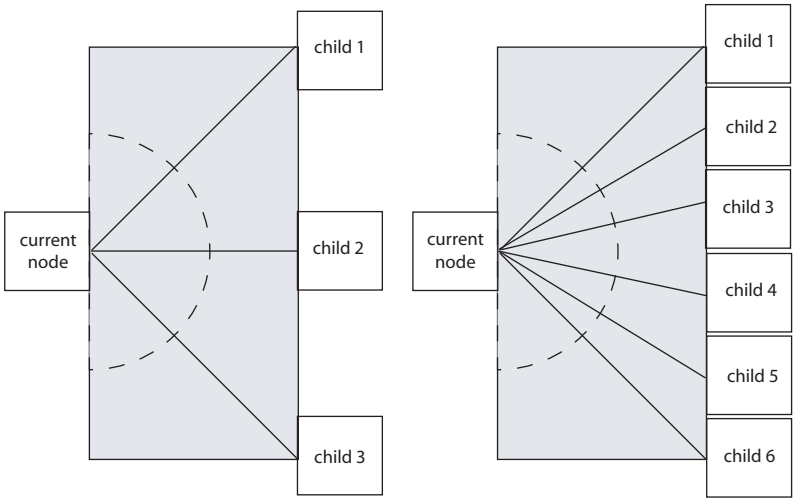


FIG. 3.26 : Degré et précision angulaire dans le cas d'un seuil fixe

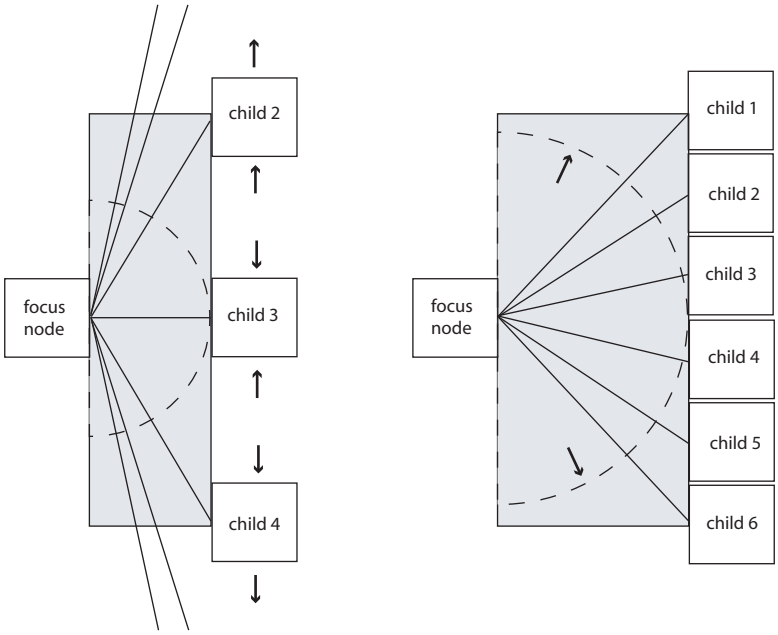


FIG. 3.27 : Stratégies simplistes pour faciliter une sélection

Pour résoudre ces problèmes, ControlTree propose une version d'OrthoZoom adaptée aux représentations nœud-lien. Alors qu'OrthoZoom utilise les distances sur la dimension orthogonale pour ajuster le facteur de zoom et la dimension colinéaire pour contrôler les déplacements, ControlTree utilise la distance entre le curseur et le nœud courant pour contrôler l'espacement entre les nœuds et la direction pour choisir les nœuds qui restent dans la surface d'affichage. Cette approche résout les deux problèmes exposés ci-dessus en contrôlant dynamiquement le nœud-cible potentiel et la précision angulaire autour de ce nœud en fonction du mouvement de l'utilisateur. À chaque mouvement du curseur, ControlTree cherche le nœud le plus proche c et ajuste l'espacement autour de c tout en préservant la position de c. c repousse progressivement ses frères pour être plus facilement accessible (Figure 3.28). Ainsi, lors de la navigation, l'utilisateur garde le regard sur le prochain nœud à atteindre et suit la ligne droite qui l'y mène.

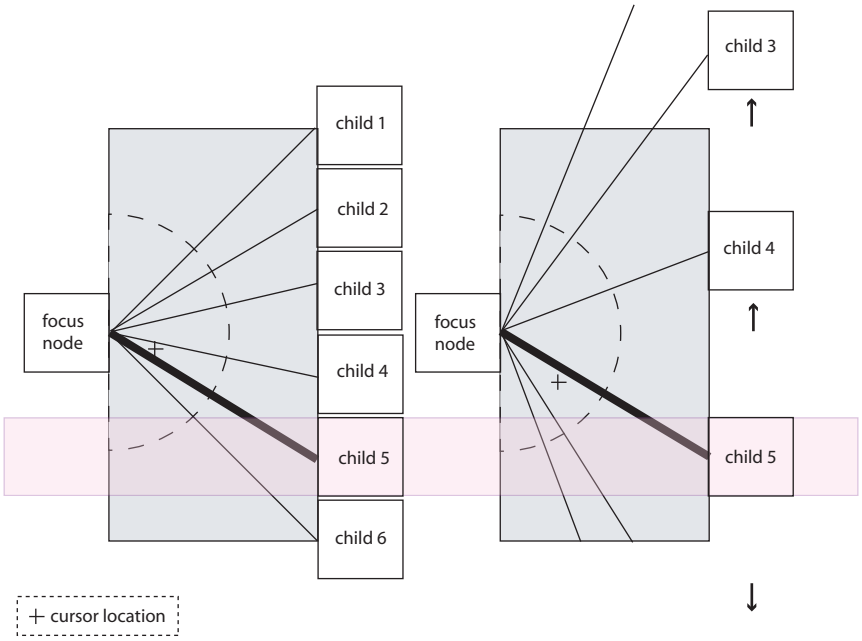


FIG. 3.28 : Stratégie de ControlTree pour faciliter une sélection

Bien que cette adaptation permette d'atteindre plus facilement un nœud même pour de grands degrés, la difficulté augmente avec le degré du nœud si la valeur de t_{child} est identique pour tous les nœuds. En effet, t_{child} définit l'aire de contrôle disponible pour contrôler l'espacement entre les nœuds et influe donc sur l'utilisabilité. Par exemple, l'utilisateur peut être déstabilisé si, dans le cas d'un nœud ayant un grand degré, un déplacement d'un pixel fait passer le nombre de nœuds visibles de 20 à 5. De plus, si le rapport entre le degré et la taille de l'aire devient très grand, certains nœuds peuvent devenir inaccessibles s'ils disparaissent de la surface d'affichage avant que l'utilisateur n'ait eu une chance de les atteindre. Pour résoudre ce problème, l'algorithme de placement de ControlTree ajuste la valeur de t_{child} en fonction du degré du nœud courant afin de fournir une difficulté bornée pour tous les nœuds de l'arbre. L'algorithme

de ControlTree dépend des paramètres de calibration suivants :

1. L'espacement minimal entre les nœuds, S_{min}
2. L'espacement maximal entre les nœuds, S_{max}

Ainsi, lorsqu'un nœud n de degré deg_n devient le nœud courant, t_{child} doit vérifier la condition (C) pour garantir que chaque nœud puisse être atteint.

$$t_{child} = x \times d \text{ avec } r^x \leq deg_n \text{ (C)}$$

Donc, pour déterminer le placement des fils de n qui est situé en (x_n, y_n) dans une surface d'affichage de dimension $w \times h$, ControlTree calcule la plus petite valeur de x qui vérifie (C) puis t_{child} qui est donc une fonction logarithmique de deg_n. ControlTree peut alors afficher les fils de n en fonction de t_{child} et de la surface disponible.

1. Calculer la valeur minimale de t_{child} qui vérifie (C).
2. Calculer la hauteur maximale disponible : $h_{max} = max(y_n, (h - c_y))$.
3. Calculer l'espace courant s entre les fils de n :
   $s = h_{max}/deg_n$ si $h_{max}/deg_n \leq S_{min}$
   $s = S_{min}$ si $h_{max}/deg_n < S_{min}$

Une fois ce placement initial fait, lorsque le curseur est à une distance d de n, l'espacement entre les nœuds vaut $s + D \times step$ avec $step = (S_{max} - s)/t_{child}$. Enfin, afin de permettre à l'utilisateur de passer facilement d'un nœud à l'autre, il peut utiliser la molette de la souris pour faire défiler les fils de n en conservant l'espacement courant entre les nœuds.

Pour résumer, ControlTree est le résultat d'une optimisation du graphe CIS d'une simple interaction "point-and-click" pour sélectionner un nœud dans une représentation nœud-lien. Nous projetons d'évaluer ses performances formellement en menant une expérimentation contrôlée pour comparer ControlTree à d'autres représentations nœud-lien comme Microsoft Windows Explorer. Nous utiliserons une tâche de sélection d'un nœud dans une partie familière de la hiérarchie puisque CIS, qui nous a servi de base pour optimiser, prend en compte l'effet des actions motrices sur la performance et non pas les aspects liés à l'expérience avec une technique donnée ou encore les capacités à identifier l'emplacement d'un nœud depuis un niveau supérieur.

3.3 Explorer l'espace de conception : Travailler au niveau de la structure de graphe

Dans cette section, nous montrons au travers de quelques exemples comment CIS peut aider à générer des techniques d'interaction efficaces. L'approche que nous proposons diffère des approches de génération automatique d'interfaces qui limitent la place accordée à l'inventivité du concepteur. Au contraire, les graphes d'interaction CIS sont des supports de réflexion pour les concepteurs. D'une part, nous avons vu qu'à partir d'un graphe existant, ils peuvent identifier quelles actions optimiser et donc sur quelles variables agir. D'autre part, les concepteurs peuvent imaginer une nouvelle technique d'interaction en commençant par travailler sur une structure de graphe afin de conférer des propriétés particulières à une future technique.

3.3.1 La génération automatique de techniques d'interaction

Toutes les approches automatiques pour la génération d'interfaces s'appuient sur une base de connaissances décrites formellement et utilise des méthodes d'optimisation pour composer au mieux des éléments d'interaction. Par exemple, l'outil Toto [34] est un moteur d'inférence pour construire une technique "optimale" à partir d'une base de connaissances. Une "technique" pour Toto est une séquence d'actions sur les périphériques d'entrée. La base de connaissances incluent des *concepts de tâches* (c'est-à-dire des correspondances entre une tâche primitive et une action sur un périphérique d'entrée), des *tâches* (c'est-à-dire des tâches primitives organisées hiérarchiquement à la manière des modèles GOMS), une *taxonomie des périphériques d'entrée* (basée sur les différents modèles présentés à la section 2.1.1), un *catalogue de techniques*, des *contraintes de conception* (l'ensemble des périphériques disponibles par exemple) et, enfin des *heuristiques de conception* basés sur des facteurs humains. Le concepteur doit donc spécifier ses tâches et ses périphériques dans la base de Toto ou les créer si la base ne les contient pas. Toto construit alors la meilleure technique pour une tâche donnée en se basant sur ses heuristiques de conception. Par exemple, pour la tâche *withHand – continuousTouch – surfaceDrag – letGo* sélectionnée à partir de la hiérarchie de tâches primitives de la figure 3.29, Toto recommande la technique *mouseMove – buttonPress – mousePressMove – buttonRelease* parmi les techniques listées dans le tableau de la figure 3.29. Toto pourrait donc donner naissance à des compositions d'actions intéressantes mais, se basant sur un ensemble de connaissances ad hoc, le "degré d'inovativité" reste limité. De plus, Toto décrit l'interaction à un très bas niveau d'abstraction, ce qui implique une description très détaillée et donc coûteuse chaque fois qu'un élément (comme un périphérique d'entrée par exemple) doit être ajouté à la base.

Class Hierarchy	Instances
Action	
PhysicalAction	
Move	surfaceMove, levitate, rotate
Drag	one-dDrag, surfaceDrag
Grasp	
Reach	withFinger, withHand, blindReach
Release	liftOff, letGo
Touch	continuousTouch, tap
Entity	
Observer	
Manipulator	hand, foot, eye
Physical object	ball, dog, window

Reach Object	Touch Object	Drag Object	Release Object
mouseMove	buttonPress	mousePressMove	buttonRelease
touchpadMove	buttonClick	mouseMove	buttonClick
stylusMove	fingerTouchOn	touchpadMove	fingerTouchOn
touchScreenMv	stylusClick	stylusMove	stylusClick
	wheelSpin	touchscreenMove	

FIG. 3.29 : À gauche, les entrées : une hiérarchie de tâches primitives disponibles ; À droite, les sorties : les techniques suggérées (Illustrations extraites de [34])

Un second exemple est le système SUPPLE [68] pour construire des interfaces adaptatives, c'est-à-dire des interfaces qui pourront s'ajuster automatiquement en fonction de l'utilisateur. Cet outil s'appuie sur une définition formelle des spécifications de l'interface, un modèle des périphériques d'entrée et de sortie, une description formelle des widgets standards et des traces d'utilisation pour adapter une interface existante aux besoins de l'utilisateur. La figure 3.30 montre un exemple d'une interface avant et après son adaptation, l'interface adaptée facilitant l'accès aux commandes de l'application les plus utilisées en les mettant dans un onglet au premier plan et sa taille s'est adaptée pour un plus petit dispositif d'affichage.

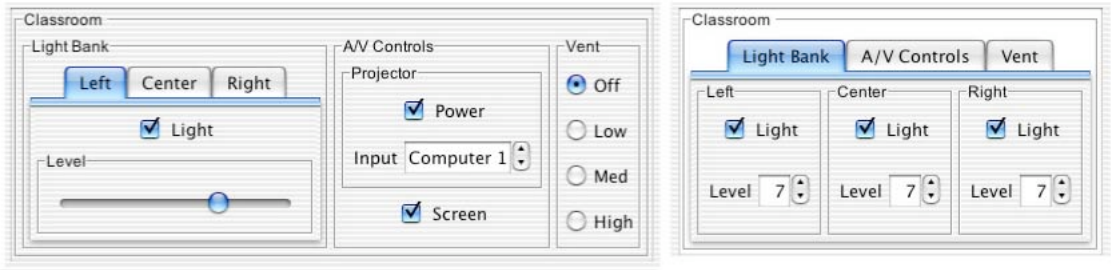


FIG. 3.30 : Exemple d'interface adaptative (Illustration extraite de [68])

Alors que ces approches visent plus à composer de manière optimale des éléments d'interaction connus selon un certain nombre de critères et en respectant un certain nombre de contraintes, l'approche présentée ici est de plus haut niveau et vise plus à mettre en avant des espaces que le concepteur peut explorer. En d'autres termes, la description d'une technique d'interaction avec CIS permet d'identifier des variables sur lesquelles le concepteur peut agir. Le concepteur peut réfléchir à deux niveaux sur une technique : il peut chercher à optimiser les actions d'un graphe donné comme nous l'avons vu ou chercher à mettre en place une structure morphologique de graphe particulière. Nous avons vu que, dans le premier cas, il agit sur les variables de la loi empirique sous-jacente à l'action considérée alors que nous allons voir que, dans le deuxième cas, il agit sur les propriétés morphologiques de la technique.

3.3.2 Concevoir une nouvelle structure CIS

Dans cette section, nous illustrons au travers de deux exemples comment le concepteur peut imaginer de nouvelles techniques à partir d'une structure de graphe CIS. Dans le premier exemple, il part d'un graphe CIS d'une technique classique et joue sur la structure pour envisager de nouvelles alternatives. Dans le second exemple, il construit un graphe à partir de deux graphes existants afin d'obtenir une technique qui regroupe les propriétés avantageuses des deux autres.

Changer les propriétés morphologiques

Partons de l'exemple de l'interaction iconique classique pour effacer un fichier. La figure 3.31 montre le graphe de cette technique qui permet d'activer la commande (delete, $f \in$ File).

Le concepteur peut envisager de conserver la même technique mais changer l'ordre des actions pour obtenir une technique dont le graphe CIS est celui qui est à gauche de la figure 3.32. Puis, à partir d'un tel ordre, il semble naturel d'offrir de la persistance pour obtenir le graphe qui est à droite de la figure 3.32. Ainsi, l'utilisateur commence par sélectionner l'icone d'un outil de suppression et le fais glisser successivement vers les icones des fichiers qu'il veut effacer. Cet ordre incite à utiliser la métaphore de l'aspirateur plutôt que celle de la corbeille comme illustré par le scénario de la figure 3.32. Ainsi, il évite les aller-retours successifs entre les icones des fichiers à effacer et l'icone de l'outil de suppression.

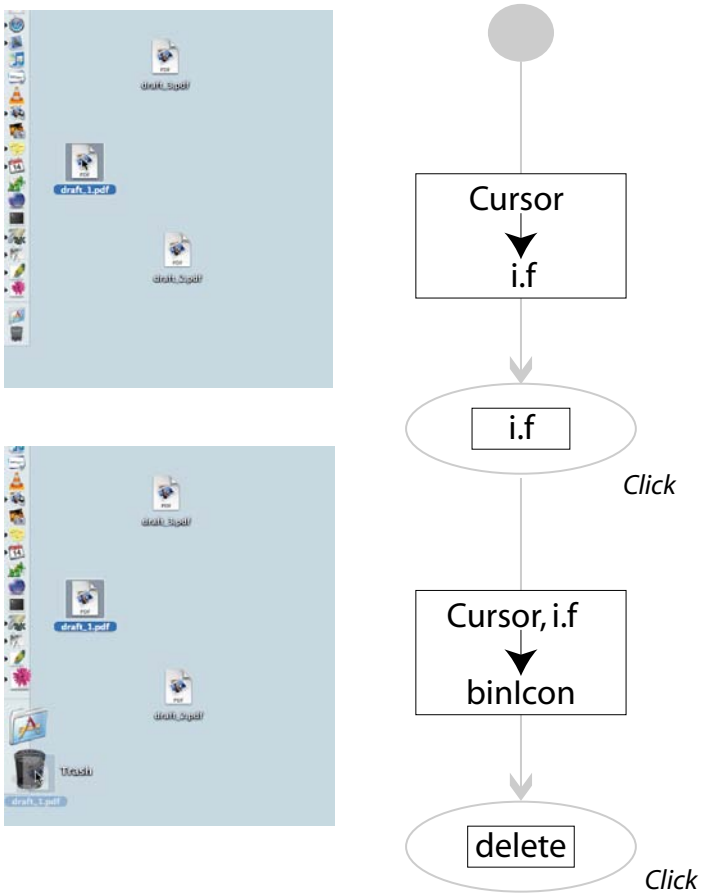
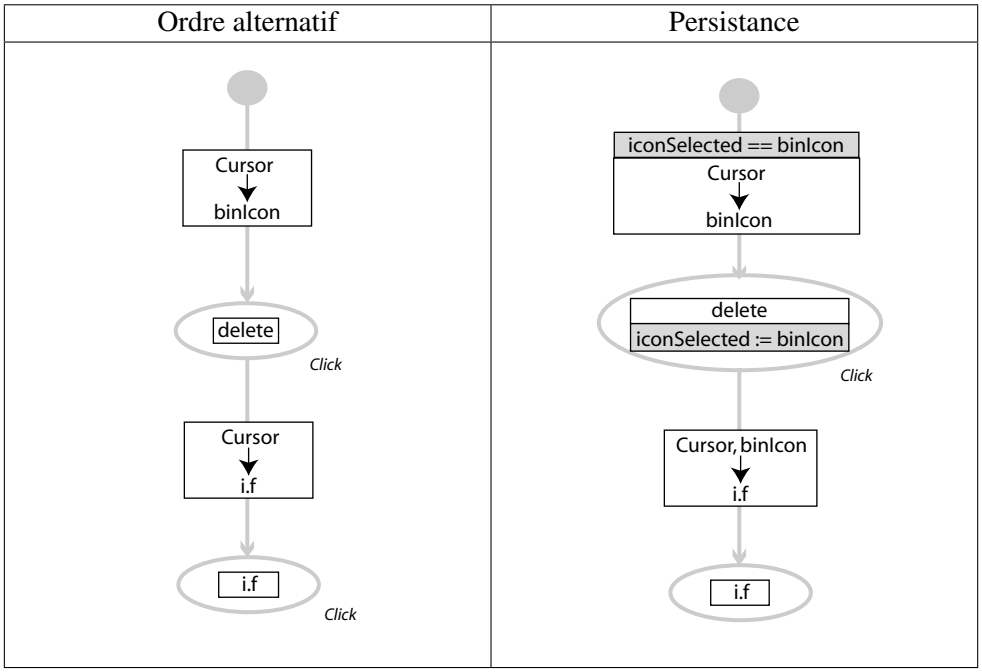


FIG. 3.31 : Le graphe d'interaction de l'interaction iconique classique pour effacer un fichier

À quoi ressemblerait cette technique d'interaction iconique si on développait ou fusionnait ses éléments ? La partie gauche de la figure 3.33 montre le graphe développé de cette technique, si à la manière d'HabilisDraw [8], l'utilisateur pouvait disposer plusieurs corbeilles sur sa surface de travail. Il fait alors un choix entre plus d'éléments mais la distance moyenne un icone de fichier d'un icone de corbeille se trouve réduite. La partie droite de la figure 3.33 montre, au contraire, le graphe de cette technique dans laquelle les éléments ont été fusionnés. Pour obtenir une telle technique, il peut imaginer proposer un outil de suppression apposé à chaque icone de fichier pour spécifier les deux éléments de la commande en une seule action. Bien évidemment, ceci ne constitue qu'une base de réflexion et CIS ne prend pas en compte tous les aspects de l'interaction. Dans le dernier exemple, il faut biensûr considérer que l'utilisateur peut effacer un fichier par erreur s'il clique sur l'icone de la corbeille au lieu de celui du fichier pour l'ouvrir et donc envisager une boîte de dialogue pour s'assurer du bon choix de l'utilisateur par exemple. Aussi, dans l'exemple sur le développement qui multiplie le nombre d'icones à l'écran, il faut prendre en compte l'espace écran supplémentaire nécessaire qui peut être impossible selon la taille de l'écran par



(1) (2) (3)

FIG. 3.32 : Changer l'ordre et conférer de la persistance

exemple.

Combiner deux graphes CIS

Regardons de nouveau rapidement les complexités de la palette fixe (FP) et de la toolglass (TG) de la figure 3.8. La palette fixe est plus efficace dans un contexte de recopie alors que c'est la toolglass qui l'emporte dans un contexte de résolution de problème. Ces performances semblent attribuées à leur propriété de persistance pour la palette fixe et de parallèlisme pour la toolglass comme le laissent penser

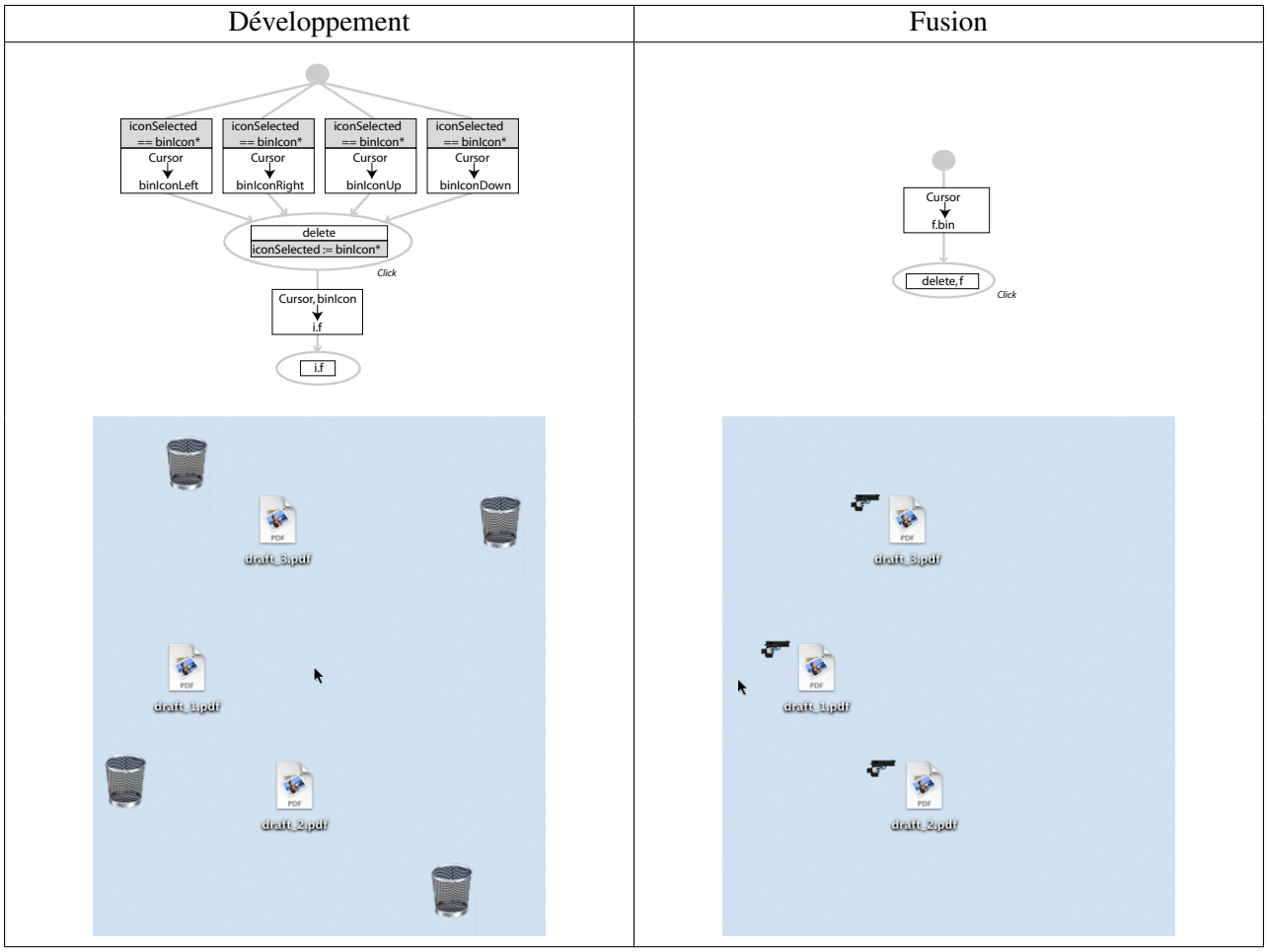


FIG. 3.33 : Développer ou fusionner

les OMDs de la figure 3.9. Pourrions-nous obtenir une technique qui combine ces deux propriétés (Figure 3.34) ?

Il suffit d'augmenter le palette fixe d'un second curseur par exemple. En fait, une telle technique existe déjà : la palette bimanuelle (BP), implémentée dans le projet CPN2000 [26]. C'est une technique dans laquelle l'utilisateur dispose de deux curseurs, un premier pour sélectionner un outil persistant dans la palette avec une main et un second pour appliquer cet outil sur l'objet ou la position d'intérêt avec l'autre main. La table de la figure 3.35 montre que la palette bimanuelle combine la persistance de la palette fixe et le parallélisme de la toolglass.

L'utilisation de SimCIS nous permet de vérifier si, par la combinaison des deux propriétés, nous obtenons bien une technique plus efficace dans les deux contextes de recopie et de résolution de problème. La figure 3.36 reporte les complexités en temps obtenues pour les trois techniques dans chacun des deux

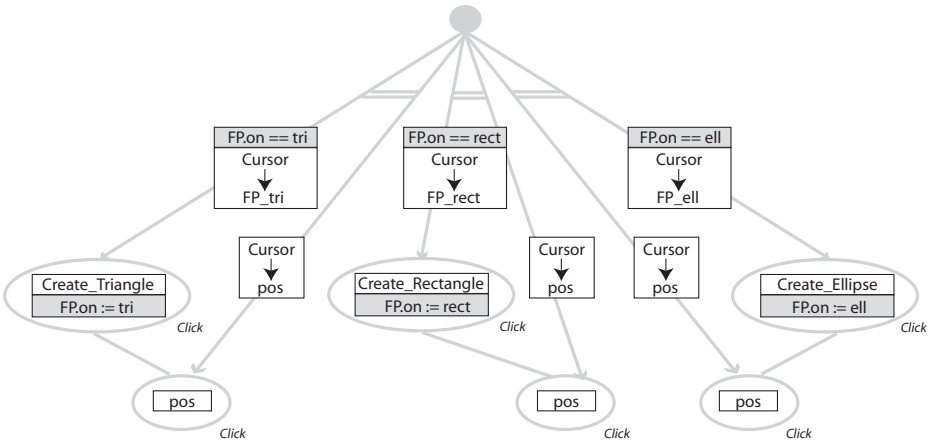


FIG. 3.34 : Un graphe d'interaction combinant parallèlisme et persistance

	FP	BP	TG
Persistance	Oui	Oui	Non
Parallélisme	Non	Oui	Oui

FIG. 3.35 : Propriétés des trois techniques évaluées

contextes et confirme que la palette bi-manuelle est plus efficace quel que soit le contexte d'utilisation. Les OMDs de la figure 3.37 montrent bien que les changements d'outils ne provoquent que des déplacement très locaux (dans l'espace de la palette) et sont donc peu coûteux.

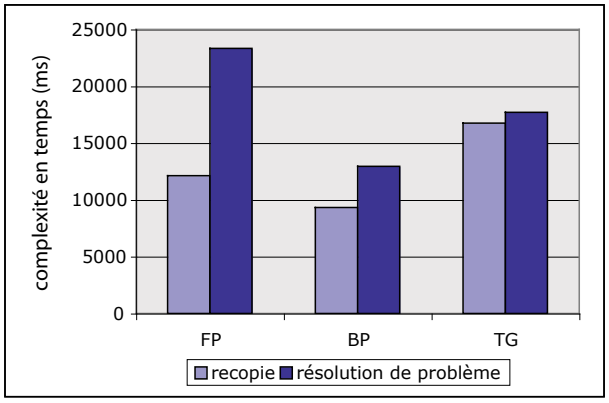


FIG. 3.36 : Complexité en temps de trois techniques dans deux contextes

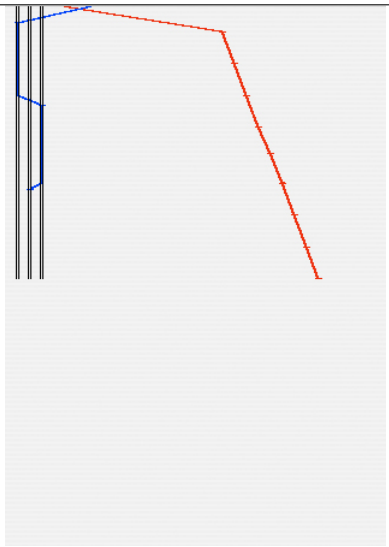
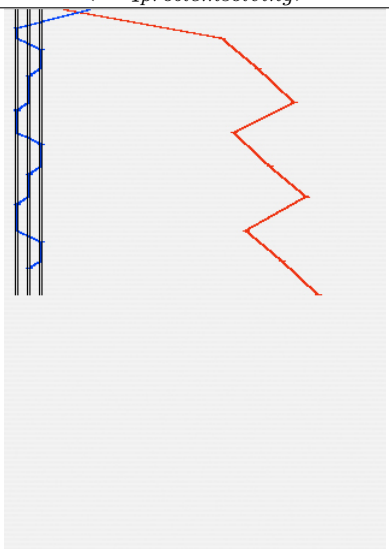
	Contexte de recopie (Seq_{copy})	Contexte de résolution de problème ($Seq_{problemsolving}$)
Palette Bi-manuelle (BP)		

FIG. 3.37 : Object Movement Diagrams pour la palette bi-manuelle : à gauche, contexte de recopie ; à droite, contexte de résolution de problème

3.4 Conclusion

La description d'une technique d'interaction avec CIS permet d'isoler les différentes actions que comporte une technique d'interaction. L'efficacité de chacune des actions est évaluée en utilisant des lois empiriques pour lesquelles le concepteur peut facilement identifier les variables sur lesquelles agir pour optimiser ces actions. De plus, ces lois empiriques servent de cadre expérimental solide pour évaluer les nouvelles techniques d'interaction et ont donc donné naissance à une partie de la littérature plus standardisée et donc plus facilement lisible. C'est notamment dans ce cadre expérimental que nous avons introduit une nouvelle technique de pointage efficace, OrthoZoom.

Une description CIS met en évidence les différentes actions d'une technique et encourage donc la génération de techniques qui résultent de l'optimisation de techniques existantes. Une fois les différentes actions identifiées, le concepteur peut consulter la littérature pour trouver quelles sont les techniques "atomiques" les plus prometteuses pour les actions de sa technique. Ainsi, nous avons optimisé chacune des actions d'une technique d'interaction pour naviguer dans des représentations nœud-lien pour créer la technique ControlTree. Aussi, les graphes CIS exhibant les propriétés morphologiques des techniques, le concepteur peut décider d'explorer l'espace de conception en envisageant d'autres structures pour imaginer ensuite des techniques qui collent à cette structure. Le niveau d'abstraction assez élevé de CIS laisse des espaces de conception ouverts et exhibe la structure morphologique des techniques pour encourager une approche de conception générative (i) en optimisant les actions et/ou (ii) en envisageant différents points dans cet espace et évaluer leur intérêt grâce à SimCIS.

Validation de CIS et Extensions

Dans ce chapitre, nous validons l'aspect prédictif du modèle CIS tel qu'il a été introduit jusqu'à maintenant. Pour ce faire, nous menons une expérimentation contrôlée dans laquelle les participants doivent utiliser des techniques d'interaction ayant des propriétés morphologiques différentes dans différents contextes d'utilisation afin de collecter des données empiriques que nous mettons en parallèle avec les prédictions fournies par le simulateur SimCIS. Nos résultats montrent que bien que SimCIS sous-estime le temps d'exécution, les schémas comparatifs, c'est-à-dire les relations entre contexte d'utilisation et performance d'une technique et les comparaisons entre techniques, sont identiques entre prédictions et observations empiriques. Nous montrons que les différences de valeurs observées entre les prédictions et les données empiriques peuvent être considérablement réduites si les coefficients des différentes lois empiriques sous-jacentes à SimCIS sont ajustés.

Nous abordons également les limites de l'aspect descriptif et prédictif de CIS en analysant la couverture par CIS et SimCIS des différentes familles de techniques d'interaction : interactions basées sur le pointage, sur le franchissement, interactions avec un monde multi-échelle, etc. Les limites de CIS sont établies par la connaissance actuelle des lois empiriques des différents types d'actions CIS. Ainsi, nous ne disposons pas à l'heure actuelle d'un modèle de choix dans une interface multi-échelle. Alors que la loi de Fitts reste valide pour le pointage multi-échelle [78], il est clair que la loi de Hick-Hyman ne convient pas à elle-seule à modéliser une tâche de choix dans un monde multi-échelle. En effet, la loi de Hick-Hyman est basée sur la perception visuelle et est donc applicable lorsque tous les objets sont visibles or, dans une interface multi-échelle, choisir un objet requiert une grande part d'actions motrices pour aller naviguer vers et inspecter les différents objets puisque, selon le niveau d'échelle et la position, ceux-ci ne sont pas toujours visibles. Nous proposons donc un cadre expérimental permettant d'étudier cette tâche de choix (ou de recherche) et menons une expérimentation préliminaire dans ce cadre. Nos résultats ne sont, pour l'instant, pas suffisants à l'élaboration d'un modèle prédictif de choix multi-échelle général mais constituent une première pierre et nous offrons des pistes et outils pour mener de nouvelles études qui permettront de consolider un tel modèle. L'étude de ces lois empiriques "atomiques" permet d'étendre la couverture de CIS afin d'améliorer non seulement l'aspect prédictif mais aussi l'aspect génératif en donnant la possibilité d'envisager des alternatives dans un spectre plus large.

4.1 Valider CIS

4.1.1 Validation du modèle : L'expérimentation

Nous avons déjà mentionné que les prédictions de SimCIS sont en accord avec les résultats rapportés précédemment dans la littérature à la section 3.1.2. Afin de tester plus rigoureusement la validité des prédictions fournies par SimCIS, nous présentons ici une expérimentation contrôlée [13] qui compare nos prédictions sur différentes techniques dans différentes séquences à des observations empiriques. Nous comparons la palette fixe (FP), la toolglass (TG) et la palette bimanuelle (BP) pour quatre séquences d'interaction opérationnalisant des contextes dans lesquels les objets d'interaction successifs sont proches ou distants et/ou les changements d'outil sont fréquents ou non.

Tâche

Un ensemble de formes graphiques (carrés verts, triangles bleus et cercles rouges) reliées par une ligne noire est initialement affiché à l'écran. La tâche de l'utilisateur consiste à effacer toutes les formes l'une après l'autre le plus rapidement possible. Pour effacer une forme, l'utilisateur doit sélectionner l'outil correspondant à la forme à effacer et appliquer cet outil sur la forme à effacer. Il dispose, à chaque instant, d'une seule technique parmi les trois évaluées. Chaque technique contient trois outils dont les icones correspondent respectivement à un des types de forme graphique (un carré vert, un triangle bleu ou un cercle rouge). Pour effacer un triangle bleu avec la palette fixe par exemple, l'utilisateur doit d'abord cliquer sur l'outil "triangle bleu" puis sur le triangle en question (figure 4.1). L'utilisateur doit de plus suivre l'ordre imposé par le chemin indiqué par la ligne noire pour effacer les formes.

Facteurs expérimentaux

Les facteurs de l'expérimentation sont organisés selon un plan $3 \times 2 \times 2 \times 3$ inter-participant. La liste ci-dessous résume les facteurs expérimentaux considérés et leurs différentes valeurs :

- $technique$: Palette Fixe (FP), Toolglass (TG) ou Palette Bimanuelle (BP) ;
- $groupement$: Groupé (G) ou Distribué (D) ;
- $ordre$: Radial (R), Spiral (S) ou Libre (F pour Free).
- $longueur$: 6 formes graphiques ou 18 formes graphiques ;

Les différentes valeurs des facteurs $groupement$, $ordre$ et $longueur$ ont été utilisées pour créer des séquences d'interaction qui opérationalisent plusieurs contextes ayant des caractéristiques différentes.

Dans tous les cas, les formes graphiques sont distribuées radialement autour du centre. Le facteur $Groupement$ comprend les conditions *Groupé* et *Distribué*. Dans la condition *Groupé* (G), les formes graphiques d'une même ligne ont toutes le même type alors que, dans la condition *Distribué* (D), les formes graphiques successives d'une même ligne ont toujours un type différent. Le facteur $Ordre$ comprend les conditions *Spiral* et *Radial*. Dans la condition *Spiral* (S), la ligne noire impose au participant d'effacer les formes selon un chemin en spirale alors que dans la condition *Radial* (R), la ligne noire impose d'effacer les formes ligne par ligne. La figure 4.2 illustre comment les combinaisons de valeurs $Groupement \times Ordre$ sont utilisées pour créer quatre séquences d'interaction aux caractéristiques différentes : deux objets d'intérêt successifs sont proches ($Near$) ou éloignés (Far) et les changements d'outil sont fréquents (Hi pour High) ou rares (Lo pour Low). Ces combinaisons ont été choisies afin que, pour chaque technique, il existe un contexte qui soit proche du contexte d'utilisation optimal. Par exemple, les séquences Lo sont des minima pour les palettes qui ont une propriété de persistance alors que les séquence $Near$ sont des minima pour les toolglasses qui sont principalement affectées par les distances entre objets.

La condition Libre (F) du facteur $ordre$ est introduite afin d'observer si les utilisateurs savent optimiser leurs interactions pour accomplir la tâche en un temps minimum. Dans cette condition, la ligne noire imposant l'ordre pour effacer les formes n'est pas affichée et les utilisateurs sont libres d'effacer les formes dans l'ordre de leur choix.

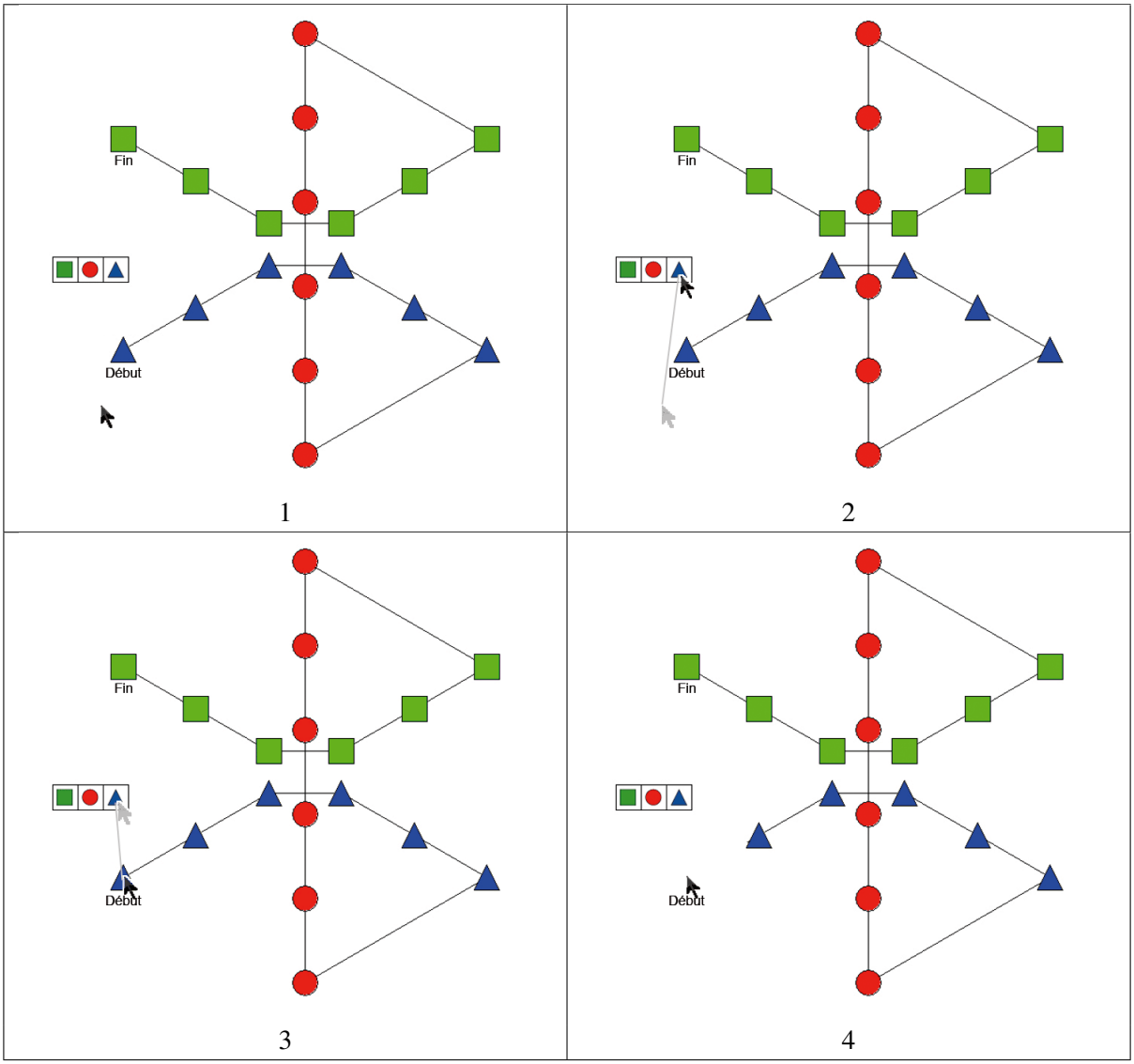


FIG. 4.1 : La tâche expérimentale

Le facteur $longueur$ sert à tester l'effet de planification sur les performances d'une technique. En effet, la *quantité de planification* est inversement proportionnelle à la charge cognitive du contexte. Par exemple, une tâche de recopie permet de planifier loin étant donné que la charge cognitive de la tâche est faible [72]. Si l'utilisateur est en mesure de planifier loin, il peut faire des choix en amont qui affecteront un plus grand nombre de ses interactions futures et donc avoir un gain sur la tâche globale d'autant plus grand que la quantité de planification est grande. Le facteur $longueur$ opérationalise la quantité de planification comme le nombre de manipulations dans la séquence d'interactions. Dans le cas $longueur = 6$, il y a 6 formes graphiques qui sont distribuées radialement autour du centre selon 3 lignes de 2 formes

alors que dans la cas $longueur = 18$, il y en a 18 qui sont distribuées selon 6 lignes de 3 formes.

Ces 4 séquences combinées à la condition $longueur$ nous permettent d'explorer 8 contextes cognitifs différents, ceux qui permettent de planifier loin ou non et les choix qui découlent d'un contexte : optimiser les distances entre objets au prix d'un plus grand nombre de changements d'outil ou, au contraire, minimiser les changements d'outil au prix de plus grandes distances à parcourir.

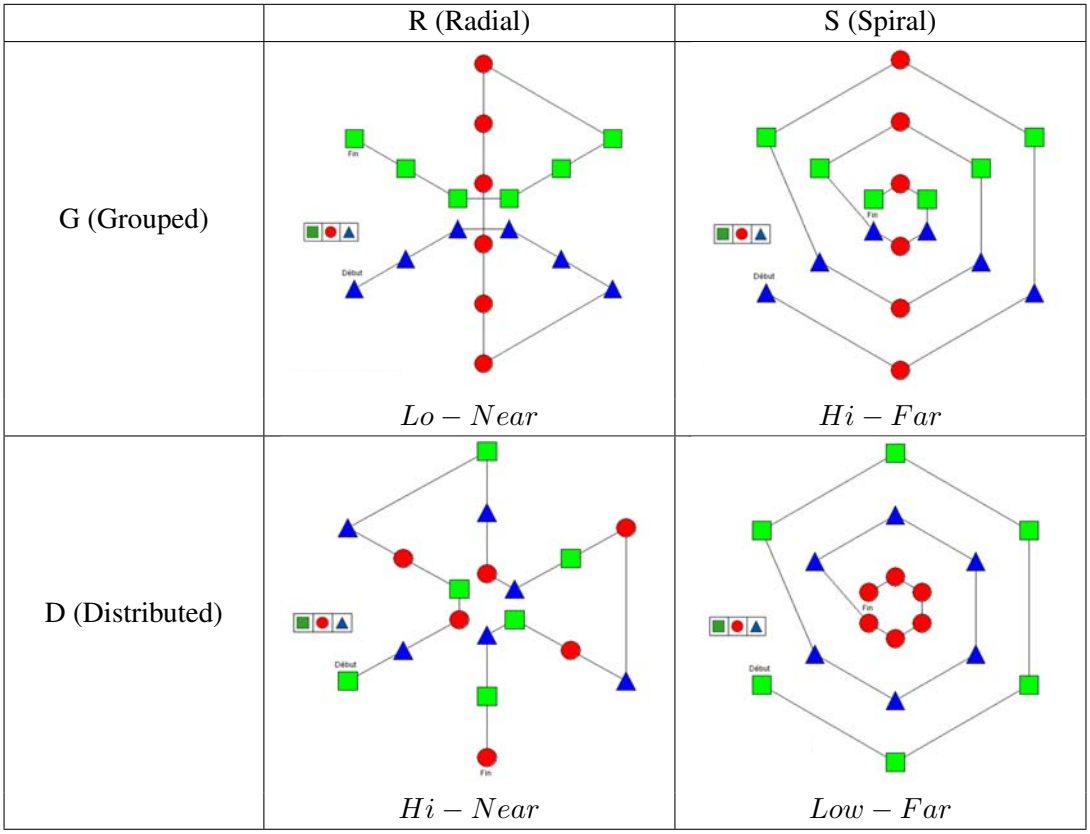


FIG. 4.2 : Les quatre types de séquences d'interaction

Prédictions et hypothèses

Nous avons utilisé SimCIS pour les trois techniques évaluées sur l'ensemble des séquences que nous venons de décrire. La figure 4.3 résume les complexités en temps que nous obtenons (Remarque : nous rapportons ici les prédictions telles qu'elles étaient au moment de l'expérience, nous verrons à la section 4.2 qu'elles ont été améliorées depuis).

À partir de ces prédictions nous faisons les quatre hypothèses suivantes que nous voulons tester avec cette expérimentation :

– (H_1) FP et BP sont toutes deux très sensibles au nombre de changements d'outil (Lo vs. Hi) à cause de leur propriété de persistance, le gain de temps étant d'autant plus grand que le nombre de

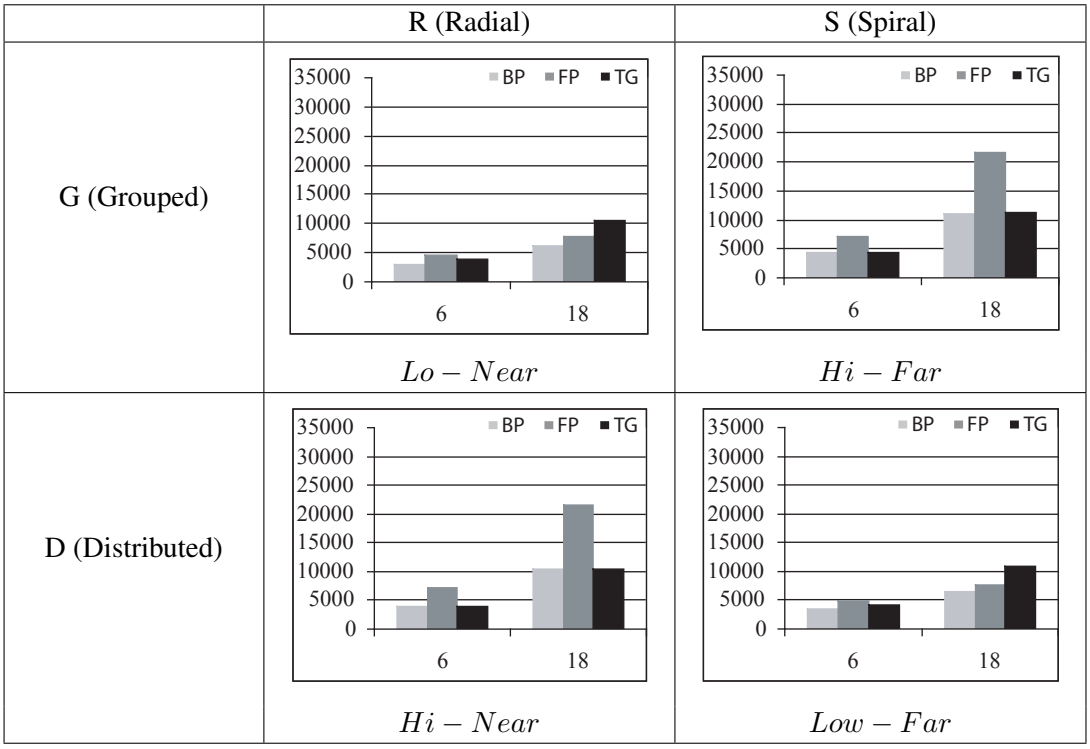


FIG. 4.3 : Les prédictions de SimCIS sur les tâches pour lesquelles l'ordre d'exécution est imposé

changements d'outil est faible.

- (H_2) BP est toujours plus rapide ou aussi rapide que les autres techniques car elle combine les propriétés de persistance et de parallélisme.
- (H_3) Toutes les techniques sont sensibles aux distances entre objets à cause de l'effet de la loi de Fitts (plus la cible est loin, plus il faut de temps pour l'atteindre).
- (H_4) La longueur de la séquence exacerbe les différences entre techniques à cause de l'effet de planification.

Dans la condition libre (F), les participants peuvent adopter l'ordre de leur choix pour accomplir la tâche. Le fait que les utilisateurs optimisent leur temps en fonction du contexte d'une part, et de la technique d'autre part est une hypothèse sous-jacente de CIS. L'analyse des observations dans la condition libre va nous permettre de tester cette hypothèse :

- (H_{Free}) Les utilisateurs optimisent leur temps. En d'autres termes, étant donnée une technique d'interaction T, ils planifient leur séquence d'interaction afin de tirer parti des avantages de T.

Participants

Douze adultes volontaires, âgés de 20 à 56 ans (29,41 en moyenne), 10 hommes et 2 femmes, tous droitiers, ont accepté de participer à l'expérimentation pour une durée de 45 minutes environ.

Matériel

L'expérimentation tournait sur un PC HP XW4000 sous Windows XP Professional. Le PC était équipé de deux tablettes WACOM avec un puck chacune pour permettre l'entrée bimanuelle. Les dimensions des tablettes étaient de 145×125 mm pour celle de droite et de 456×361 mm pour celle de gauche. L'expérimentation a été implémentée en Java en utilisant la boîte à outils ICON [59].

Procédure et stratégie de contrebalancement

L'expérimentation consiste en 36 conditions groupées en trois blocs, un par technique, pour éviter de perturber le sujet avec de multiples changements de technique. L'ordre de présentation des techniques est contrebalancé entre les sujets en utilisant un carré latin 3×3 (nous assignons quatre participants à chacun des trois ordres obtenus). Chaque bloc comporte trois sous-blocs : un bloc d'entraînement pour découvrir la technique, un sous-bloc de tâches libre (F) contenant les séquences pour lesquelles l'ordre n'est pas imposé et un sous-bloc contenant les séquences d'interaction imposées ($ordre$ =R ou $ordre$ =S). Afin de contrebalancer les valeurs du facteur $longueur$, chacun de ces sous-bloc est divisé en deux parties, une pour les séquences de $longueur = 6$ et une pour les séquences de $longueur = 18$. Parmi les quatre participants assignés au même ordre de présentation des techniques, deux commencent les sous-blocs par la partie $longueur = 6$ et deux par la partie $longueur = 18$. Le sous-bloc d'entraînement est toujours le premier du bloc et l'ordre de présentation du sous-bloc libre (F) et du sous-bloc imposé ($ordre$ =R ou $ordre$ =S) au sein d'un bloc est aussi contrebalancé : parmi les deux participants assignés à un ordre de techniques et un ordre de longueur, un participant commence par le sous-bloc libre tandis que l'autre participant commence par le sous-bloc imposé. L'ordre de présentation des 18 tâches, $\{G, D\} \times \{R, S\}$ répétées trois fois, au sein d'une partie est aléatoire.

Afin que le sous-bloc d'entraînement ne fausse pas notre stratégie de contrebalancement en familiarisant le sujet avec un type de tâche, nous demandons au sujet d'utiliser la technique différemment. Il doit effacer des formes apparaissant une par une dans la fenêtre principale et peut appuyer sur un bouton une fois qu'il s'estime familier avec la technique. La prochaine forme à effacer est toujours affichée en haut à droite afin que les sujets puissent anticiper leur prochaine interaction (à la manière du jeu populaire Tetris).

A la fin de l'expérimentation, les participants doivent répondre à un questionnaire afin de savoir les connaissances antérieures qu'ils ont de chaque technique, la technique qu'ils préfèrent et les cas pour lesquels ils la préfèrent. Le questionnaire comporte également une section présentant les 4 séquences libres (F) (Figure 4.4) et demande aux participants de classer les techniques dans leur ordre de préférence pour exécuter chacune de ces séquences.

4.1.2 Validation du modèle : Les résultats

Les mesures de l'expérimentation sont enregistrées à deux niveaux : le temps, le nombre d'erreurs et de changements d'outils entre le premier clic d'une séquence et l'effacement de la dernière forme de cette même séquence et le temps entre l'effacement de deux formes successives. Les observations pour $longueur = 6$ et $longueur = 18$ sont analysées séparément et les comparaisons entre paires sont

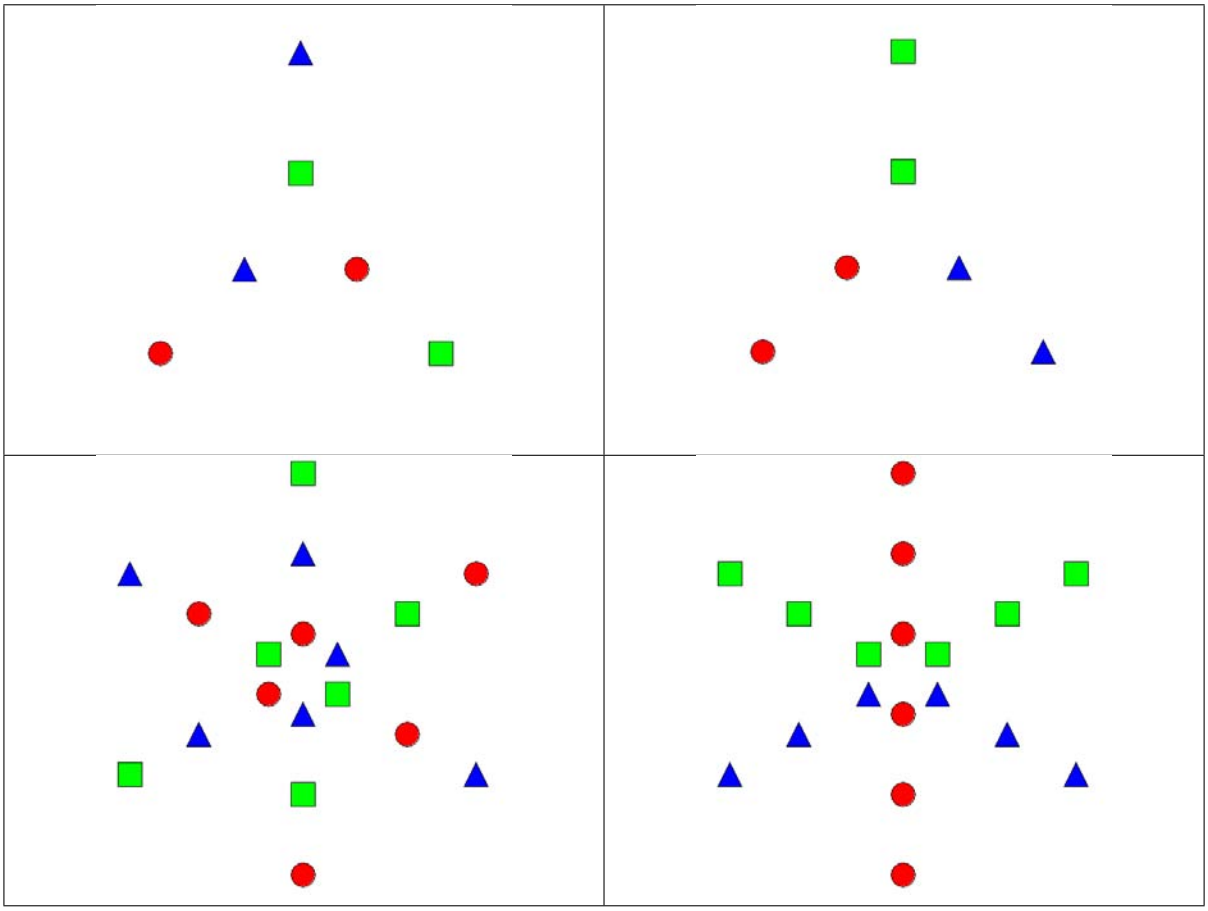


FIG. 4.4 : Les séquences libres

faites en utilisant le test HSD de Tukey. Dans un premier temps, nous comparons les observations et les prédictions de CIS afin de juger la qualité de ces dernières puis, dans un second temps, nous nous intéressons à la stratégie adoptée par les participants lorsque qu'ils sont libres quant à la planification de leurs interactions.

Comparaisons entre les observations empiriques et les prédictions de CIS

Dans cette section, nous focalisons nos analyses sur les séquences pour lesquelles l'ordre est imposé ($ordre =$ R ou $ordre =$ S). Nous commençons par comparer les résultats empiriques de la figure 4.5 aux prédictions que nous avons reportées dans la table de la figure 4.3. Bien que les prédictions de SimCIS sous-estiment le temps d'exécution de chaque tâche, les schémas comparatifs, c'est-à-dire les relations entre contexte d'utilisation et performance d'une technique et les comparaisons entre techniques, sont identiques entre prédictions et observations empiriques.

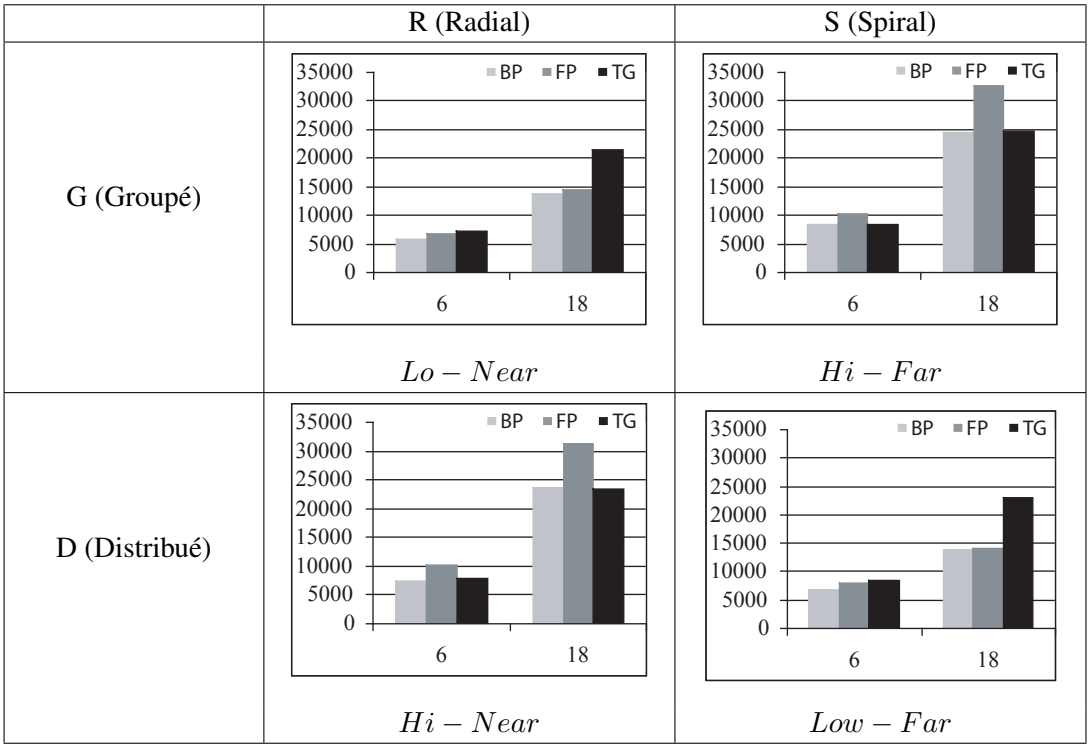


FIG. 4.5 : Les observations empiriques sur les tâches pour lesquelles l'ordre d'exécution est imposé

L'analyse de variance révèle un effet significatif de la technique sur le temps d'exécution ($longueur = 6 : F_{2,33} = 8,67$ et $p = 0,0009$; $longueur = 18 : F_{2,33} = 13,82$ et $p < 0,0001$). Les comparaisons par paire montrent que seulement les deux paires (BP, FP) et (BP, TG) sont significativement différentes. Nous obtenons donc l'ordre $BP < TG \simeq FP$ pour le temps d'exécution, ce qui est en accord avec l'hypothèse (H_2) : BP est toujours plus rapide ou aussi rapide que les deux autres techniques. La table de la figure 4.6 synthétise les temps d'exécution moyens pour chacune des trois techniques. Il est important d'analyser l'effet de la technique sur le nombre d'erreurs pour voir si les erreurs peuvent expliquer les différences entre techniques. en effet, SimCIS fait l'hypothèse que l'utilisateur interagit sans faire d'erreur. L'analyse de variance ne rapporte aucun effet significatif de la technique sur le nombre d'erreurs : ($longueur = 6 : F_{2,33} = 1,42$ et $p = 0,25$; $longueur = 18 : F_{2,33} = 0,58$ et $p = 0,556$).

	TG	BP	FP
$longueur = 6$	7972	**7134**	8812
$longueur = 18$	23142	**18903**	23100

FIG. 4.6 : Temps d'exécution moyen d'une séquence (en ms)

Les données collectées montrent un effet d'interaction entre $ordre$ et $groupement$ sur le temps d'exécution pour les deux types de palette ($longueur = 6$: $F_{2,132} = 9,63$ et $p = 0,0001$; $longueur = 18$: $F_{2,132} = 105,13$ et $p < 0,0001$). Ce résultat soutient l'hypothèse (H_1) : les palettes sont sensibles au nombre de changements d'outil et sont plus rapides dans les cas où les changements d'outil sont moins fréquents (Lo, c.à.d. S $\times$ D et R $\times$ G vs. Hi, c.à.d. S $\times$ G et R $\times$ D). Les résultats empiriques (figure 4.5) sont en accord avec les prédictions de SimCIS (figure 4.3) sur l'influence du contexte sur la performance des palettes : les différences entre les moyennes minimales et maximales du temps d'exécution d'une séquence sont significativement plus grandes pour FP que pour BP ($longueur = 6$: $diff_{FP} = 3348$ et $diff_{BP} = 2728$; $longueur = 18$: $diff_{FP} = 18465$ et $diff_{BP} = 10849$)

Les histogrammes de la figure 4.5 montrent que les temps moyens d'exécution d'une séquence sont plus courts dans les conditions $Near$ (les objets successifs sont proches les uns des autres) que dans les conditions Far (les objets successifs sont plus loin les uns des autres). Cependant, l'analyse de variance ne montre pas de différences significatives ($longueur = 6$: $F_{2,66} = 0,15$ et $p = 0,85$; $longueur = 18$: $F_{2,66} = 0.22$ et $p < 0.8$). Ceci est probablement dû à la disposition en étoile choisie pour les formes graphiques : dans les conditions $Near$, les distances pour passer d'une branche à l'autre sont assez grandes alors que, dans les conditions Far, les distances sont de plus en plus petites au fur et à mesure que l'on se rapproche du centre.

	FP		BP		TG	
	R	S	R	S	R	S
G	*2,12*	**3,20**	*2,40*	**2,91**	2,98	2,96
D	**3,08**	*1,74*	**3,13**	*2,02*	2,96	2,94

FIG. 4.7 : Rapport du temps d'exécution pour les séquences longues ($longueur = 18$) sur le temps d'exécution pour les séquences courtes ($longueur = 6$)

La table de la figure 4.7 liste les rapports entre les temps moyens d'exécution pour une séquence de $longueur = 18$ et celui pour une séquence de $longueur = 6$ pour chaque condition. Nous avons vu que les palettes sont sensibles au nombre de changements d'outil, la table de la figure 4.7 montre que, comme nous l'avions prédit avec H_3 ces sensibilités sont exacerbées par la longueur de la séquence. Par exemple, pour FP et BP, une séquence de $longueur = 18$ est environ 3 fois plus longue qu'une séquence de $longueur = 6$ lorsque les séquences maximisent le nombre de changements d'outil (Hi, en **gras** dans la figure 4.7) alors que le rapport est d'environ 2 lorsque les séquences minimisent le nombre de changements d'outil (Lo, en *italique* dans la figure 4.7). Notons aussi que dans le cas de TG, insensible au contexte, le rapport est d'environ 3 quelle que soit la séquence.

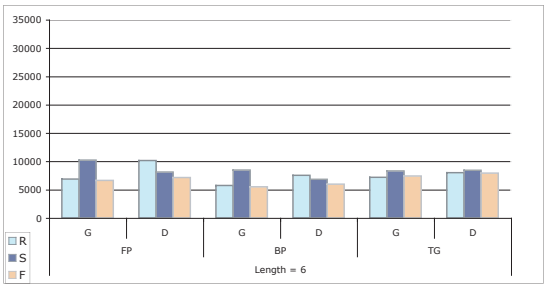
En résumé, les prédictions de SimCIS sont en accord avec les résultats empiriques que nous avons collecté et accréditent le fait que le concept absolu de "meilleure technique" n'existe pas. Montrer les avantages d'une nouvelle technique est légitime mais il est très important de les relativiser au contexte d'utilisation en envisageant différents scénarios afin de rapporter des résultats plus généralisables.

Les utilisateurs optimisent leur temps

Dans cette section, nous concentrons nos analyses sur les séquences pour lesquelles l'ordre n'est pas imposé ($ordre$ =F) afin de voir si les utilisateurs savent tirer profit d'une technique pour optimiser leur temps (H_{Free}).

Les combinaisons des valeurs des conditions d'$ordre$ imposé (R et S) et de $groupement$ (G et D) ont été choisies afin que, pour chaque technique, il y ait un type de séquence qui soit proche du contexte le plus favorable d'utilisation. Les séquences qui minimisent les changements d'outil sont favorables aux techniques persistantes (FP et BP) alors que les séquences qui minimisent les distances sont favorables aux techniques parallèles (BP et TG). L'analyse des résultats pour les séquences dans lesquelles l'ordre est libre (F) est résumée par la table de la figure 4.8. Elle montre que les utilisateurs se rapprochent de ces optima et sont même parfois en-deçà. Afin de s'assurer que ces résultats ne sont pas dus à un effet d'apprentissage lorsque les séquences pour lesquelles l'ordre est imposé sont présentées avant les séquences libres, nous nous sommes assurés qu'il n'y avait pas d'effet significatif de l'ordre de présentation sur le temps d'exécution : ($longueur = 6$: $F_{1,22} = 3,82$ et $p = 0,0633$; $longueur = 18$: $F_{1,22} = 0,28$ et $p < 0,6013$). La charge cognitive ajoutée par le fait d'avoir à suivre le chemin de la ligne noire pour effacer les formes peut expliquer pourquoi les utilisateurs "battent" les optima lorsqu'ils sont libres.

	$longueur = 6$						$longueur = 18$					
	FP		BP		TG		FP		BP		TG	
	G	D	G	D	G	D	G	D	G	D	G	D
R	6846	10116	5722	7529	7187	7986	14542	31181	13711	23539	21385	23476
S	10194	8093	8450	6834	8312	8403	32570	14105	24560	13802	24632	23075
F	**6633**	**7128**	**5479**	**5946**	**7389**	**7907**	**14379**	**15796**	**13754**	**16224**	**21710**	**24095**



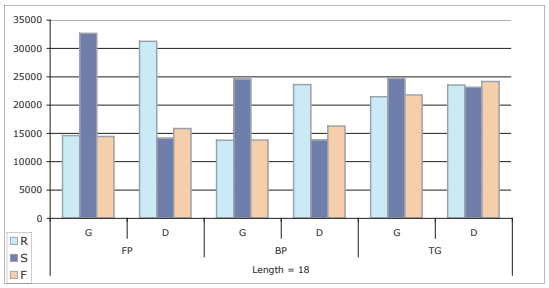


FIG. 4.8 : Séquences libres vs. séquences imposées : temps d'exécution moyen pour chacune des 36 conditions

L'analyse de variance révèle que l'effet de la technique sur le nombre de changements d'outil n'est pas significatif pour les séquences de $longueur = 6$ ($F_{2,33} = 2,67$ et $p = 0,0836$) mais l'est pour les séquences de $longueur = 18$ ($F_{2,33} = 64,56$ et $p = 0,0001$, les paires (TG, BP) et (TG, FP) étant significativement différentes). Les utilisateurs tendent donc à minimiser le nombre de changements d'outil (et donc à se rapprocher de la séquence optimale) avec une palette mais pas avec une toolglass

(figure 4.9). Les utilisateurs semblent donc “comprendre” que l’influence du contexte affecte la technique et cherchent à se mettre dans un contexte optimal en fonction de la technique utilisée comme nous l’avions prédit en H_{Free}.

	FP	BP	TG
$longueur = 6$	2,80	3,12	3,23
$longueur = 18$	2,00	3,11	8,40

FIG. 4.9 : Nombre moyen de changements d’outil par technique pour les séquences libre ($ordre$ =F)

Cependant, dans le questionnaire final, seulement trois participants ont été capables de décrire pour quelles séquences une technique donnée serait plus efficace. Un participant a préféré TG quelle que soit la séquence, trois BP et cinq FP, les derniers justifiant leur choix par leur familiarité avec la palette fixe par rapport aux deux autres techniques. Rappelons que la dernière question présentait les séquences imprimées et demandait aux participants de noter chacune des techniques pour ces séquences. Il est surprenant de constater que les réponses à cette question sont consistantes avec les données empiriques bien que (i) les participants sont dans l’ensemble incapables de répondre à la question leur demandant de caractériser les contextes dans lesquels une technique donnée est plus efficace et que (ii) les préférences globales ne reflètent pas toujours celles collectées grâce à cette dernière question. En effet, dans cette dernière question, BP obtient toujours la meilleure note et FP a une meilleure note que TG pour les séquences $groupement$ = G alors que le résultat s’inverse pour les séquences $groupement$ =D. Il semble donc que les utilisateurs ne “comprennent” pas vraiment la nature de la relation entre performance d’une technique mais qu’ils savent cependant tirer parti des avantages d’une technique en pratique.

Tous ces résultats montrent que les utilisateurs savent optimiser leur utilisation d’une technique et adaptent leur façon d’interagir en fonction du contexte d’utilisation. De plus, bien que les participants n’aient pas été capables d’identifier clairement les propriétés des techniques d’interaction, ils ont su identifier la technique la plus efficace pour une tâche donnée.

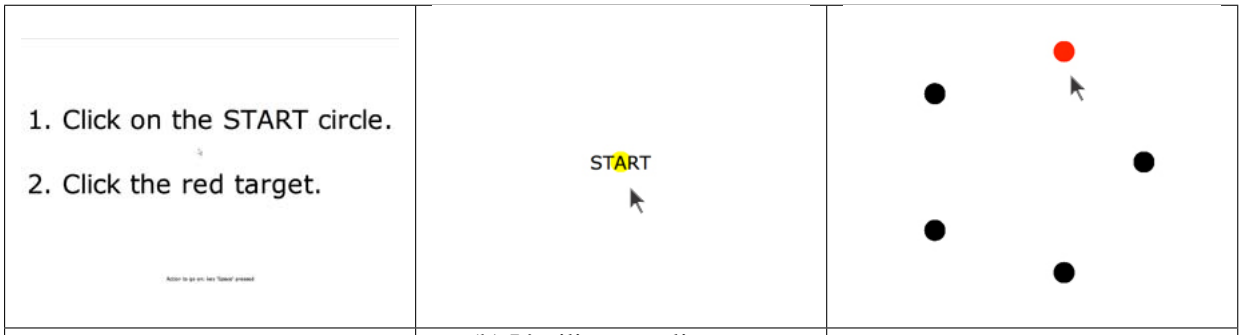
4.2 Raffiner CIS

Les résultats de cette expérimentation ont révélé que les prédictions de SimCIS étaient bien en deçà des observations empiriques que nous avons collectées. Nous identifions ici deux causes de ces différences : d’une part, nous avions fixé a priori les constantes des lois empiriques sur lesquelles repose CIS et, d’autre part, nous ne disposons pas de modèle pour le pointage bimanuel.

Rappelons que CIS utilise les lois de Hick-Hyman et de Fitts pour prédire les temps de choix et de pointage, ces deux lois comportant chacune deux constantes à déterminer empiriquement (voir la section 3.1.1). Pour les prédictions que nous avions faites avant l’expérimentation, nous avions utilisé les valeurs reportées dans la littérature : $a = 0$ et $b = 100$ pour la loi de Fitts et $a = 0$ et $b = 150$ pour la loi de Hick-Hyman. Les différences observées entre les prédictions et les observations peuvent être dues à

l'utilisation de constantes qui n'ont pas été déterminées dans les mêmes conditions expérimentales. Nous avons mis au point deux petits protocoles expérimentaux d'environ 3 minutes chacun afin de déterminer a posteriori les valeurs des constantes dans nos conditions expérimentales. Le premier programme implémente la tâche de pointage réciproque de Fitts [64] qui a été fréquemment utilisée dans la littérature pour évaluer des techniques de pointage [118, 33, 15] afin de déterminer les coefficients de la loi de Fitts. Cette tâche consiste à pointer des cibles successives apparaissant une par une. Ce programme présente 8 blocs de 10 pointages chacun, 1 bloc par indice de difficulté (1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5, 5.5).

Le second programme sert à déterminer les constantes de la loi de Hick-Hyman. À notre connaissance, il n'existe pas de tâche expérimentale typique pour opérationaliser une tâche de choix. Nous nous sommes donc inspirés des protocoles expérimentaux originaux de Hick [88] et Hyman [94]. Dans celui de Hick, les participants se trouvent face à un ensemble de lampes initialement éteintes disposées circulairement devant eux. Chaque lampe correspond à une touche Morse. Chaque fois que l'expérimentateur allume une des lampes, le participant doit presser la touche Morse correspondante. Dans celui de Hyman, les lampes sont organisées en matrice et chaque lampe porte un nom simple (*Bee* ou *Boo* par exemple) que le participant doit prononcer lorsque la lampe correspondante s'allume. La figure 4.10 illustre la tâche que nous avons utilisée. L'utilisateur est initialement face à un écran ne présentant qu'un bouton START. Il doit presser le bouton START pour faire apparaître un ensemble de points équidistants du bouton START et déclencher le chronomètre. Parmi ces points, seule la cible est rouge alors que tous les autres distracteurs sont noirs. L'utilisateur doit alors aller cliquer la cible rouge. Aux temps collectés, nous soustrayons le temps pris par le pointage (grâce aux constantes que nous avons déterminées dans la première expérimentation de Fitts) afin d'isoler le temps de choix. Pour cette seconde expérience, le participant doit accomplir 8 blocs de 10 choix chacun, 1 bloc par nombre d'éléments dans l'ensemble (3, 5, 8, 11, 15, 19, 24 et 29).



(a) L'utilisateur lit les instructions. (b) L'utilisateur clique sur START pour faire apparaître les cibles. (c) L'utilisateur clique sur la cible rouge (a)

FIG. 4.10 : La tâche expérimentale pour déterminer les coefficients de la loi de Hick

À partir des données collectées, les programmes calculent les valeurs des coefficients par régression linéaire. En utilisant uniquement les données sur un seul utilisateur, l'expérimentateur, nous obtenons les

coefficients suivants : $a = 225$ et $b = 130$ pour la loi de Fitts et $a = -120$ et $b = 60$ pour la loi de Hick-Hyman. La figure 4.11 rapporte les prédictions obtenues en injectant ces nouvelles constantes. La comparaison des tables 4.3 et 4.5 puis des tables 4.11 et 4.5 montrent que les prédictions s'en trouvent nettement améliorées. Notons que le fait d'avoir une ordonnée à l'origine, a, négative pour la loi de Hick-Hyman n'est pas surprenant car le choix de zéro élément parmi un ensemble n'a pas de sens.

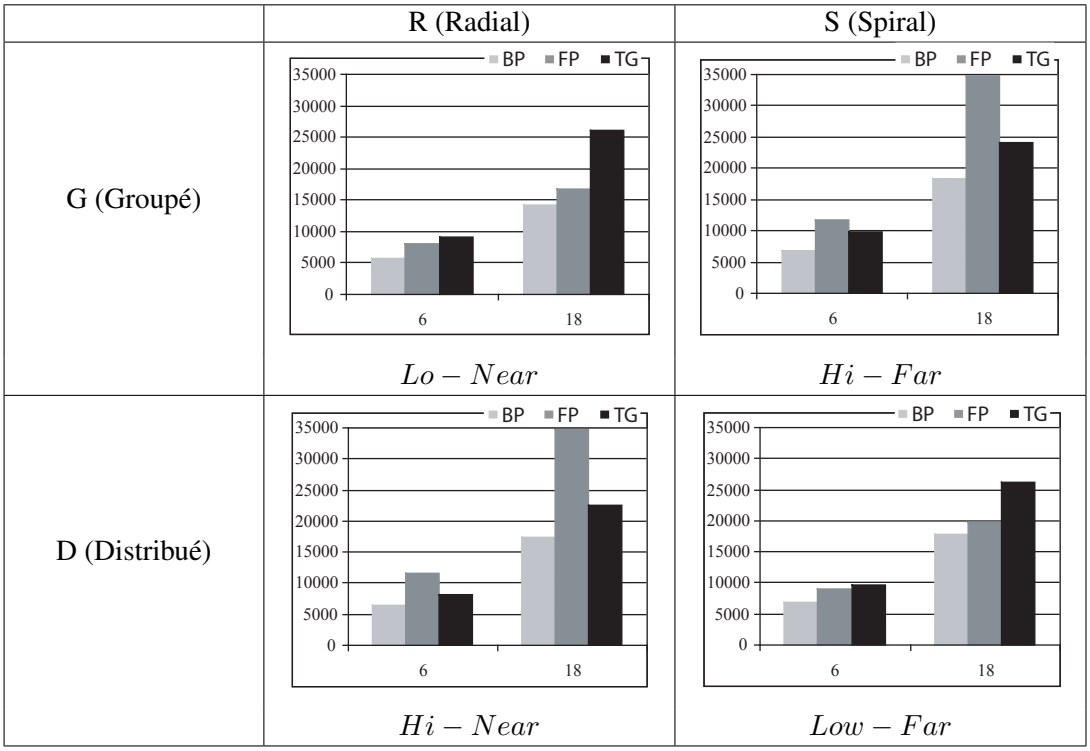


FIG. 4.11 : Les prédictions de SimCIS raffinées sur les tâches pour lesquelles l'ordre d'exécution est imposé

En comparant les tables des figures 4.11 et 4.5, on peut voir que les principales différences sont pour la toolglass pour toutes les séquences et la palette bimanuelle pour les séquences Hi (c.à.d. changements d'outil fréquents). Ces séquences sont celles qui font intervenir la plus grande proportion de pointages bimanuels, or nous ne disposons pas de modèle prédictif de pointage bimanuel. Inspirés par les recherches de Bourgeois [36] sur la navigation multi-échelles avec une interaction bimanuelle qui présentait un fort degré de parallélisme entre les contrôles des deux mains, nous avons évalué le coût d'un pointage bimanuel comme celui du pointage le plus difficile auquel on applique une pénalisation proportionnelle à l'indice de difficulté du pointage le plus simple. Considérons un pointage parallèle $P_{//}$ constitué des pointages P_{min} et P_{max} tels que $ID_{P_{min}} < ID_{P_{max}}$, SimCIS évalue la complexité en temps de $P_{//}$ selon la formule complexité($P_{//}$) = complexité(P_{max}) $+1,5\times$ complexité(P_{min}).

Au vu des données empiriques sur les séquences Hi, la palette bimanuelle et la toolglass semblent équivalentes (Figure 4.5) alors que SimCIS prédit une meilleure performance pour la palette bimanuelle (Figure 4.11). Notre modèle du double pointage n'est donc pas correct car il avantage la palette bimanuelle pour laquelle les pointages les plus simples (P_{min}) sont beaucoup plus faciles (simplement passer du dernier outil utilisé à celui que l'utilisateur veut maintenant sélectionner) que pour la toolglass (amener le bon outil depuis le dernier point effacé jusqu'au point que l'utilisateur veut maintenant effacer). Nous avons donc besoin d'un modèle plus précis pour le pointage bimanuel afin d'améliorer les mesures de complexité en temps de SimCIS. Cette étude n'a pas été abordée au cours de ce travail de thèse mais nous projetons d'étudier plus finement le pointage bimanuel en fonction des deux indices de difficulté des pointages parallèles. Alors que l'étude du pointage simple est extrêmement bien balisée par les précédentes recherches, celle du pointage bimanuel n'a pas été abordée et est plus délicate car elle fait intervenir des facteurs supplémentaires comme la place relative des deux cibles à atteindre. Par exemple, dans le cas de la toolglass, il s'agit d'une seule et même cible (l'attention visuelle n'est pas divisée) alors que, dans le cas de la palette bimanuelle, les deux cibles à atteindre peuvent être arbitrairement loin l'une de l'autre (l'utilisateur doit donc regarder successivement deux endroits).

4.3 Étendre CIS

Afin d'évaluer l'étendue de la couverture de CIS, nous répertorions ici les principales familles de techniques d'interaction pour les applications graphiques et les profils CIS correspondant. Bien évidemment toutes ces familles d'interaction ne sont pas exclusives et CIS permet de décrire et évaluer des techniques qui combinent des actions de différentes familles. Par exemple, une technique peut très bien consister en une sélection d'un outil par pointage puis en l'application de cet outil à un ou plusieurs objets par franchissement.

Ce qui est couvert L'interaction "point-and-click" est la plus ancienne des six familles d'interaction répertoriées par la figure 4.12. Elle est née avec l'interface du Xerox Star [161] qui consistait en un dispositif d'affichage présentant un ensemble de widgets avec lequel l'utilisateur interagissait grâce à une souris et un clavier. Ces techniques sont des séquences "choix d'une cible" - "pointage de la cible" - "clic" qui peuvent être décrites avec CIS comme "branchement" - "acquisitions" - "validations de temps constant" (figure 4.12-a). Les techniques basées sur le franchissement ont des graphes similaires mais les nœuds (validations) de ces graphes ont un coût nul. Ainsi, cette famille de techniques est décrite avec des graphes traduisant des séquences d'action "branchement" - "acquisitions" (figure 4.12-b) comme nous l'avons vu avec l'exemple du ControlTree. Selon le modèle de Accot et Zhai [2], suivre une trajectoire avec un dispositif de pointage peut être vu comme une somme de franchissements. Dans un graphe CIS, une trajectoire se traduit par un arc double qui est étiqueté par l'équation de la trajectoire que doit suivre l'objet déplacé (figure 4.12-c). Cette équation est nécessaire pour fournir les complexités en temps de ces techniques en se fondant sur le modèle de Accot et Zhai. Enfin, nous avons déjà vu avec les exemples de la toolglass ou de la palette bimanuelle que les interactions bimanuelles étaient traduites par des arcs parallèles dans une modélisation CIS (mais aussi que le modèle que nous utilisons actuellement pour ce type d'arcs n'est pas tout à fait satisfaisant) (figure 4.12-d).

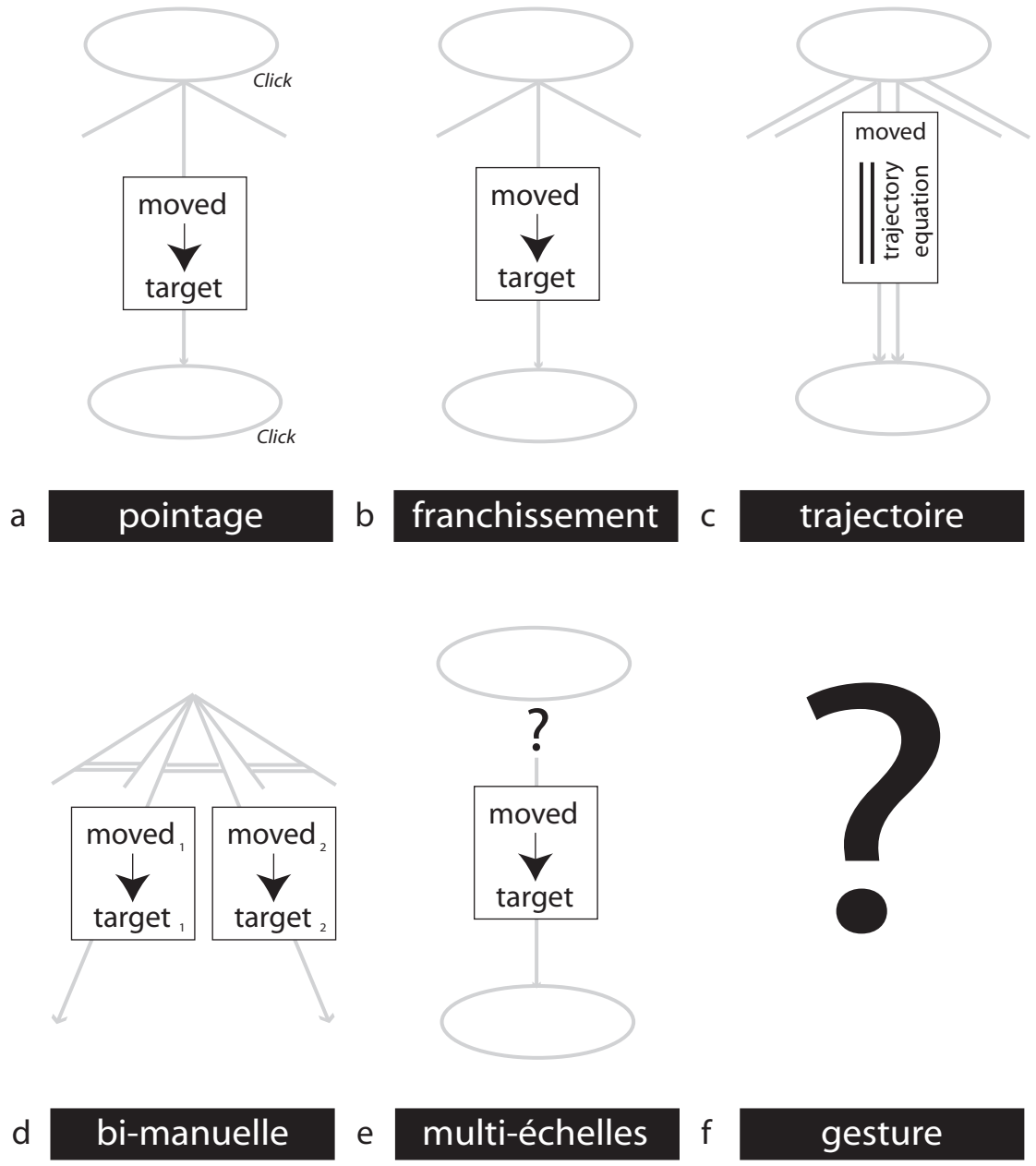


FIG. 4.12 : Les familles d'interaction et leurs profils CIS

Les extensions à faire La figure 4.12 résume les différentes familles d'interaction que nous avons identifié et les profils CIS correspondants. À l'heure actuelle, il reste deux grandes familles de techniques que nous n'avons pas encore complètement abordées avec CIS : l'interaction multi-échelle (figure 4.12-e) et l'interaction gestuelle (figure 4.12-f). Pour l'interaction multi-échelle, nous disposons du modèle d'acquisition puisque nous avons déjà mentionné que la loi de Fitts s'applique pour les pointages dans une interface zoomable. Ce qui fait défaut est un modèle de choix dans le contexte des interfaces multi-échelle. En effet, la loi de Hick-Hyman est valide lorsque les éléments de l'interface sont visibles or, dans une interface zoomable, la visibilité des objets dépend du facteur de zoom. Pour faire un choix parmi plusieurs objets de l'interface, l'utilisateur doit naviguer par pan and zoom pour aller explorer les éléments avant d'identifier celui qu'il cherche à atteindre. Il semble donc qu'un modèle de choix dans ce cas implique des actions motrices en plus de la perception visuelle modélisée par la loi de Hick-Hyman. La section suivante présente le cadre expérimental et l'étude préliminaire que nous avons conduit pour aider à l'élaboration d'un modèle de choix dans une interface zoomable. Enfin, les interactions gestuelles souffrent du même manque : alors que Cao et Zhai [44] ont récemment introduit un modèle des actions motrices mises en jeu pour réaliser un geste, nous n'avons pas de modèle de choix ou plutôt de sélection. En effet, alors que le modèle de Cao et Zhai permet d'évaluer un geste isolé à partir de sa description sous forme d'une somme de traits primitifs, il n'existe pas de modèle pour évaluer la sélection d'un geste parmi un vocabulaire. Est-ce que la taille du vocabulaire entre en jeu ? Est-ce que la composition du vocabulaire a un effet (si les gestes sont très différents ou, au contraire, assez proches) ? Etc. Cette question reste ouverte et n'a pas été abordée dans le cadre de ce travail de thèse mais elle ouvre des pistes de recherche que nous envisageons de poursuivre. Aussi, le modèle de Cao et Zhai n'est pas encore directement utilisable : par exemple, un geste doit être décrit comme une somme de traits primitifs (lignes droites, courbes et angles) mais la façon de décomposer un geste n'a pas été abordée.

4.4 Un modèle de choix pour l'interaction multi-échelles

Cette section présente l'étude récente que nous avons menée [141] afin de fournir les premiers éléments pour élaborer un modèle de choix pour l'interaction multi-échelle. Comme nous venons de l'évoquer, le choix d'un élément dans un ensemble est principalement composé d'actions motrices pour naviguer dans l'interface zoomable. Ainsi, la tâche de choix s'apparente plus à une *tâche de recherche* dans l'espace.

Ce que nous introduisons dans cette section est un cadre expérimental générique dans un esprit similaire à celui du pointage réciproque de Fitts mais adapté à l'étude de la recherche multi-échelle. Après une brève revue des expérimentations sur la recherche multi-échelle, nous introduisons notre tâche et le monde abstrait que nous utilisons pour évaluer quatre techniques de navigation multi-échelle dans une configuration particulière du monde. Nous présentons les résultats de cette évaluation préliminaire et présentons une application plus réaliste qui permet d'identifier les configurations à étudier pour élaborer un modèle de recherche multi-échelle plus général.

4.4.1 Travaux antérieurs

Plusieurs expérimentations ont comparé les performances de différentes techniques de navigation multi-échelle et ont rapporté des résultats contrastés. Le pan & zoom classique (pan avec un drag et

zoom avec la roulette de la souris) a été comparé avec la technique de pan & zoom augmenté d'une vue d'ensemble (*overview+detail*) et avec les lentilles *fisheye* (*focus+contexts*) pour des tâches cognitives de haut niveau sur des documents électroniques [92] : écrire un essai après avoir exploré un document ou trouver des réponses dans le document à des questions posées. Le pan & zoom classique était la technique la moins efficace ; les participants lisaient plus vite avec la technique *focus+context* ; ils écrivaient de meilleurs essais avec la technique *overview+detail* mais prenaient plus de temps pour répondre aux questions avec cette dernière technique. Dans l'expérimentation de North et Shneiderman [133], les participants devaient parcourir la base de données des états et comtés des États-Unis pour répondre à des questions en utilisant une *vue scrollable* ou une technique *overview+detail*. Leur expérimentation a rapporté que la technique *overview+detail* permettait aux participants d'être de 30 à 80 % plus rapide que la vue scrollable. Enfin, dans l'étude de Nekrasovski et al. [131], le pan & zoom classique ou la technique *overview+detail* n'ont pas montré de différences significatives lorsque les participants devaient naviguer dans de grandes représentations nœud-lien pour faire des comparaisons topologiques alors qu'Hornbaeck et al. [91] ont rapporté le résultat inverse dans le cas où les participants devaient explorer une des deux cartes géographiques utilisées dans leur expérimentation.

Les résultats de ces expérimentations montrent qu'il est difficile d'obtenir des résultats cohérents lorsque les tâches expérimentales sont dépendantes d'un domaine d'application, d'autant plus lorsque ces tâches sont des tâches cognitives de haut niveau (ce qui n'est pas étonnant puisque nous avons précédemment montré que la performance d'une technique dépend de son contexte d'utilisation). Identifier et isoler des tâches abstraites de plus bas niveau aiderait à obtenir des résultats plus généralisables. D'un point de vue moteur, une tâche récurrente des utilisateurs d'interface multi-échelle consiste à naviguer dans l'espace pour trouver des cibles parmi des ensembles d'objets. C'est cette tâche que nous nous proposons d'étudier ici.

4.4.2 Opérationalisation de la tâche de recherche multi-échelle

L'étude d'une tâche avec une expérimentation contrôlée passe par une opérationalisation de cette tâche, c'est-à-dire la définir par sa définition comme une fonction de variables d'intérêt (facteurs ou variables indépendantes) afin que les expérimentateurs puissent agir sur ces variables afin de collecter des mesures. La tâche de pointage [64] est un exemple bien connu : pour étudier la performance des techniques de pointage, les expérimentateurs agissent sur la variable *indice de difficulté* (ID) et mesurent le temps de mouvement. Lors d'une recherche dans un espace multi-échelle, les utilisateurs naviguent vers les positions qui révèlent assez de détails sur chaque objet. Initialement, lorsqu'ils n'ont pas assez d'information, les utilisateurs partent d'un facteur de zoom assez petit et font un choix "en aveugle" d'un objet à aller explorer. S'il s'agit d'un distracteur alors ils doivent aller naviguer vers la prochaine cible potentielle (typiquement par une phase de zoom-out, suivie d'une phase de pan et enfin d'une phase de zoom-in [77]) et ainsi de suite jusqu'à ce qu'ils trouvent la cible.

Puisque nous nous intéressons à étudier la performance d'une technique en fonction d'une *quantité d'exploration* d'un point de vue strictement moteur, nous définissons une représentation abstraite dans laquelle les objets n'ont ni relation sémantique ni relation topologique entre eux car ces relations pourraient suggérer un certain ordre de parcours aux participants pour trouver la cible (par exemple, localiser Moscou à partir de Saint-Pétersbourg est une recherche informée pour toute personne connaissant la

géographie de la Russie). Afin de quantifier l'exploration, notre monde abstrait contient un ensemble de n objets, l'un d'entre eux est la cible, les autres sont des distracteurs. La *quantité d'exploration* est alors le nombre k de distracteurs que les utilisateurs ont à explorer avant de trouver la cible. D'un point de vue probabiliste, k est dépendent de n : plus il y a d'objets, plus la probabilité d'explorer un grand nombre d'objets avant de trouver le bon est grande. Afin de contrôler ce paramètre, nous forçons les utilisateurs à explorer un nombre prédéfini d'objets avant de trouver la cible. Fixer a priori quel objet est la cible introduit un facteur aléatoire incontrôlé : l'utilisateur peut par chance tomber dessus tout de suite sur la cible ou par malchance la découvrir en dernier. C'est pourquoi, notre cadre expérimental assure que la cible est le k^{eme} objet visité quel que soit l'ordre d'exploration choisi par le participant. À cette fin, le système doit donc pouvoir (i) déterminer quels sont les objets vus par le participant, et (ii) à quel moment le participant a collecté assez d'information pour faire la discrimination cible/distracteur. Fournir ces informations au système nécessite de répondre aux deux questions suivantes :

- Quelle est l'échelle minimale qui fournit assez d'information ? La réponse dépend de l'acuité visuelle qui varie d'un utilisateur à l'autre.
- Dans quelle région de la vue et pour combien de temps un objet doit-il être affiché afin d'être considéré comme exploré ? Supposer que l'utilisateur est capable de balayer tout l'écran à tout instant est bien évidemment une hypothèse trop forte et infondée. Aussi, s'il n'y a qu'une partie de l'objet qui est visible, le système ne peut être sûr que l'utilisateur l'a vu et a collecté assez d'informations.

Pour répondre à ces questions, nous avons introduit une action explicite des participants. Tous les objets sont visuellement identiques à tous les niveaux d'échelle et l'utilisateur peut demander, pour chaque objet, quelle est sa nature (cible ou distracteur) dès que le facteur d'échelle *minScale* est atteint. Nous avons donc, tout d'abord, fixé un facteur d'échelle minimal (*minScale*) à partir duquel l'utilisateur avait le droit de révéler la nature de l'objet par une action explicite de dévoilement. Ensuite, nous nous sommes assurés que le participant ne collectait l'information sur un seul objet à la fois. Une fois qu'un objet a été dévoilé, l'utilisateur doit traiter l'information qui désambiguïse la nature de l'objet et fournir une action explicite supplémentaire dans le cas où cet objet est la cible.

Bien que nous ne puissions être sûrs que les participants regardent effectivement les objets lorsqu'ils les dévoilent, ceci est dans leur intérêt puisqu'ils ont la consigne de trouver la cible le plus rapidement possible. Cette conception nous semble donc opérationaliser correctement une tâche de recherche réaliste sans avoir recours à des périphériques plus complexes comme les systèmes de suivi du regard (*eye trackers*). Avant de présenter l'instanciation de ce cadre expérimental que nous avons utilisée pour la tâche de notre évaluation, nous présentons les quatre techniques de navigation multi-échelle que nous avons comparé.

4.4.3 Les techniques d'interaction multi-échelles évaluées

Un grand nombre de techniques de navigation multi-échelle existent, certaines étant des variations d'autres techniques. Dans cette expérimentation, nous avons choisi quatre techniques qui constituent un échantillon représentatif des techniques les plus efficaces et les plus couramment utilisées. La figure 4.13 représente ces techniques en utilisant des diagrammes espace-échelle [67].

La première technique est le *Pan & Zoom* classique (Figure 4.13-a). Pour obtenir plus de détails sur la représentation, l'utilisateur doit changer l'échelle (zoom) et déplacer (pan) la totalité de la vue.

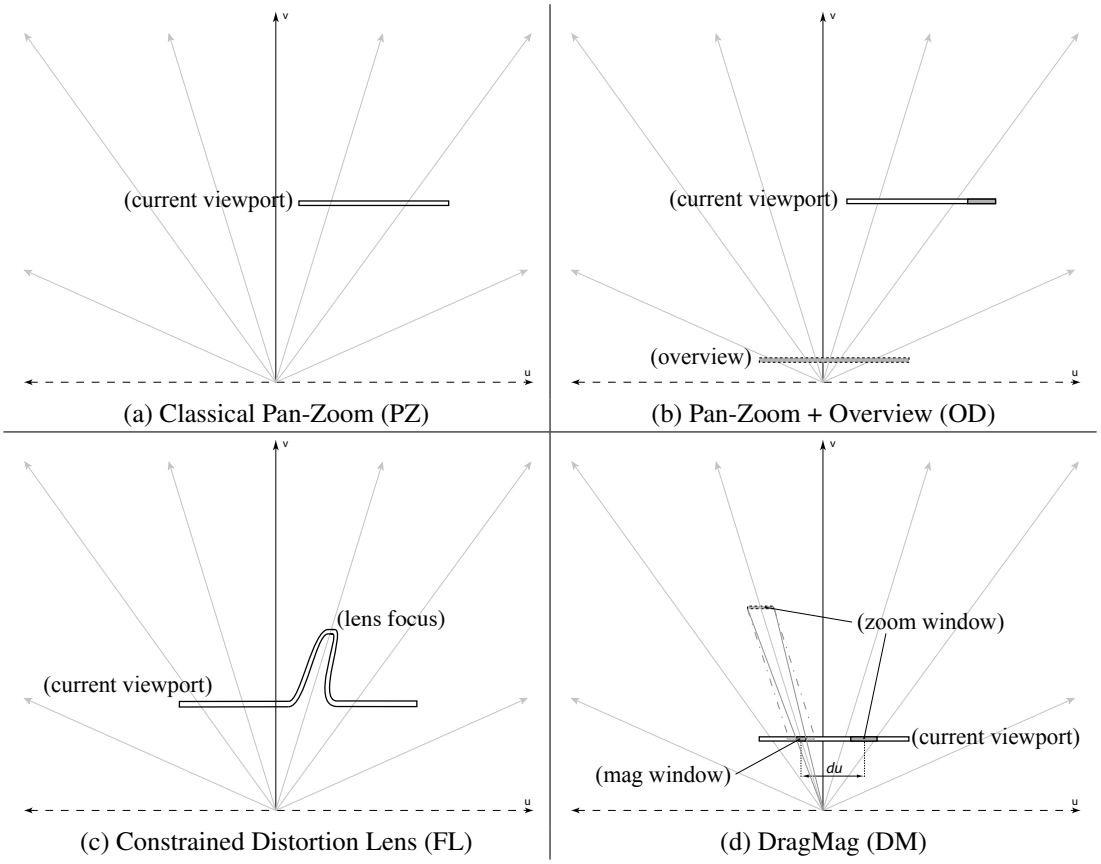


FIG. 4.13 : Représentation des techniques d'interaction multi-échelle dans des diagrammes espace-échelle. L'aire grisée représente la position de la seconde fenêtre de vue dans la vue courante.

Cette technique ne fournit aucune information de contexte et peut donc entraîner une désorientation de l'utilisateur.

La seconde technique, *Pan & Zoom + Overview*, résout ce problème en augmentant le Pan & Zoom classique par une vue secondaire (*vue d'ensemble* ou *overview*) qui affiche la représentation à un facteur d'échelle supérieur afin de fournir le contexte de la vue principale. La vue d'ensemble est affichée à l'intérieur de la vue principale, typiquement à l'un des quatre coins. Cet affichage minimise l'occlusion avec la région centrale qui constitue bien souvent la région d'intérêt mais introduit le problème de l'attention divisée (*divided attention*) [146].

Les troisième et quatrième techniques sont des techniques couramment qualifiées de *focus+context*. Contrairement aux techniques *overview+detail* comme le Pan & Zoom + Overview, elles accordent plus d'espace d'affichage au contexte qu'à la vue détaillée et la position de la représentation détaillée est ajustée en fonction des interactions de l'utilisateur afin d'être proche de la région d'intérêt. La troisième technique, la *lentille de distortion* (ou *lentille fisheye*) [50], déforme la représentation autour du curseur afin d'offrir plus de détails. Le grossissement *in situ* de la région d'intérêt résout le problème de l'attention divisée mais la distorsion peut poser des problèmes de lisibilité. Enfin, la quatrième technique, le *DragMag* [172] ne distord pas la vue *focus* et place la vue détaillée à côté du curseur et non pas sous le

curseur.

4.4.4 Expérimentation

Nous avons conduit une expérimentation selon un plan 4x9 intra-participant pour comparer ces quatre techniques sur une instanciation de la tâche de recherche multi-échelle que nous avons introduite.

Tâche

La tâche consiste à trouver une cible parmi un ensemble d'objets le plus rapidement possible en minimisant le nombre d'erreurs. La scène contient initialement neuf carrés gris clair organisés spatialement selon une grille 3x3. Cet agencement régulier permet d'une part aux participants de savoir où sont les objets à inspecter et d'autre part de s'assurer que les distances parcourues pour des essais de même rang k sont du même ordre de grandeur. Une grille rouge foncé est dessinée dans le fond de la scène pour éviter le problème de *desert fog* [100] et pour renforcer le feed-back aux opérations de zoom de l'utilisateur. Tous les objets ont des coins carrés à l'exception de la cible qui a des coins arrondis. Les coins arrondis ne peuvent être vus qu'à partir du facteur d'échelle *minScale* (Figure 4.14). Les participants doivent zoomer pour voir si l'objet est la cible ou non : une fois *minScale* atteint, la bordure de l'objet est dessinée en noir, indiquant que l'objet peut être dévoilé en appuyant sur la touche espace. L'agencement spatial assure qu'un seul objet peut être révélé à la fois. La taille de la vue est telle que deux objets ne peuvent être simultanément dans la vue lorsque l'échelle est supérieure à *minScale*.

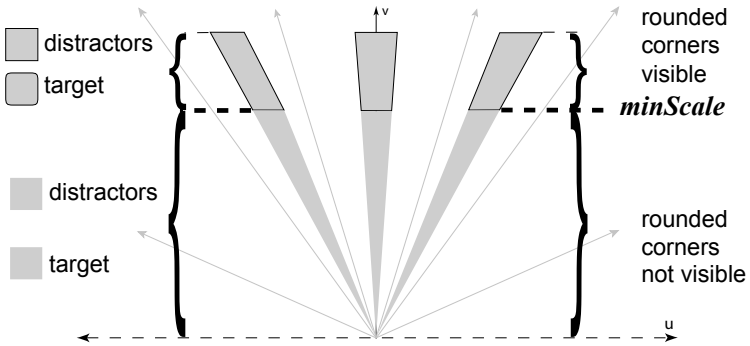
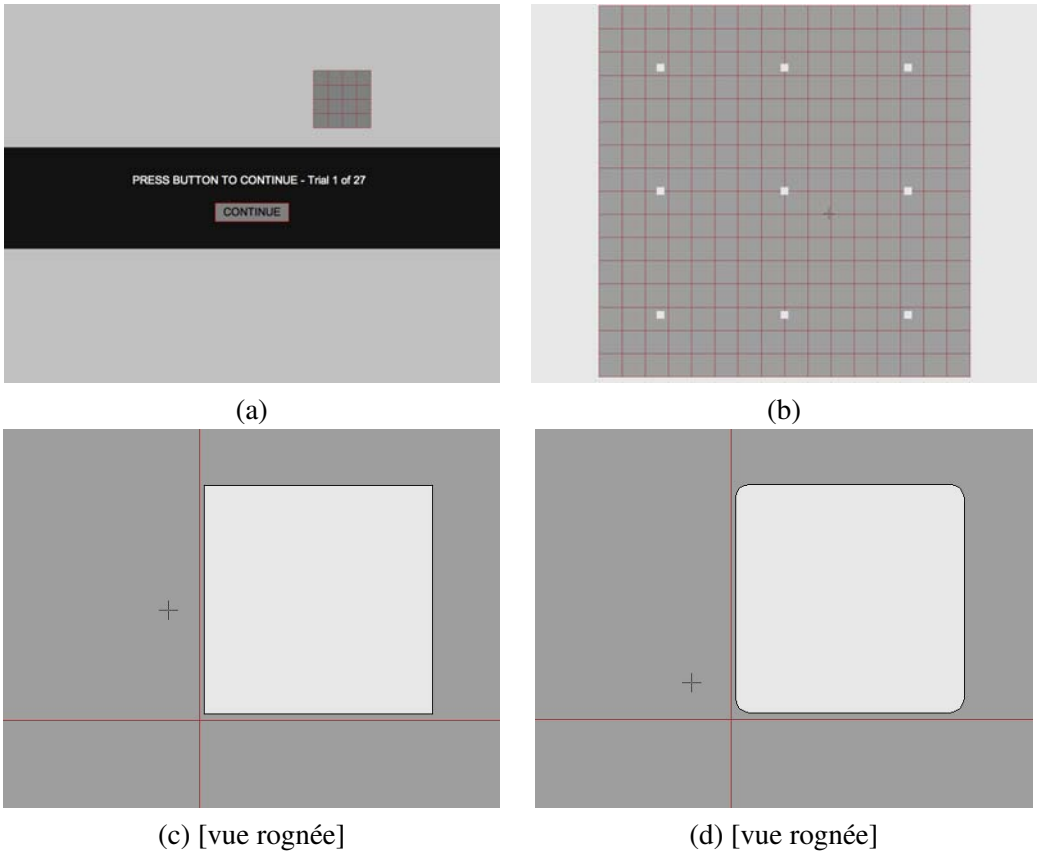


FIG. 4.14 : Diagramme espace-échelle de la scène utilisée lors de l'expérimentation

La figure 4.15 montre un storyboard de la tâche. Les participants commencent chaque essai en cliquant un bouton localisé au centre de l'écran (Figure 4.15-a). La vue est initialisée de manière à ce que la région qui contient tous les objets ne soit pas centrée dans la vue : le participant doit naviguer par Pan & Zoom pour l'atteindre (Figure 4.15-b) afin d'éviter un effet d'apprentissage sur les mouvements initiaux de la tâche et mieux simuler une tâche de recherche. Ensuite le participant doit inspecter chacun des objets de plus près en utilisant la technique courante (Figure 4.15-c). Le participant est autorisé à zoomer encore plus mais zoomer trop conduit à une situation dans laquelle l'objet est plus grand que la vue, ce qui rend impossible son identification. Une fois *minScale* atteint pour un objet, le participant

peut dévoiler cet objet en appuyant sur la barre d'espace. La bordure de l'objet est alors verte pendant 400 ms pour informer le participant que l'objet est bien dévoilé. Si les coins de l'objet restent carrés alors l'utilisateur part inspecter un autre objet ; s'ils sont devenus arrondis (Figure 4.15-d), le participant tape sur la touche `F1` pour indiquer qu'il a trouvé la cible et ainsi finir l'essai courant. S'il appuie sur la touche `F1` dans le cas d'un distracteur, l'essai se termine et est enregistré comme une erreur. Alors que nous disons aux participants que la cible est choisie aléatoirement parmi l'ensemble des 9 objets, ce n'est pas le cas en réalité : le programme compte les objets visités grâce aux actions explicites de dévoilement et fait en sorte que la cible soit le k^{eme} objet dévoilé par le participant. Ce comportement nous permet de contrôler complètement le facteur k.



(a) (b)

(c) [vue rognée] (d) [vue rognée]

FIG. 4.15 : Storyboard : (a) début d'un essai, (b) navigation vers l'ensemble d'objets, (c) inspection d'un objet (avant dévoilement), (d) après avoir dévoilé la cible (c et d sont rognées afin d'illustrer la taille réelle des objets sur l'écran de l'expérimentation)

Implémentation des techniques

La première variable indépendante de l'expérimentation est la technique de navigation. La première technique est le *pan & zoom* classique (PZ). Les participants peuvent déplacer la vue en la faisant glisser par une interaction de drag et zoomer en utilisant la roulette de la souris. Le facteur de grossissement utilisé à chaque mouvement de la roulette est réglé pour que l'utilisateur ait une vitesse moyenne de zoom

de 8x par seconde [91]. Les trois techniques suivantes sont des augmentations de celle-ci et disposent donc des mêmes opérations basiques de pan & zoom.

La seconde technique est le *Pan & Zoom + Overview* (OD) et présente une vue d'ensemble en bas à droite de la vue principale de 200 pixels de côté. La région couverte par la vue principale y est représentée par un carré vert qui peut être déplacé par une interaction de drag afin de déplacer la vue principale à une granularité plus grande que le pan dans la vue principale. De plus, la représentation de la vue d'ensemble est dynamique, comme si elle suivait la caméra associée à la vue détaillée, et son facteur d'échelle est éventuellement ajusté de façon à ce que le facteur d'échelle entre la vue globale et la vue détaillée ne soit jamais supérieure à 24 (à la manière de la vue d'ensemble implémentée par Google Maps[1]).

La troisième technique est la *lentille de distortion* (FL) qui grossit la région autour du curseur. Le diamètre total de la lentille est de 200 pixels et elle présente une région centrale non distordue sur un rayon de 60 pixels qu'elle intègre au contexte selon une fonction de descente gaussienne et la distance euclidienne $L(2)$ [50]. La lentille est activée (Figure 4.16-b) lorsque l'utilisateur clique le bouton droit de la souris et désactivée lorsqu'il clique une nouvelle fois le bouton droit. Le facteur de grossissement de la région centrale de la lentille est initialement fixé à quatre fois le facteur d'échelle du contexte et peut ensuite être ajusté en utilisant la roulette de la souris (en restant dans l'intervalle de 2 fois à 12 fois le facteur d'échelle du contexte). Les actions sur la roulette de la souris sont donc contextuelles : si la lentille est active, c'est le facteur d'échelle de la lentille qui est modifié ; sinon, c'est le facteur d'échelle du contexte qui est modifié.

La dernière technique est le *DragMag* (DM) [172], une technique qui n'avait jamais été évaluée auparavant. Le DragMag est composé de deux sous-fenêtres : la **pan window** entoure la région grossie par la **zoom window** dans le contexte (Figure 4.16-c). L'utilisateur peut activer et désactiver le DragMag avec des clics droits comme dans le cas de la lentille fiheye. L'utilisateur peut faire du pan de la région grossie à une granularité fine en faisant un drag directement dans la **zoom window** ou à une granularité plus grosse en déplaçant la **pan window** par un drag. L'utilisateur peut également déplacer la **zoom window** en utilisant la barre de titre de celle-ci dans les cas où l'occlusion est gênante. Enfin, la roulette de la souris ajuste le facteur de grossissement de la **zoom window** lorsque le curseur se trouve dans celle-ci (le facteur de grossissement de la zoom window suit les mêmes ajustements que ceux décrits dans le cas de la lentille).

Afin de comparer ces techniques, la vue d'ensemble, de la lentille ou de la **zoom window** occupe toutes la même aire à l'écran, 200 x 200 pixels, ce qui représente 4.5% de l'aire totale de l'espace d'affichage.

Prédictions

Nos prédictions pour cette expérience étaient les suivantes :

- **Le temps de complétion dépend du rang k de la cible.** En effet, quelle que soit la technique utilisée, le participant doit inspecter les objets un par un et nous faisons l'hypothèse que le temps de chaque navigation est le même. De plus, le coût associé à une revisite d'un objet étant relativement élevé, le nombre de revisites devrait être assez faible. Un modèle de choix pour l'interaction multi-échelle devrait donc prendre en compte l'effet de cette quantité d'exploration k et nous faisons la prédiction que, dans notre cas, cet effet sur le temps de complétion est linéaire.

[1] http ://maps.google.com

(a)

(b)

(c)

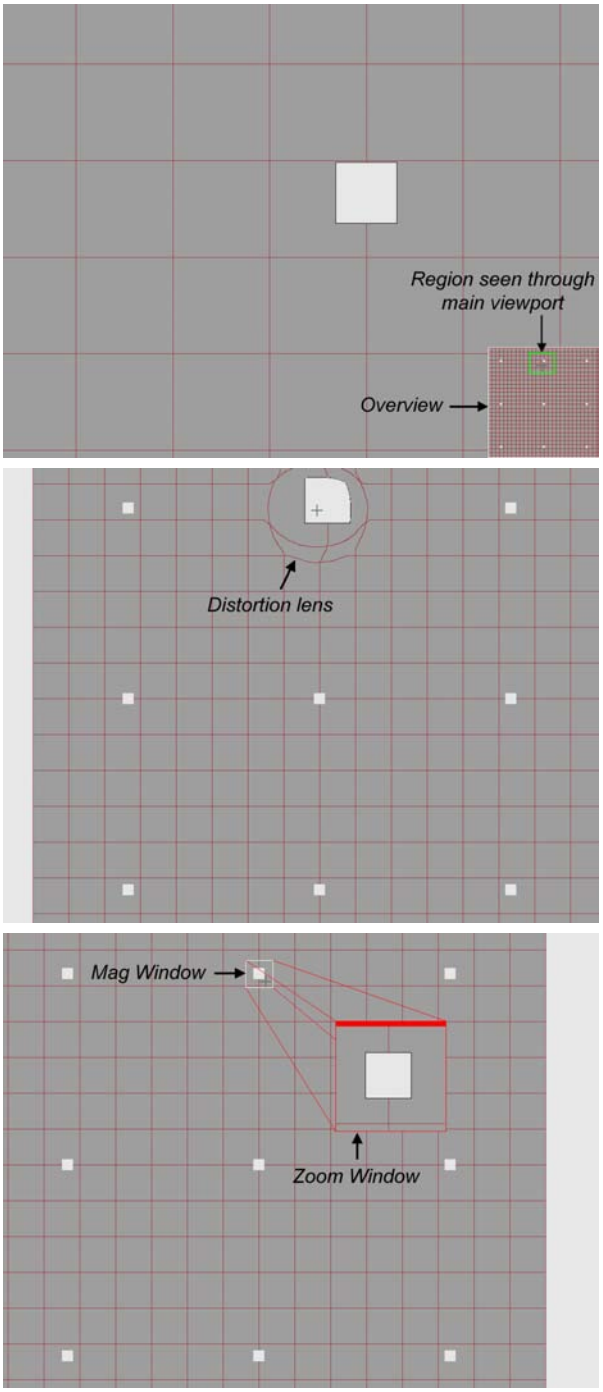


FIG. 4.16 : Dévoiler un objet avec (a) *Pan & Zoom + Overview* [OD], (b) *lentille de distortion* [FL], (c) *DragMag* [DM]

- **Focus + Context (FL, DM) et Overview + Detail (OD) sont plus efficaces que Pan & Zoom (PZ).** Avec PZ, naviguer d'un objet à l'autre est typiquement découpé selon la séquence zoom-in/pan/zoom-out. Avec DM, FL et OD, il s'agit simplement de déplacer la vue la plus détaillée d'un objet à l'autre. Puisque dans les trois cas, la position de la vue détaillée peut être contrôlée à travers la vue d'ensemble à une granularité plus grande, nous faisons l'hypothèse que ces trois techniques sont plus efficaces que PZ.
- **Overview + Detail (OD) est plus efficace que Focus + Context (DM and FL).** Avec OD, DM et FL, naviguer d'un objet a à un objet b revient à déplacer la vue détaillée de a vers b. Ce mouvement peut être vu comme un pointage réalisé en déplaçant une vue. Selon Guiard et al. [78], ces pointages sont des *pointages de vue* dont l'indice de difficulté dépend de la taille de la vue. Puisque la vue détaillée est plus grande pour la technique OD que pour les techniques DM et FL, OD devrait être plus efficace que les deux techniques *focus+context*.

Participants

Douze adultes volontaires, onze hommes et une femme, âgés de 23 à 52 ans (28 en moyenne) ont accepté de participer à notre expérimentation. Avant de commencer, l'expérimentateur montre successivement une série de carrés au coins arrondis et aux coins carrés en s'assurant que le participant perçoit bien la différence. Avant chaque bloc, l'expérimentateur explique comment utiliser la technique de ce bloc et le participant fait alors des essais choisis aléatoirement jusqu'à ce qu'ils se sentent suffisamment à l'aise avec la technique.

Matériel

L'expérimentation était exécutée sur un PC Dell Precision 380 équipé d'un processeur 3 GHz Pentium D, et d'une carte graphique NVidia Quadro FX4500. Le dispositif d'affichage était un écran LCD 1280x1024 (19") et le périphérique d'entrée était une souris optique Dell équipée d'une roulette. Le programme était écrit en Java 1.5 avec la boîte à outils ZVTM [140] spécialement adaptée pour les interfaces multi-échelle. Les dimensions de la fenêtre de l'application étaient 1080x824 pixels.

Stratégie de contrebalancement

	G_1	G_2	G_3	G_4
bc_1	S_1	S_4	S_7	S_{10}
bc_2	S_2	S_5	S_8	S_{11}
bc_3	S_3	S_6	S_9	S_{12}

FIG. 4.17 : Stratégie de contre-balancement pour les 12 participants (S_i).

Le plan expérimental est un plan 9x4 intra-participant : nous testons 9 valeurs de rang ($k \in [1..9]$) pour les 4 techniques (PZ, FL, DM et OD), c'est-à-dire $9 * 4 = 36$ conditions. Chaque condition est répliquée 3 fois pour que chaque participant exécute $9 * 4 * 3 = 108$ essais (≈ 45 minutes). La position

initiale de la région contenant les objets est différente pour chacune des trois réplications et est contrebalancée au sein des blocs en utilisant un carré latin. Les essais sont groupés en 4 blocs, un par technique, pour éviter de perturber le participant avec des changements de technique. Afin de minimiser les effets de l'ordre de présentation, nous avons calculé 4 ordres possibles en utilisant un carré latin et formé 4 groupes de 3 participants (G_1, G_2, G_3, G_4), un groupe par ordre.

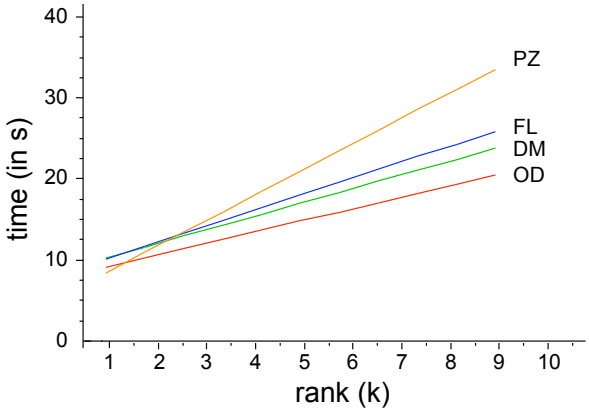
Nous contrebalançons aussi l'ordre de présentation des différentes valeurs de k dans un bloc : nous utilisons un carré latin pour calculer 9 ordres possibles et concaténons 3 ordres pour obtenir 1 bloc (3 ordres de 9 essais = 27 essais par bloc). Le carré latin fournit donc 3 compositions de bloc (bc_1, bc_2, bc_3) et nous associons une composition par participant au sein d'un groupe. La table de la figure 4.17 résume cette stratégie de contrebalancement.

Résultats

Pour chaque essai, le programme enregistre le temps de complétion, si l'essai se termine en succès (*hit*) ou en échec (*miss*) et, pour chaque objet, le nombre de fois où il a été dévoilé afin de compter les revisites. Chaque participant répondait également à un questionnaire en fin d'expérimentation afin de collecter les préférences qualitatives. Pour les analyses, nous avons retiré 14 essais *miss* ($\approx 1\%$) puis 31 essais *outliers* ($\approx 1,5\%$ des essais *hit*) en s'assurant que les essais *miss* et *outlier* était distribués aléatoirement entre les participants, techniques et valeurs de rang. L'analyse de variance révèle qu'il n'y a pas d'effet de l'ordre de présentation des techniques. Les effets d'apprentissage au sein d'une technique ne sont pas significatifs pour PZ ($p = 0.42$) et FL ($p = 0.75$), et sont légèrement significatifs pour DM ($p = 0.03$) et OD ($p = 0.02$).

Nous isolons le facteur *rang* (k) en l'analysant séparément pour chaque *technique*. Nous faisons une régression linéaire du temps relativement au rang, en traitant le participant comme une variable aléatoire, et obtenons les coefficients de corrélation élevés listés dans la table de la figure 4.18-b. Ceci supporte notre première prédiction : le temps de complétion est linéairement dépendant du rang (Figure 4.18-a).

Étant donné que le temps est linéairement dépendant du *rang*, nous faisons désormais nos analyses en traitant le *rang* comme un facteur continu. L'analyse de variance (avec la méthode REML pour les mesures répétées) révèle un effet simple du *rang* (k) sur le temps ($F_{1,411} = 1500.5$, $p < 0.0001$), un effet simple de la *technique* sur le temps ($F_{3,411} = 91.6$, $p < 0.0001$) et une effet d'interaction *rank* * *technique* significatif sur le temps ($F_{3,411} = 54.2$, $p < 0.0001$). La figure 4.19-a illustre ce résultat : plus le *rang* est grand, plus les différences entre techniques s'accentuent. Le test post-hoc de Tukey révèle que chaque technique est significativement différente des autres : OD est la technique la plus efficace, suivie de FL, puis DM et enfin PZ ($OD > DM > FL > PZ$). Ceci supporte nos deuxième et troisième prédictions : la technique *overview+detail* est plus efficace que les techniques *focus+context* qui sont elles-mêmes plus efficaces que le Pan & Zoom classique. Nous pensons que les différences entre FL et DM proviennent de la gêne introduite par la distorsion dans le cas de FL [85]. La figure 4.19-b montre également que la différence entre les moyennes de ces deux techniques (FL_{mean} = 18 s., DM_{mean} = 17 s.) est beaucoup plus petite que les autres différences entre paires (OD_{mean} = 14.8 s., PZ_{mean} = 21.2 s.). Les préférences subjectives collectées grâce au questionnaire de fin sont en accord avec ces résultats : 11 participants ont classé PZ comme la pire technique et 9 ont classé OD comme la meilleure technique.

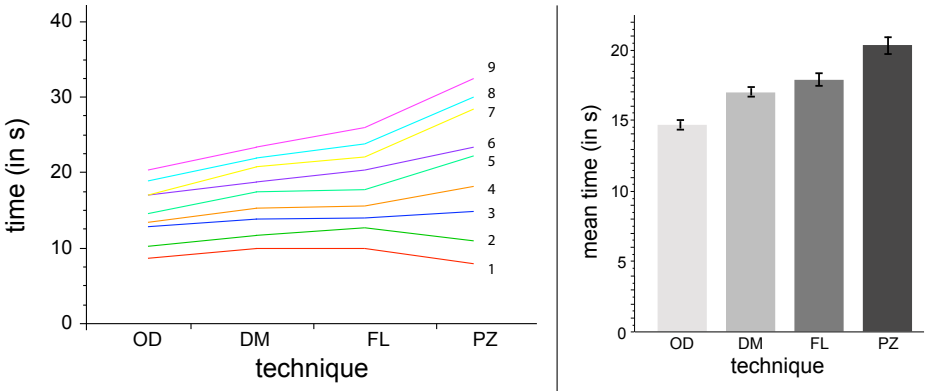


(a)

	PZ	FL	DM	OD
r^2	0.84	0.84	0.81	0.80
a	5.1	8.2	8.5	7.8
b	3.2	2.0	1.7	1.3

(b)

FIG. 4.18 : (a) Régression linéaire pour les quatre techniques ; (b) Coefficients de corrélation (r^2) et coefficients a et b ($time = a + b * rank$) pour les quatre techniques.



(a) (b)

FIG. 4.19 : (a) Effet d'interaction $rank * technique$ sur le temps de complétion (en s) ; (b) Temps de complétion moyen par technique (en s)

4.4.5 Vers un modèle de recherche plus général

La tâche de recherche introduite dans cette section couvre un ensemble de situations dans lesquelles l'utilisateur doit explorer chaque cible potentielle dans un environnement multi-échelle. Elle permet de mesurer le temps pris par les actions motrices impliquées en excluant les aspects cognitifs qui dépendent du domaine d'application. Le but est le même que le paradigme du pointage réciproque de Fitts et son utilisation en IHM : atteindre les performances limites de chaque technique et aider à élaborer des modèles empiriques prédictifs. L'opérationalisation que nous proposons couvre un large espace de conception dont la première étude présentée ci-dessus est un point limite de cet espace : un monde dans lequel les objets ont tous la même taille et sont uniformément disposés sur une grille, ce monde étant de petite taille (il existe un facteur d'échelle qui permet de scanner visuellement l'ensemble des objets). De plus

l'espacement entre les objets et la valeur de *minScale* ne permettant d'explorer qu'un seul objet à la fois. Les résultats de cette première étude ne peuvent donc à eux seuls permettre d'élaborer un modèle. Il est nécessaire d'étudier un plus grand nombre de points dans l'espace de conception.

Cependant, cet espace est vaste et il n'existe que très peu de guides théoriques nous permettant de structurer cet espace afin d'en étudier les points intéressants. Le degré d'indirection par rapport au but (*goal-directedness*) de Guiard et al. [83] peut aider à quantifier l'information dont les utilisateurs ont besoin pour reconnaître la cible tandis que les diagrammes espace-échelle de Furnas & Bederson [67] peuvent aider à explorer les différentes structures de mondes multi-échelle. Cependant, ces travaux ne sont pas directement applicables en pratique. Nous avons donc réalisé une interface réaliste afin d'identifier ces points d'une manière exploratoire. Cet environnement (Figure 4.20) affiche une version multi-échelle de la carte du monde de la NASA, Blue Marble Next Generation [166], dans laquelle nous avons ajouté un ensemble de données géographiques (pays, états, villes, parcs et lacs) que nous avons collectées à partir de la base de données Geo-Names[2]. L'environnement fournit un ensemble de techniques de navigation multi-échelle. Cet environnement réaliste ainsi que l'environnement abstrait utilisé dans l'étude sont disponibles[3] publiquement.

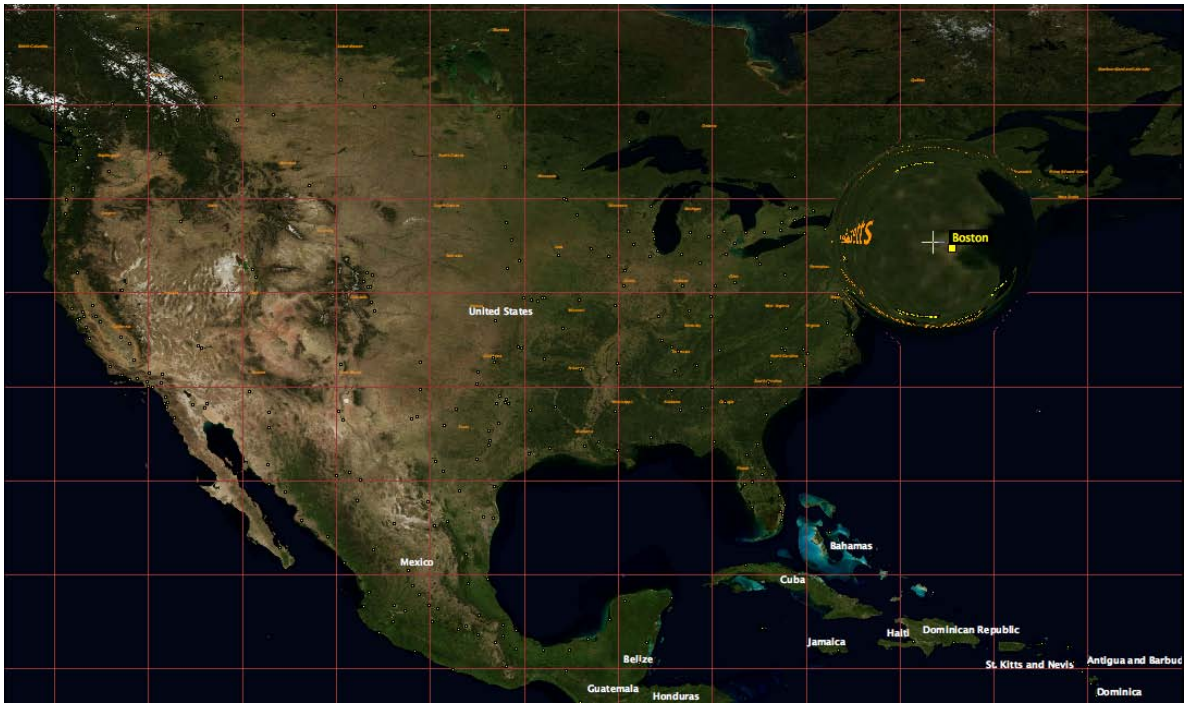


FIG. 4.20 : Rechercher Boston, Massachusetts, USA sur une carte du monde multi-échelle avec une lentille de distortion

[2]http ://www.geonames.org
[3]http ://zvtm.sourceforge.net/eval/pb

(a)

(b)

FIG. 4.21 : Explorer une région éparse avec un drag-mag (a), et une région dense avec une vue d'ensemble (b)

Ces deux environnements combinés sont des outils utiles pour élaborer un modèle de recherche ou de choix multi-échelle. L'environnement permet d'identifier les points intéressants et l'environnement abstrait de les étudier de manière contrôlée. Par exemple, l'expérimentation présentée plus haut correspond à la recherche d'une grande ville en Australie (Figure 4.21-a) : un nombre restreint de cibles potentielles à explorer, relativement distantes les unes des autres. Pour cette tâche, le temps de recherche est fonction du nombre de cibles à explorer et la technique la plus efficace parmi celles testées est la technique *overview+detail*. L'interaction typique consiste à ajuster initialement l'échelle et la position de manière à ce que le nom de la ville soit lisible dans la vue détaillée et que la vue d'ensemble soit à une position qui permette de voir tous les points représentant les villes. Ainsi, il est possible de déplacer la vue détaillée

d'une ville à l'autre par un simple drag dans la vue d'ensemble.

D'autres observations que nous avons collectées lors d'expérimentations pilotes dans cet environnement réaliste ouvrent la voie à d'autres études. Par exemple, dans les régions ayant une densité forte, comme l'Asie du sud-est qui contient un grand nombre de villes (Figure 4.21-b), les utilisateurs ont aussi tendance à utiliser la technique *overview+detail* afin de pouvoir afficher le maximum de villes avec leur nom lisible à la fois. Ce cas pourrait être étudié en changeant la valeur de *minScale* dans notre tâche abstraite. Il serait également intéressant d'étudier des mondes abstraits dans lesquelles les différents objets ont différentes valeurs de *minScale* (dans l'environnement concret, les pays et les villes sont visibles à des niveaux d'échelle différents).

4.5 Conclusion

La première étude présentée dans ce chapitre montre que les prédictions de SimCIS sont en accord avec les observations empiriques. Cependant, le cadre étudié ne couvre pas tout le spectre des techniques d'interaction existantes. En effet, SimCIS dépend de la connaissance que nous avons des lois empiriques sous-jacentes aux actions CIS. Nous avons donc besoin de nouveaux cadres expérimentaux afin d'étudier finement ces actions de manière contrôlée pour pouvoir élaborer de nouvelles lois empiriques.

La deuxième étude de ce chapitre est une première piste vers une extension de CIS aux interfaces multi-échelle. Elle est fondée sur un cadre expérimental abstrait pour opérationnaliser et évaluer une tâche de recherche dans une interface multi-échelle. Bien que nous ayons mené une étude préliminaire d'une recherche dans un monde virtuel très spécifique, notre approche est plus générale et l'application concrète que nous présentons permet d'identifier de manière exploratoire les différents cas à étudier pour proposer un modèle de recherche multi-échelle plus général. Plus la connaissance de ces lois empiriques "atomiques" est grande, plus la couverture de CIS est étendue et plus les aspects prédictif et génératif de CIS sont améliorés. En effet, les concepteurs d'interfaces peuvent ainsi étudier des possibilités dans un éventail de plus en plus large.

Implémenter les techniques d'interaction avec SwingStates

Jusqu'ici, nous nous sommes concentrés sur les outils dont dispose le concepteur d'applications graphiques interactives pour explorer les différentes possibilités qui s'offrent à lui et faire les meilleurs choix. Désormais, nous nous intéressons aux moyens dont disposent les développeurs pour programmer les techniques d'interaction retenues. À l'heure actuelle, les développeurs utilisent principalement des boîtes à outils d'interface fondées sur le modèle classique du widget [25]. Ces bibliothèques offrent un jeu de widgets couvrant les interactions standard telles que menus et boutons, et permettent de construire une interface graphique en les assemblant. Ces widgets sont reliés entre eux et au reste de l'application en utilisant des fonctions de rappel (*callback*) ou des écouteurs d'événements (*listener*). D'une part, les applications produites sont difficiles à développer et à maintenir car il est difficile de suivre clairement le fil d'exécution et, d'autre part, la réalisation de nouveaux widgets est très difficile avec ces outils. De nombreuses boîtes à outils ont été développées par la recherche afin d'augmenter la puissance d'expression, notamment pour couvrir les nouvelles techniques d'interaction telles que les marking menus [104], les magnetic guidelines [25] ou l'interaction gestuelle. Malheureusement, ces boîtes à outils sont restées expérimentales tandis que les boîtes à outils industrielles ne permettent toujours pas de mettre en œuvre ces nouvelles techniques d'interaction, au détriment de l'utilisabilité des applications graphiques. Encourager une approche de conception générative pour favoriser l'introduction de techniques originales et efficaces n'a de sens que si les outils utilisés par les développeurs permettent de les mettre en œuvre.

Nous présentons ici *SwingStates* [12] que nous avons développé selon une approche différente qui consiste à augmenter une boîte à outils industrielle, en l'occurence Java Swing, afin de permettre la mise en œuvre de nouvelles techniques d'interaction avec les outils habituellement utilisés par les développeurs. La force de SwingStates réside dans l'intégration entre les machines à états, le langage Java et les boîtes à outils Swing et Java2D. L'augmentation d'un outil existant nécessite quelques compromis comme l'impossibilité de descendre à un niveau d'abstraction inférieur à celui de l'outil et le respect de ses schémas de conception. En contrepartie, l'avantage escompté est de faire adopter par les développeurs de nouveaux concepts et des outils d'interface avancés tout en capitalisant les connaissances dont ils disposent. L'objectif est d'avoir ainsi un impact réel sur le développement des applications, l'outil augmenté s'intégrant dans un environnement connu des développeurs. Conscients du fait que ce but est ambitieux puisqu'il requiert en particulier de fournir des outils de qualité industrielle, nous nous intéressons dans un premier temps à deux catégories particulières d'usages qui ne nécessitent pas des outils aussi robustes : le prototypage d'interfaces et l'enseignement de l'IHM. Ceci nous permet d'une part d'exposer et d'accoutumer les développeurs et futurs développeurs à des techniques d'interaction à des outils de développement de nouvelle génération, d'autre part d'évaluer notre extension en la soumettant à un usage in situ. Le développement de techniques originales à moindre frais que permet SwingStates encourage une approche générative intervenant dans les stades de conception plus avancés.

5.1 Les boîtes à outils existantes et l'approche de SwingStates

Outre Java Swing, GTk [113] et Qt [61] sont largement utilisées pour le développement d'interfaces, de même que les boîtes à outils natives de MacOS et Windows. Toutes ces boîtes à outils n'offrent qu'un jeu de widgets réduit au vocabulaire classique des interfaces graphiques et sont difficilement extensibles. La communication entre les périphériques d'entrée et les widgets et celle entre les différents widgets se fait selon le paradigme classique d'*écouteurs d'événements*. Un écouteur d'événement (i) s'abonne

auprès d'un objet émetteur pour écouter un certain type d'événements et (ii) implémente une fonction de rappel (ou *callback*) par type d'événement. Or, de nombreux travaux montrent que ce style de programmation n'est pas toujours approprié pour l'interaction [128, 96], les programmes ressemblant à des "spaghettis de callbacks" difficiles à relire et maintenir [128]. Il existe un très grand nombre de boîtes à outils d'interface qui sont beaucoup moins répandues, les thèses de Dragicevic [58] et Blanch [31] fournissent un état de l'art très complet. Ici, nous listons seulement celles qui ont directement inspiré notre travail soit parce qu'elles sont programmées en Java soit parce qu'elles ont exploré de nouvelles façons de programmer l'interaction.

Il existe certaines boîtes à outils en Java spécialisées pour certains types d'interfaces. Ainsi, les boîtes à outils Piccolo [28] et Zoomable Visual Transformation Machine (ZVTM) [140] sont dédiées spécifiquement à la programmation des interfaces zoomables. La boîte à outils InfoVis [63] a été conçue pour faciliter la visualisation d'informations, c'est-à-dire définir la correspondance entre des attributs des données et des attributs graphiques. Les avantages de ces boîtes à outils se situent au niveau du modèle graphique alors que la programmation de l'interaction se fait en utilisant les écouteurs d'événements standards de Java. La boîte à outils SubArctic [93], également programmée en Java, propose une approche légèrement différente de l'approche par écouteurs d'événements en utilisant des agents et des politiques de distribution des événements mais ne fournit pas de support pour programmer de nouveaux agents facilement. Enfin, la boîte à outils ICON est certainement la plus originale pour la spécification de l'interaction puisqu'elle repose sur un formalisme réactif pour programmer l'interaction. Ici, le programmeur sort complètement de son cadre habituel et doit définir une technique d'interaction par un ensemble de fonctions des entrées vers des attributs des objets graphiques.

Plutôt qu'une totale redéfinition, SwingStates se situe plus dans une approche d'extension de Swing pour faciliter son adoption à l'image de GmlCanvas [41] avec Tcl/Tk [136]. En effet, le widget canvas de Tcl/Tk est bien adapté à la programmation grâce à la puissance d'expression des tags et la concision de la syntaxe. Les tags sont de simples chaînes de caractères qui peuvent être utilisés pour grouper des objets ou les distinguer pour la programmation de l'interaction. GmlCanvas [41] est une extension à Tcl/Tk qui reprend le principe du canvas Tk en augmentant son modèle graphique grâce à un rendu OpenGL. Nous reprenons des éléments du modèle graphique de GmlCanvas, mais sur la base d'une boîte à outils plus largement répandue chez les développeurs et en ajoutant un modèle élaboré pour la gestion de l'interaction. La contribution principale sur ce point est donc l'intégration d'un modèle graphique approprié à la programmation de l'interaction dans l'environnement Java en relevant le défi du passage d'un langage de scripts (Tcl) à un langage à objets typé (Java).

Pour pallier les problèmes dûs à la gestion des entrées par des écouteurs d'événements, SwingStates intègre les machines à états dans le langage Java pour programmer l'interaction. Depuis le travail séminal de Newman [132], les machines à états ont très souvent été utilisées pour décrire l'interaction mais rarement pour la programmer. Par exemple, les Statecharts [86] sont une extension du formalisme des machines à états qui a été conçu pour décrire des systèmes complexes dirigés par des événements mais, à notre connaissance, ils n'ont jamais été implémentés probablement à cause de leur sémantique complexe. Plus récemment, les *Interactors* implementés dans la boîte à outils Garnet [126] et son successeur, Amulet [127], sont des objets qui interceptent des événements pour les traduire en opérations

sur une forme graphique cible. Ils utilisent une unique machine à états prédéfinie (*press-interact-release*) qui peut être paramétrée. Cependant, de nombreuses interactions ne peuvent être programmées à partir de cette machine à états. Par exemple, les techniques d'interaction qui utilise le temps pour déclencher des actions, comme l'affichage d'un *tooltip* lorsque le curseur reste immobile sur un widget pendant un certain temps ou encore les dossiers à ressort (*spring-loaded*) de MacOS[1], requièrent des machines à états plus évoluées.

Jacob et al. [96] ont proposé un modèle et un langage pour capturer la structure formelle des interactions post-WIMP qui reposent sur les machines à états. Pour utiliser ce modèle, le développeur doit disposer d'un éditeur graphique particulier pour entrer sa description qu'il peut ensuite transformer en code fonctionnel en cliquant sur chacun des éléments pour spécifier son comportement en code C++. HsmTk [32] est une librairie pour programmer l'interaction avancée en utilisant les machines à états hiérarchiques. Avec HsmTk, la réalisation d'une interface se fait en deux étapes : le rendu graphique est décrit en SVG (Scalable Vector Graphics) et le comportement est programmé avec des machines à états hiérarchiques. Pour fournir un format textuel naturel pour définir des machines à états, HsmTk utilise un pré-processeur pour générer du code C++ à partir de ces définitions. Dans SwingStates, l'interaction est également spécifiée par des machines à états mais ces machines sont directement écrites en Java, les développeurs n'ayant besoin d'aucun outil supplémentaire. À l'image de la boîte à outils Ubit [109] qui utilise la surcharge des opérateurs en C++, nous exploitons les capacités du langage hôte afin de rendre la syntaxe simple et efficace et d'éviter d'avoir recours à des environnements de programmation non standards.

5.2 Les machines à états et le langage Java

5.2.1 Machines à états vs. Écouteurs d'événements

Un écouteur d'événements est un objet capable d'entendre un type d'événement particulier (des clics de souris, des appuis de touches de clavier, etc.) qui arrivent sur le composant graphique auquel il est abonné. Cet écouteur d'événement doit implémenter les fonctions de rappel ou callback à exécuter lorsqu'un événement se produit. Par exemple, le petit programme suivant permet d'afficher un message chaque fois que l'utilisateur clique sur le composant graphique `canvas` :

```
// L'écouteur d'événements provenant des boutons de souris
 class AMouseListener implements MouseListener {
    public void mouseClicked(MouseEvent e) {
       System.out.println("Click");
    }
    public void mousePressed(MouseEvent e) {  }
    public void mouseReleased(MouseEvent e) { }
 }
 // Abonnement de l'écouteur au composant graphique
 canvas.addMouseListener(new AMouseListener());
```

[1]Si, au cours d'un drag'n drop, l'utilisateur s'arrête un certain temps au-dessus d'un dossier, celui-ci s'ouvre pour dévoiler son contenu

Dans ce type d'approche, il y a généralement un *callback* par type d'événements, tous les callbacks étant actifs à la fois.

Au contraire, une machine est constituée d'un ensemble d'états, dont un état initial qui est l'état de départ et un état actif qui varie au cours du temps. Le passage d'un état à un autre est décrit par des transitions, qui sont déclenchées par l'arrivée d'événements tels que le déplacement de la souris, l'appui sur un bouton, l'écoulement d'un délai, etc. Ces transitions ne peuvent être déclenchées que lorsque leur état de départ est l'état actiif. A chaque transition peuvent être associées :

- une garde, c'est-à-dire une condition booléenne qui retourne Vrai lorsque la transition peut être déclenchée. Une garde non spécifiée retourne toujours Vrai.
- une action, qui est du code exécuté lorsque la transition est déclenchée.

A chaque état peuvent être associées :

- une action d'entrée, qui est du code exécuté lorsque l'état devient l'état courant ;
- une action de sortie, qui est du code exécuté lorsque l'état cesse d'être l'état courant.

Au départ, l'état courant est l'état initial. Lorsqu'un événement arrive, s'il existe une transition correspondant à cet événement dans l'état courant et dont la garde retourne Vrai, cette transition est déclenchée :

1. l'action de sortie de l'état courant est exécutée ;
2. l'action de la transition est exécutée ;
3. l'état courant devient l'état d'arrivée de la transition et son action d'entrée est exécutée.

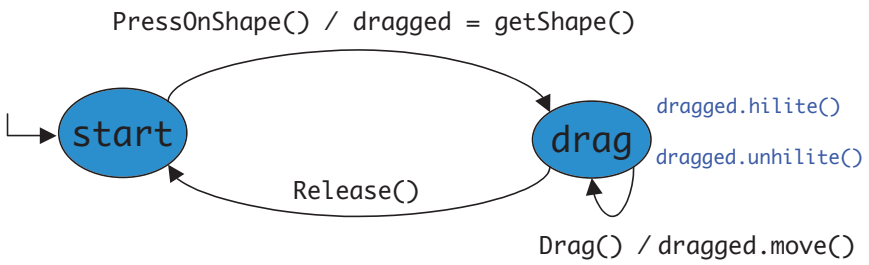


FIG. 5.1 : Représentation graphique d'une machine à états pour le drag'n drop

La figure 5.1 montre une représentation graphique d'une machine à états simple. Les états sont des ellipses et les transitions sont des flèches orientées de leur état de départ vers leur état d'arrivée. Les transitions sont étiquetées par des expressions de la forme événement / action. Les états contiennent leur nom et l'état initial est repéré par une flèche en forme de L.

La figure 5.2 résume la dualité des deux approches (écouteurs et machines). Alors que le pouvoir d'expression est le même dans les deux cas, l'approche cognitive suivie par le développeur est différente. En effet, dans l'approche par écouteurs, tous les *callbacks* sont actifs à la fois et, l'action déclenchée

par un événement dépend généralement de l'état de l'interface, le code commence par une série de tests pour identifier l'état de l'interface et exécuter la bonne action (dans l'exemple de la figure 5.2), un clic de souris lorsqu'un outil est sélectionné provoque la création d'une forme alors qu'un clic lorsqu'aucun outil n'est sélectionné va sélectionner la forme qui est en dessous s'il y en a une). Alors que, dans l'approche par machines, le code est organisé selon les différents états de l'interface, chaque état n'écoutant que les événements qui ont de l'intérêt pour lui, c'est-à-dire ceux qui déclenchent des transitions. Les écouteurs d'événement suggèrent donc une approche *centrée sur les événements* (à partir d'un type d'événement, le développeur se demande "Comment l'interface répond-elle à cet événement dans chacun de ses états ?") alors que les machines à états suggèrent une approche *centrée sur les états* (à partir d'un état, le développeur se demande "Quels sont les événements qui ont un effet lorsque l'interface est dans cet état ?"). Il n'y a pas de meilleure solution absolue, ces deux approches peuvent être utilisées conjointement avec SwingStates. L'approche par écouteurs est plus appropriée lorsque l'interface a peu d'états et qu'ils diffèrent peu dans leur comportement alors que l'approche par machines est plus appropriée dans le cas d'une interface ayant plusieurs états qui diffèrent dans leur comportement. Les machines à états montrent cependant les avantages (i) de faire l'économie des différentes variables globales pour décrire l'état de l'interface et (ii) de centraliser les *variables propres à l'interaction* dans la machine. Pour éclaircir ce dernier point, considérons l'interaction de drag'n drop de la figure 5.1 : avec des écouteurs d'événements (partie droite de la figure 5.2), la forme graphique déplacée doit être une variable globale à l'application afin que le callback pour les événements *press* et celui pour les événements *drag* puissent la partager ; avec une machine (partie gauche de la figure 5.2), la forme graphique peut être une variable locale à la machine, la machine étant l'objet qui représente l'interaction.

5.2.2 Syntaxe des machines à états avec SwingStates

Pour programmer des machines à états directement en Java, SwingStates utilise le mécanisme des classes internes (*inner class*) pour :

- décrire les machines à états avec une syntaxe qui reste proche des formats textuels habituellement utilisés pour décrire des machines à états tout en étant purement en Java donc directement interprétable par la machine virtuelle.
- rester dans des paradigmes familiers aux développeurs d'applications graphiques interactives en Java.

En effet, les classes internes sont très souvent utilisées pour écrire des écouteurs d'événements. Par exemple, l'écouteur de notre premier exemple qui affiche un message en cas de clic de souris peut se réécrire comme ci-dessous :

```
1 canvas.addMouseListener(
2    new MouseListener() {
3        public void mouseClicked(MouseEvent e) {
4           System.out.println("Click");
5        }
6        public void mousePressed(MouseEvent e) {  }
7        public void mouseReleased(MouseEvent e) {  }
8    }) ;
```

Les lignes 2 à 7 spécifient une sous-classe de la classe `MouseListener` dont une unique instance est créée et passée en paramètre à la méthode `addActionListener`. De plus, le nouvel objet est créé

Machine à états

état 1
événement d'intérêt 1
exécuter action
...
événement d'intérêt m_1
exécuter action

état n
événement d'intérêt 1
exécuter action
...
événement d'intérêt m_n
exécuter action

```
StateMachine {
  Shape dragged;

  State start {
    PressOnShape --> drag {
      action() { dragged = getShape(); }
    }
  }

  State drag {
    enter() { dragged.hilite() }
    Drag --> drag {
      action() { dragged.move(); }
    }
    Release --> start { }
    leave() { dragged.unhilite() }
  }
}
```

Écouteur d'événements

variables globales décrivant l'état de l'application
événement 1
test 1 sur l'état
exécuter action
...
test m_1 sur l'état
exécuter action

événement n'
test 1 sur l'état
exécuter action
...
test m'_n sur l'état
exécuter action

```
Shape dragged = null;
boolean first = true;

PressOnShape() {
  dragged = getShape();
}

Drag() {
  if(dragged == null) return;
  if(first) {
    first = false;
    dragged.hilite();
  }
  dragged.move();
}

Release() {
  dragged = null;
  dragged.unhilite();
  first = false;
}
```

FIG. 5.2 : Machines à états VS. Écouteurs d'événements

"à l'intérieur" de l'objet qui exécute ce code et il a un accès direct à l'état et aux méthodes de cet objet englobant.

Les machines à états de SwingStates exploitent cette possibilité de spécifier une sous-classe partout où l'on peut créer un objet et de créer des objets ayant directement accès à leur contexte. La machine de la figure 5.1 s'écrit alors comme ci-dessous :

```
1  sm = new CStateMachine(canvas) {

2     CShape dragged = null;
3     Point2D pPrevious = null;
4     Paint fillPaint = null;

5     public State start = new State() {
6       Transition dragOn = new PressOnShape(BUTTON1, ">> drag"){
7           public void action(){
8              dragged = getShape();
9              pPrevious = getPoint();
10          }
11       };
12    };

13    public State drag = new State() {
14       public void enter() {
15          fillPaint = dragged.getFillPaint();
16          dragged.setFillPaint(Color.green);
17       }
18       Transition drag = new Drag(BUTTON1){
19          public void action(){
20             dragged.translateBy(
21                getPoint().getX()  pPrevious.getX(),
22                getPoint().getY()  pPrevious.getY());
23             pPrevious = getPoint();
24          }
25       };

26       Transition dragOff = new Release(BUTTON1, ">> start"){
27          public void action(){
28             dragged.translateBy(
29                getPoint().getX()  pPrevious.getX(),
30                getPoint().getY()  pPrevious.getY());
31          }
32       };
33       public void leave(){
34          dragged.setFillPaint(fillPaint);
35       }
36    };

37 };
```

Une machine à états est un objet de la classe `CStateMachine`. Les états de cette machine sont représentés dans des champs de cet objet, ici `start` et `drag`. Chaque état est une sous-classe anonyme de `State` (l.5 et 13). Les actions d'entrée et de sortie d'un état sont spécifiées en surchargeant les méthodes `enter()` (l.14) et `leave()` (l.33). Les transitions sont définies de façon similaire, sous forme de champs de chaque état (l.6, 18 et 26). SwingStates définit un ensemble de classes dérivées de `Transition` correspondant aux différents événements possibles. Les transitions sont des sous-classes

anonymes de ces différentes classes de transition dans lesquelles on redéfinit les méthodes `action()` (l.7, 19 et 27) et `guard()` pour spécifier l'action et la garde de la transition. L'état de sortie d'une transition est toujours le dernier argument de son constructeur et est spécifié par une chaîne de caractères qui peut être précédée de n'importe quel symbole parmi l'ensemble `{-, >, =}`. Par exemple, `"» drag"` (l.6) référence l'état `drag` (l.13). Nous utilisons une chaîne de caractères plutôt que l'objet lui-même puisqu'il peut ne pas être déclaré à ce moment. Nous utilisons l'interface d'introspection de Java au moment de l'initialisation de la machine pour mettre à jour les références aux états (c'est pourquoi chaque état doit être `public`).

Ces différentes classes réifient le formalisme des machines à états en objets Java et constituent une grande originalité de SwingStates. A notre connaissance, les autres implémentations de machines à états en Java n'utilisent pas cette approche, mais traduisent une description graphique ou textuelle de la machine à états en code exécutable Java.

La figure 5.3 montre les quatre types de machines à états de SwingStates. Chaque machine contient différentes classes de Transition adaptées au type d'objets que l'on veut contrôler. La classe `StateMachine` ne gère que des événements virtuels non reliés à un périphérique (par exemple, des événements qui sont les noms des commandes d'une application) et des événements liés au temps (un *timer* est arrivé à expiration). La classe `BasicInputStateMachine` ajoute un ensemble de classes de Transition pour les événements clavier (KeyPress, KeyRelease et KeyType) et souris (Press, Release, Click, Move et Drag). La classe `CStateMachine` ajoute des transitions pour programmer l'interaction avec un canvas de SwingStates pour définir un nouveau widget alors que la classe `JStateMachine` ajoute des transitions pour re-programmer l'interaction avec un widget prédéfini de Swing. Nous reviendrons sur ces deux dernières classes au cours des sections suivantes.

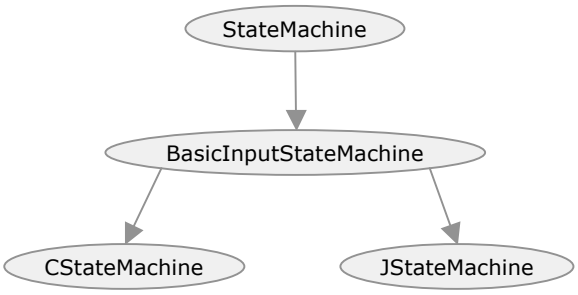


FIG. 5.3 : Les classes de machines à états dans SwingStates

5.3 Un canvas à personnaliser

Le modèle graphique de SwingStates repose sur celui de Java2D mais permet au programmeur de penser en termes d'objets dotés non seulement d'attributs de rendu mais aussi d'attributs pour l'interaction. Les formes graphiques d'un *canvas* SwingStates peuvent être liées par différentes relations et

étiquetées par des tags avancés. Leurs attributs et leurs relations sont exploitées par des machines à états spécifiquement dédiées au canvas, `CStateMachine`, afin d'offrir un vocabulaire d'événements plus riche que celui de Java Swing. Dans cette section, nous présentons la flexibilité que ce widget offre afin de permettre aux développeurs de définir facilement de nouveaux widgets. Dans un premier temps, nous présentons comment définir le rendu graphique d'un widget et, dans un second temps, nous présentons comment programmer l'interaction grâce à la classe de machines à états qui permet de gérer les éléments de ce canvas.

5.3.1 Le modèle graphique

Un canvas contient une liste d'objets graphiques stockés dans une liste d'affichage. Chaque objet graphique est une spécialisation de la classe de base `CShape` (Figure 5.4). Un objet de type `CShape` dispose de ses propres attributs de style contrairement à l'approche Java2D qui incite à raisonner en termes d'attributs d'un contexte graphique global et non pas en termes d'attributs pour chaque objet graphique. Les champs qui définissent le rendu d'un objet graphique dans un canvas SwingStates réutilisent les classes existantes de la librairie Java2D couramment utilisée pour faire du dessin en 2D (`Shape`, `Paint`, etc.).

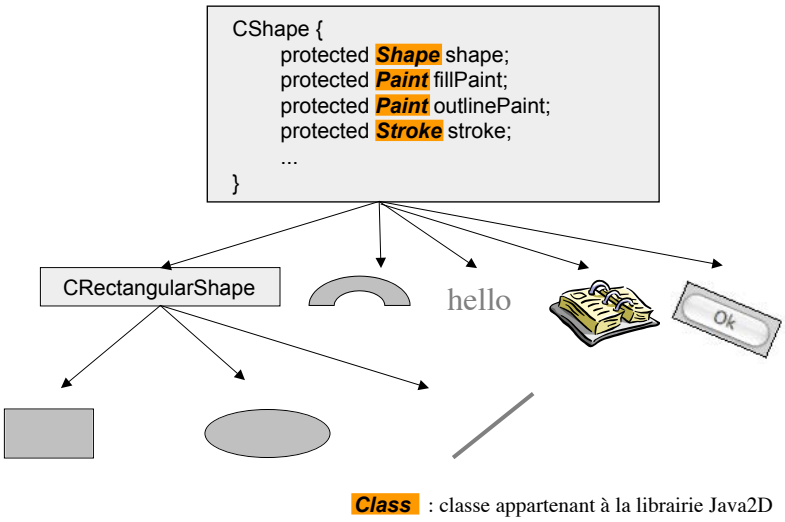


FIG. 5.4 : Les classes d'objets graphiques dans SwingStates

SwingStates propose un ensemble de classes de formes graphiques prédéfinies répertoriées par la figure 5.4 : `CRectangle`, `CEllipse`, `CSegment`, `CText`, `CImage`, `CPolyLine` et `CWidget`. Ces classes proposent les méthodes générales de la classe `CShape` pour ajuster les attributs de style et de géométrie mais aussi des méthodes plus spécifiques comme la méthode `setText(t)` pour changer la chaîne de caractères d'un objet `CText` ou la méthode `lineTo(x, y)` pour ajouter un point à un objet `CPolyLine`. `CWidget` est une classe originale qui permet aux développeurs d'ajouter n'importe quel widget Swing dans le canvas et de le manipuler comme n'importe quelle autre forme graphique (par

exemple, changer sa couleur de fond ou encore lui appliquer une rotation) tout en gardant la possibilité d'interagir avec ce widget de façon standard (par exemple, cliquer sur un bouton pour l'invoquer).

Les coordonnées d'un objet sont, par défaut, relatives au repère du canvas, l'origine étant le coin haut gauche. Ces coordonnées peuvent être transformées par rotation, translation et changement d'échelle de manière relative ou absolue : par exemple, `translateBy(dx, dy)` ajoute le vecteur (dx, dy) aux coordonnées courantes alors que `translateTo(x, y)` change les coordonnées pour placer l'objet à la position (x, y). Par défaut, le point de référence d'une transformation est le centre de l'objet graphique. Ce point de référence peut être modifié en le spécifiant par rapport à sa boîte englobante : (0, 0) correspond au point haut gauche de cette boîte et (1, 1) au point bas droite. Par défaut, il vaut (0.5, 0.5), soit le centre de l'objet. Pour un menu contextuel linéaire, on utilise un point de référence (0, 0) et une translation absolue pour amener le menu en bas à droite du curseur alors qu'on utilise le point de référence (0.5, 0.5) et une translation absolue pour amener un menu circulaire centré autour du curseur. La figure 5.5 illustre les différents résultats obtenus en fonction du type de la transformation (relative ou absolue) et la valeur du point de référence.

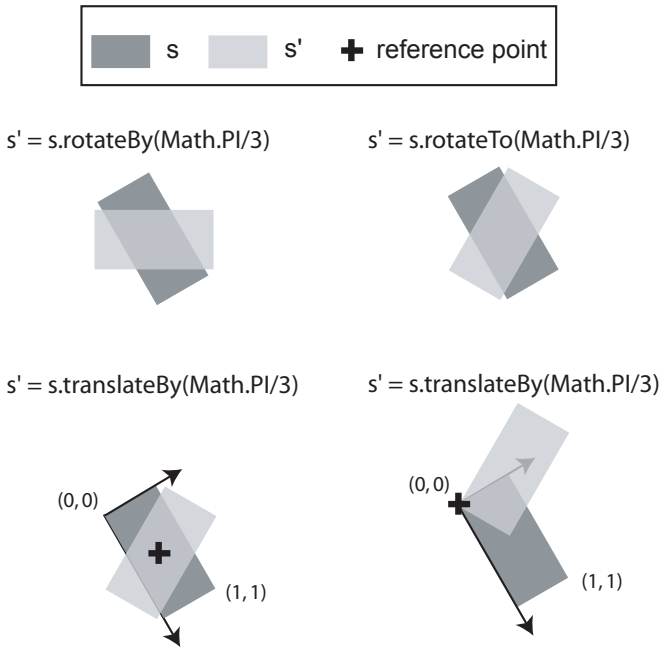


FIG. 5.5 : Les transformations dans le canvas SwingStates

Chaque objet peut être ajouté ou enlevé de la liste d'affichage et son rang dans cette liste changé. Un objet `CShape` a également un attribut spécial, `drawable`, qui spécifie si cette forme est visible ou non. Ceci est particulièrement utile pour optimiser certaines techniques comme les menus contextuels qui ne sont pas toujours visibles et qui devraient être de nombreuses fois ajoutés puis retirés de la liste. Le rendu graphique consiste à afficher chacun des objets graphiques marqués comme visibles dans l'ordre de la

liste d'affichage. Outre la relation définie par la liste d'affichage, les formes graphiques peuvent être liées par une relation hiérarchique et une relation de *clipping*. En effet, un objet peut avoir un parent afin de pouvoir exprimer ses coordonnées par rapport à ce parent (la transformation du parent et de l'objet sont combinées) et un objet peut avoir un *clip* de sorte qu'il ne sera visible qu'à travers celui-ci. Comme GmlCanvas [41] et contrairement à d'autres graphes de scène comme celui d'Inventor [167], les relations d'affichage, de hiérarchie et de clipping sont indépendantes les unes des autres. La figure 5.6 montre la flexibilité offerte par l'indépendance de ces trois relations sur l'exemple du marking menu implémenté avec SwingStates.

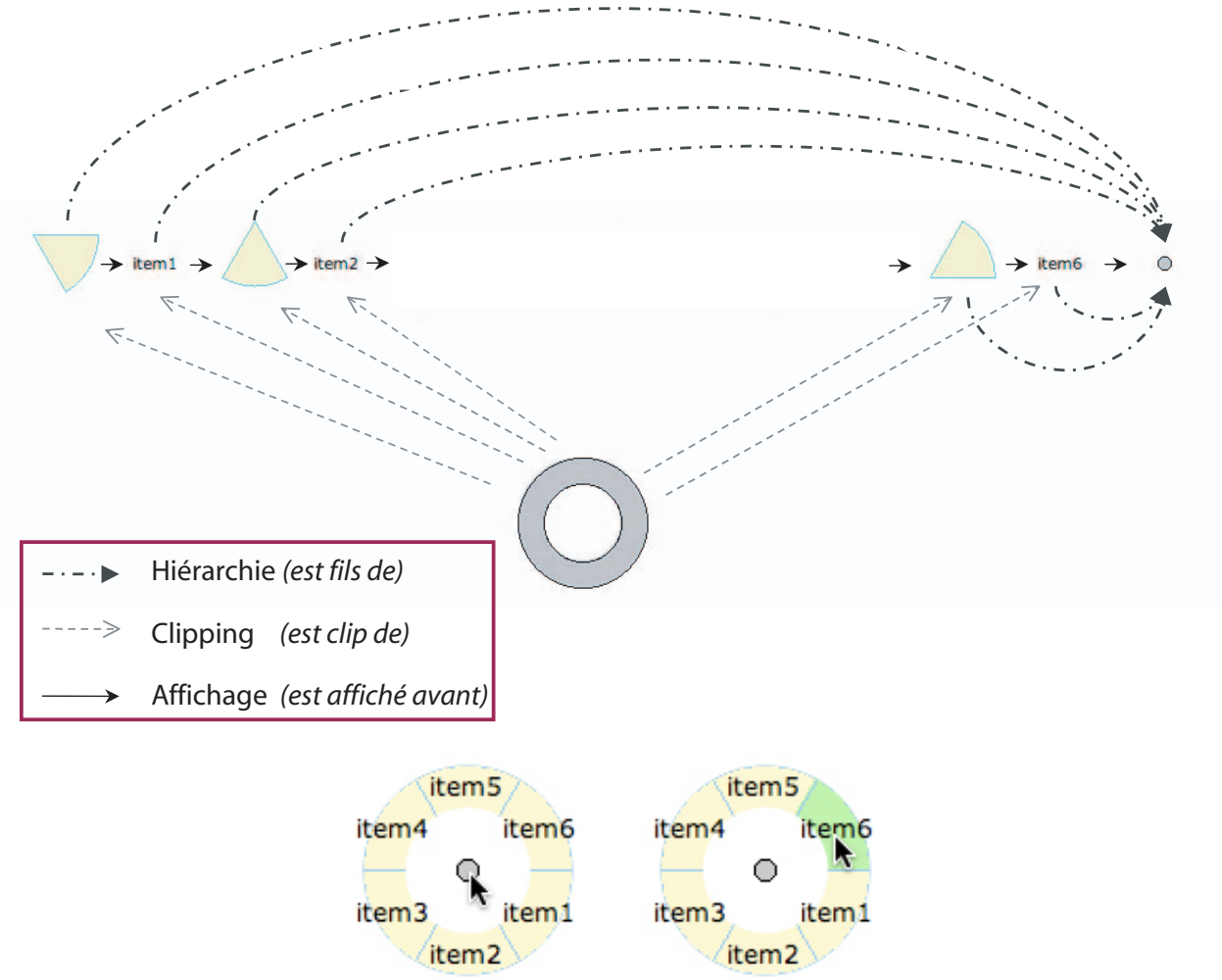


FIG. 5.6 : Un marking menu réalisé avec SwingStates

5.3.2 Les tags

Comme dans la boîte à outils Tk [136], chaque objet graphique peut porter une ou plusieurs étiquettes (*tags*). Un tag est une collection d'objets avec un itérateur permettant de l'énumérer :

```
for(aTag.reset(); aTag.hasNext(); )
   CShape aTaggedShape = aTag.nextShape();
}
```

Chaque objet peut avoir plusieurs tags et un même tag peut être apposé à différents objets. Lorsqu'un tag est partagé par plusieurs objets, il permet de représenter un groupe d'objets. La plupart des opérations graphiques disponibles sur un objet le sont également sur un tag afin que le développeur puisse modifier tous les objets du groupe par une simple instruction. Supposons que l'on ait ajouté le tag "menu" à chacun des objets graphiques qui constituent le marking menu de la figure 5.6. Pour rendre le menu visible à la position de la souris, le développeur a simplement à appliquer une translation au centre du menu (qui est parent de tous les autres objets graphiques) et rendre le tag "menu" visible :

```
menuCenter.translateTo(mouseX, mouseY);
canvas.getTag("menu").setDrawable(true);
```

Contrairement à Tk et GmlCanvas où les tags ne sont que des chaînes de caractères pour étiqueter un objet, les tags de SwingStates sont de véritables objets Java. SwingStates définit deux principales classes de tags : les tags *extensionnels* et les tags *intentionnels*. Les tags *extensionnels* sont ajoutés et retirés explicitement par le programmeur et peuvent être *actifs*. Les tags *intentionnels* sont définis par un prédicat sur un objet graphique de sorte que chaque objet graphique vérifiant ce prédicat est automatiquement étiqueté par ce tag. Par exemple, le programmeur peut utiliser un tag extensionnel actif pour programmer un mécanisme de sélection et son feed-back :

```
class SelectionTag extends CExtensionalTag {
   Stroke initialStroke = null;
   public SelectionTag() { super(); }
   public void added(CShape s) {
      initialStroke = s.getStroke();
      s.setStroke(new BasicStroke(initialStroke.getLineWidth()+1));
   }
   public void removed(CShape s) {
      s.setStroke(initialStroke);
   }
}
```

Les méthodes `added` et `removed` permettent de spécifier le comportement lors de l'ajout ou du retrait de ce tag. Ainsi, à chaque fois que le programmeur ajoutera ce tag à un objet en cas de sélection, l'épaisseur du contour de cet objet sera augmentée de 1 pixel et son épaisseur initiale sera restaurée lors du retrait de ce tag. Pour les tags intentionnels, les sous-classes surchargent la méthode `criterion` pour spécifier le prédicat d'appartenance. Par exemple, pour manipuler l'ensemble des formes graphiques qui se situent en haut de la fenêtre, le développeur peut utiliser le tag intentionnel ci-dessous. L'ensemble des formes étiquetées sera automatiquement mis à jour même en cas de changement de la géométrie.

```
class UpperTag extends CIntentionalTag {
   public UpperTag(Canvas c) { super(c); }
   public boolean criterion(CShape s) {
      return s.getMaxY() < 100;
   }
}
```

5.3.3 Les machines à états pour le canvas de SwingStates

La classe de machines à états `CStateMachine` a été spécialement conçue pour faciliter la programmation avec les formes d'un canvas SwingStates. Ces machines à états peuvent être attachées à une forme, un tag ou à la totalité du canvas. Outre les transitions correspondant aux événements d'entrée usuels, elles disposent de transitions contextuelles de la forme `*OnShape` et `*OnTag` qui permettent de filtrer les événements qui déclenchent la transition en fonction de la position à laquelle surviennent ces éléments. Par exemple, pour les appuis de bouton de souris, le programmeur peut utiliser une transition de type `Press` qui sera déclenchée par un appui de souris n'importe où, une transition de type `PressOnShape` qui ne sera déclenchée que si l'appui de bouton survient au-dessus d'une forme graphique ou encore une transition de type `PressOnTag` qui ne sera déclenchée que si l'appui de bouton survient au-dessus d'une forme graphique ayant un tag donné. Un autre exemple, pour les déplacements de souris, le programmeur peut facilement mettre en surbrillance la forme graphique qui se situe en dessous du curseur grâce aux transitions `EnterOnShape` ou `LeaveOnShape`. Le *picking* (c'est-à-dire la recherche des objets graphiques à une position donnée) est fait automatiquement par le canvas lorsque l'état courant contient des transitions de la forme `*OnShape` ou `*OnTag`. Le développeur peut également décider si une forme graphique participe ou non à l'interaction : en effet, il peut modifier l'attribut `pickable` d'une forme graphique s'il ne veut pas que celle-ci soit réactive à l'interaction. Par exemple, un item de menu est représenté par une `CPolyLine` pour son fond et un `CText` affiché au-dessus pour son texte. Si on désire que les items se mettent en surbrillance lorsque le curseur passe au-dessus d'eux, le programmeur peut définir le texte comme *non pickable* et programmer l'interaction en fonction du fond. Ainsi, bien qu'étant au-dessus du fond, le texte ne provoquera pas d'événements `EnterOnShape` ou `LeaveOnShape` afin d'éviter les clignotements dans la machine suivante (en supposant que chaque fond d'item s'est vu attribué le tag "hilitableItem") :

```
new CStateMachine(canvas) {
   public State start = new State() {
      Transition hilite = new EnterOnTag("hilitableItem") {
         public void action() {
            getShape().setFillPaint(HILITE_COLOR);
         }
      };
      Transition unhilite = new LeaveOnTag("hilitableItem") {
         public void action() {
            getShape().setFillPaint(BG_COLOR);
         }
      };
   };
};
```

SwingStates permet également de déclencher une transition lorsqu'un événement se produit sur un objet portant un tag d'une classe donnée. On peut alors factoriser le traitement de l'interaction pour un ensemble de formes tout en pouvant récupérer des informations spécifiques à cette forme. Dans l'exemple du marking menu, on souhaite pouvoir récupérer le nom de l'item sélectionné tout en factorisant l'interaction qui permet de mettre en surbrillance n'importe quel item de menu lorsque le curseur de la souris passe au-dessus. Or, nous venons de voir que c'est le fond de l'item avec lequel on interagit et non pas le texte de cet item. Nous définissons alors `HilitableItem`, une sous-classe de `CNamedTag` (la classe de tag extensionnel de SwingStates qui permet d'apposer une chaîne de caractères à un objet). Lors de la construction de chaque fond d'item, nous lui associons une instance de cette classe de tag dont la chaîne de caractères est le texte de l'item. Nous pouvons ainsi écrire une machine qui se charge de changer leur couleur de fond des items et de renvoyer le nom du dernier item visité lorsque l'utilisateur relâche le bouton droit de la souris :

```
class HilitableItem extends CNamedTag {
   public HilitableItem(String name) {
      super(name);
   }
}

new CStateMachine(canvas) {
   String labelItemSelected;
   public State menuOff = new State() {
      Transition invokeMenu = new Press(BUTTON3, ">> menuOn") {
         public void action() {
            showMenu();
         }
      };
   };
   public State menuOn = new State() {
      Transition hilite = new EnterOnTag(HilitableItem.class) {
         public void action() {
            getShape().setFillPaint(HILITE_COLOR);
            // Récupérer le nom de l'item
            labelItemSelected = ((HilitableItem)getTag()).getName();
         }
      };
      Transition unhilite = new LeaveOnTag(HilitableItem.class) {
         public void action() {
            getShape().setFillPaint(BG_COLOR);
         }
      };
      Transition select = new Release(BUTTON3, ">> menuOff") {
         public void action() {
            getShape().setFillPaint(BG_COLOR);
            System.out.println("item selected: "+ labelItemSelected);
         }
      };
   };
```

```
};
```

5.3.4 Les animations

Les animations permettent d'améliorer l'efficacité et la qualité graphique des interfaces. Une animation peut suggérer comment déclencher une commande (une animation du geste à effectuer dans une interface gestuelle par exemple), fournir un feedback sur l'état de l'application (une barre de progression lors du téléchargement d'un fichier par exemple), rendre les changements de l'interface plus compréhensibles [16], etc. SwingStates permet aux développeurs d'animer les objets graphiques d'un canvas en utilisant un mécanisme qui s'intègre aux machines à états.

Dans SwingStates, une animation est un objet qui peut être attaché à n'importe quel élément graphique d'un canvas (forme, tag ou canvas). Cet objet a un champ `t` dont la valeur évolue selon une fréquence régulière de 0 à 1 durant les *tours* pairs puis de 1 à 0 durant les tours impairs. Ce champ `t` est utilisé pour interpoler une valeur. Chaque animation est paramétrée par :

- son nombre de tours, *nbLaps*, et la durée d'un tour, *lapDuration*,
- sa fréquence, *delay*, et sa fonction de pas, *pacingFunction*, qui spécifient la durée d'un pas et la façon dont la paramètre `t` évolue d'un pas à l'autre (selon une fonction linéaire ou sigmoïde),
- sa durée totale, *duration*.

Le couple (*lapDuration*, *nbLaps*) et le champ *duration* sont liés donc, lorsqu'un de ces paramètres est modifié, les champs *lapDuration*, *nbLaps* et *duration* sont ajustés afin de conserver des valeurs cohérentes par rapport au dernier ajustement. Par défaut, *delay* vaut 40 millisecondes pour assurer une animation fluide de 25 images par seconde.

SwingStates contient un ensemble d'animations prédéfinies pour animer la géométrie ou les attributs de style des éléments d'un canvas. Chacune de ces animations existe en deux versions : *one-way* ou *loop*. Une animation *one-way* change continûment un attribut selon une dimension infinie alors qu'une animation *loop* change continûment un attribut de façon à ce qu'il fasse des allers-retours d'une borne à l'autre d'un intervalle. Par exemple, une animation `AnimationTranslateBy(dx, dy)` applique indéfiniment une translation (dx, dy) alors qu'une animation `AnimationTranslateTo(x, y)` fait évoluer un élément graphique entre sa position de départ et la position (x, y). Le développeur peut également définir ses propres animations en héritant de la classe `Animation` et en surchargeant la méthode `step(double t)` qui spécifie le comportement de l'animation. Une animation peut être démarrée, endormie, réveillée, arrêtée ou s'arrêter d'elle-même. Chaque fois que l'un de ces événements se produit, le développeur peut le traiter comme une transition de la machine à états.

Afin d'illustrer l'utilisation dans SwingStates, nous allons considérer l'exemple de la balle rebondissante. Celle-ci se déplace continûment dans le canvas et rebondit contre les bords du canvas afin de ne pas en sortir. Chaque fois que la balle touche un bord, elle est animée par un changement d'échelle donnant l'impression que la balle se déforme contre le bord avant de rebondir. La figure 5.7 illustre ce comportement. Le code complet correspondant peut être consulté en annexe 8.1, nous ne reportons ci-dessous qu'un extrait :

```
1  AnimationTranslateBy animBall = new AnimationTranslateBy(-10, 10);
```

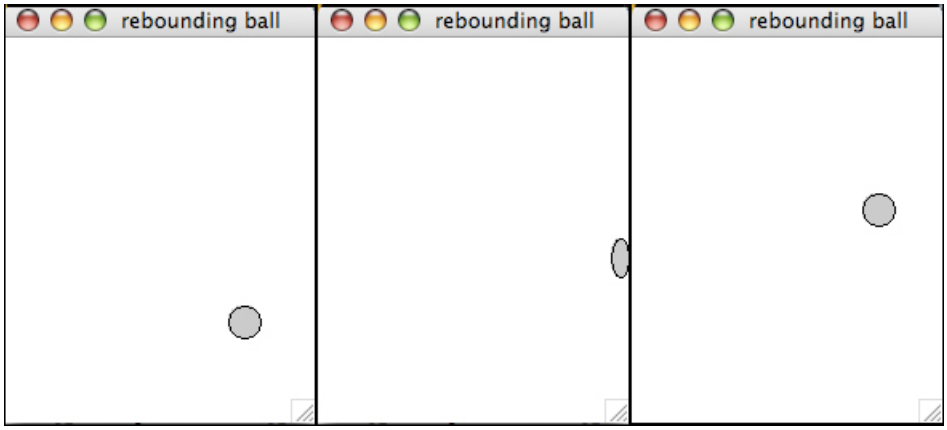


FIG. 5.7 : Une animation avec SwingStates

```
2  AnimationScaleTo animCollide = new AnimationScaleTo(0.3, 1.3);
3  CEllipse ball = canvas.newEllipse(50, 50, 20, 20);
4  animBall.setAnimatedElement(ball);
5  animCollide.setDurationLap(200).setNbLaps(2).setAnimatedElement(ball);

6  CStateMachine smBall = new CStateMachine(canvas) {
7     public State idle = new State() {
8        Transition collideY = new CElementEvent(ball, ">> collide") {
9           public boolean guard() {
10             // teste si l'ordonnée de la balle est dans la fenêtre
11             // renvoie vrai si ce n'est pas le cas
12          }
13          public void action() {
14             animBall.setDelta(-animBall.getDx(), animBall.getDy());
15             animBall.suspend();
16             animCollide.start();
17          }
18       };
         ...
19    };

20    public State collide = new State() {
21       Transition endCollide = new AnimationStopped(animCollide, ">> idle") {
22          public void action() {
23             animBall.resume();
24          }
25       };
26    };
27 };
```

Nous avons donc besoin de deux animations pour le comportement de la balle : (1) une animation de translation continue pour le déplacement de la balle et (2) une animation de changement d'échelle depuis l'échelle (1, 1) vers (1.3, 0.3) et depuis l'échelle (1.3, 0.3) vers (1, 1) (pour les collisions avec les bords horizontaux). Nous utilisons donc (1) une `AnimationTranslateBy` (l.1) et (2) une `AnimationScaleTo` (l.2) qui dure deux tours de 200 millisecondes (l.5). Les transitions de type `CElement(element, outputState)` sont déclenchées chaque fois qu'une modification est faite sur l'objet `element`. Ici, nous une utilisons une transition de ce type afin de suivre la balle (l.8) et de détecter si elle arrive sur un bord du canvas dans la garde de cette transition (l.10 et 11). Lorsque cette transition est déclenchée, l'animation de translation de la balle est endormie pour lancer l'animation de changement d'échelle (l.15 et l.16). L'animation de translation est réveillée (l.23) lorsque la transition `AnimationStopped` est déclenchée par la terminaison de l'animation de changement d'échelle (l.21).

Le mécanisme d'animations est non seulement intégré aux machines à états mais il est aussi intégré à la gestion des tags. En effet, les animations peuvent porter des tags afin de manipuler plusieurs animations simultanément. La figure 5.8 montre un exemple dans lequel nous avons utilisé quatre animations ayant des directions de translation opposées pour représenter l'explosion d'une brique lorsqu'elle est touchée par la balle. Nous ajoutons un même tag à ces animations et appelons la méthode `start` sur ce tag pour faire démarrer les quatre animations au même moment.

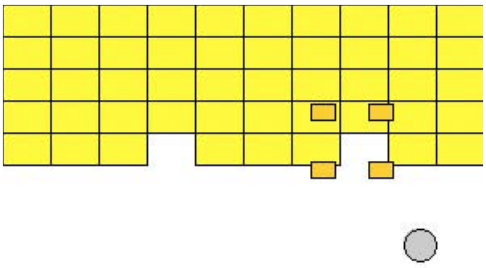


FIG. 5.8 : Utiliser un tag pour manipuler plusieurs animations simultanémanent

Les animations des applications graphiques font l'objet de nombreuses recherches ([53], [168], [170]) et elles sont intégrées aux boîtes à outils en Java pour l'interaction avancée que nous avons mentionnées au début de ce chapitre (SubArctic [93], ZVTM [140] et Piccolo [28]). Ici, nous recherchions un modèle simple qui puisse être intégré à la logique de SwingStates. Le modèle choisi est semblable à celui de ces boîtes à outils mais les animations génèrent des événements qui peuvent déclencher des transitions dans une machine à états afin de détecter facilement la fin d'une animation par exemple pour déclencher un autre traitement. Les animations peuvent également être étiquetées par des *tags* pour les grouper. Dans les autres boîtes à outils, la façon d'enchaîner différents traitements se fait soit en ajustant les dates de départ des animations soit en surchargeant la méthode `stop` d'une animation qui est appelée lorsqu'un

animation se termine.

5.4 Combiner les machines à états

SwingStates permet de combiner l'exécution de plusieurs machines à états. Elles peuvent être combinées *en séquence* ou *en parallèle*. Combiner les machines en séquence permet la modularité : une machine peut traiter des événements de bas niveau pour fabriquer et envoyer des événements de haut niveau qui seront consommés par une seconde machine. Combiner les machines en parallèle permet de pallier le problème de l'explosion du nombre d'états afin d'obtenir des programmes concis. Dans cette section, nous abordons au travers de deux exemples les avantages que confèrent les constructions *en séquence* et *en parallèle*.

5.4.1 Combiner en séquence pour communiquer

Se restreindre aux événements standard du clavier et de la souris est insuffisant pour développer des interfaces avancées. L'utilisation de la multimodalité requiert, par exemple, le traitement d'événements gestuels ou vocaux. Ici, nous présentons comment utiliser deux machines à états qui communiquent afin de programmer une interface gestuelle pour une application graphique gérant des commandes pour copier, couper et coller. Le fait d'utiliser deux machines à états permet d'obtenir un programme modulaire et plus facile à réutiliser dans le cas où le développeur cherche à tester une autre modalité comme la voix.

La classe `VirtualEvent` permet de définir des événements de haut niveau. Ces événements virtuels peuvent être générés par une machine grâce à la méthode `fireEvent` pour déclencher des transitions génériques de la classe `Event`. Sur notre exemple d'une application graphique qui permet de copier, couper et coller, la machine suivante permet d'entendre des événements de haut niveau portant le nom de la commande pour exécuter la bonne commande :

```
smApplication = new StateMachine() {
   public State start = new State() {
      Transition copy = new Event("copy"){...};
      Transition cut = new Event("cut"){...};
      Transition paste = new Event("paste"){...};
   };
};
```

Ainsi, notre application ne se préoccupe pas des périphériques d'entrée utilisés. Dans notre exemple d'interface gestuelle, nous définissons une deuxième machine qui se charge de dessiner le geste et qui utilise un objet `Classifier` implémentant l'algorithme de reconnaissance de geste de Rubine [156] (l. 7 et 27). La machine `smInk` dessine l'encre du geste et ajoute les points un par un à un objet `Gesture` (l. 13, 21 et 26). Lorsque l'utilisateur relâche le bouton de la souris, elle classifie le geste et génère un événement virtuel du nom de la classe reconnue (l. 30).

Dans cet exemple, la machine `smApplication` s'abonne à la machine `smInk` (l. 37) afin d'entendre les événements de haut niveau générée par celle-ci. En effet, toute machine peut être écoutée selon le paradigme usuel des écouteurs d'événements de Swing : tout objet implémentant l'interface `StateMachineListener` peut traiter les événements générés par une machine à états en utilisant le

callback `eventOccured`. Une machine à états implémente elle-même l'interface `StateMachineListener` et son callback consiste à traiter l'événement dans cette machine pour y déclencher une transition si nécessaire. SwingStates donne également la possibilité de *diffuser* un événement dans un canvas afin qu'il puisse être traité par toutes les machines attachées à ce canvas.

```
1  smInk = new CStateMachine(canvas) {
2     // l'encre du geste
3     CPolyLine ink = (SMPolyLine) canvas.newPolyLine(0, 0)
         .setPickable(false).setFilled(false);
4     // le geste
5     Gesture gesture = new Gesture();
6     // le classifieur pour reconnaître les gestes
7     Classifier classifier = Classifier.newClassifier("cutcopypaste.cl");
8     public State start = new State() {
9        Transition begin = new Press(BUTTON1, ">> draw"){
10            public void action(){
11             ink.reset(getPoint()).setDrawable(true).beforeAll();
12             gesture.reset();
13             gesture.addPoint(getPoint());
14          }
15       };
16    };
17    public State draw = new State() {
18       Transition draw = new Drag(BUTTON1){
19          public void action(){
20             ink.lineTo(getPoint());
21             gesture.addPoint(getPoint());
22          }
23       };
24       Transition end = new Release(BUTTON1){
25          public void action(){
26             gesture.addPoint(getPoint());
27             GestureClass gc = classifier.classify(gesture);
28              // envoyer un événement virtuel portant
29              // le nom du geste reconnu à smApplication
30             fireEvent(new VirtualEvent(gc.getName()));
31             ink.setDrawable(false);
32           }
33        };
34     };
35 };
36
37 smInk.addStateMachineListener(smGesture);
```

La figure 5.9 illustre l'aspect modulaire que permet cette construction pour faciliter la réutilisation de la machine à états de l'application en remplaçant simplement la machine gérant les événements de bas niveau issus des périphériques d'entrée. Ainsi, dans notre exemple, la machine `smApplication` peut s'abonner à une machine de plus bas niveau traitant les entrées gestuelles ou les entrées vocales ou aux deux machines pour gérer différents types d'entrée au sein de la même interface.

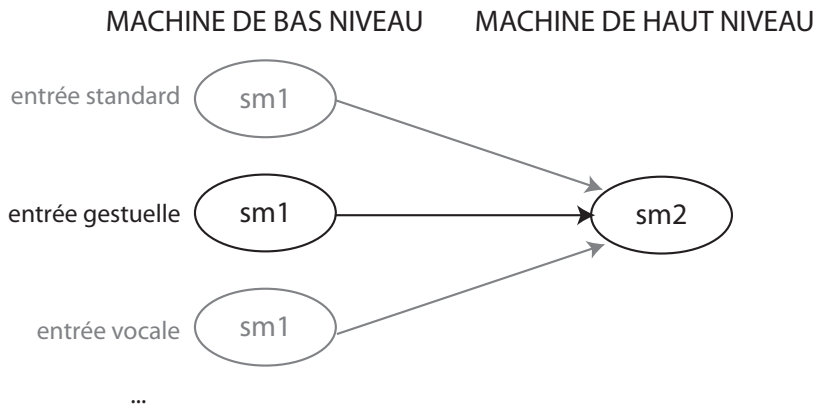


FIG. 5.9 : Construction en séquence

5.4.2 Combiner en parallèle pour factoriser

La possibilité d'exécuter plusieurs machines à états en parallèle permet de pallier le problème de l'explosion du nombre d'états lors de la description de l'interaction en utilisant des machines à états. Supposons qu'un menu linéaire et qu'un marking menu cohabitent au sein d'une même interface, chacun aura sa propre machine pour gérer son interaction. Cependant, ils partagent tous deux le comportement qui consiste à mettre en surbrillance un item lorsque le curseur de la souris se trouve au-dessus de cet item. Pour ce faire, il suffit de faire tourner indépendamment en parallèle une machine qui gère la mise en surbrillance afin que les machines des deux autres menus puissent ignorer cet aspect de l'interaction (au lieu d'avoir à inclure ce comportement dans chacun des états concernés). Le comportement qui aurait dû être décrit dans les deux machines est ainsi factorisé en une seule machine comme l'illustre la figure 5.10.

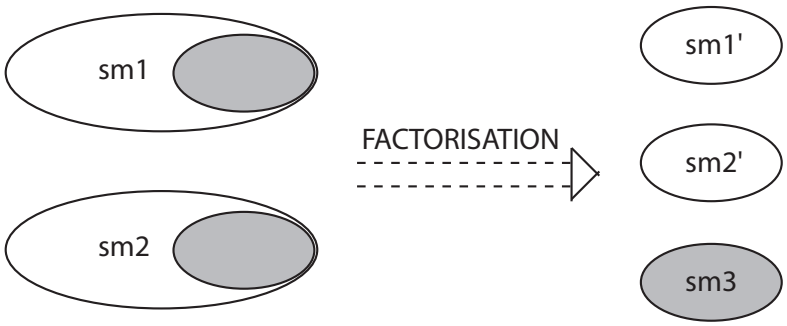


FIG. 5.10 : Construction en parallèle

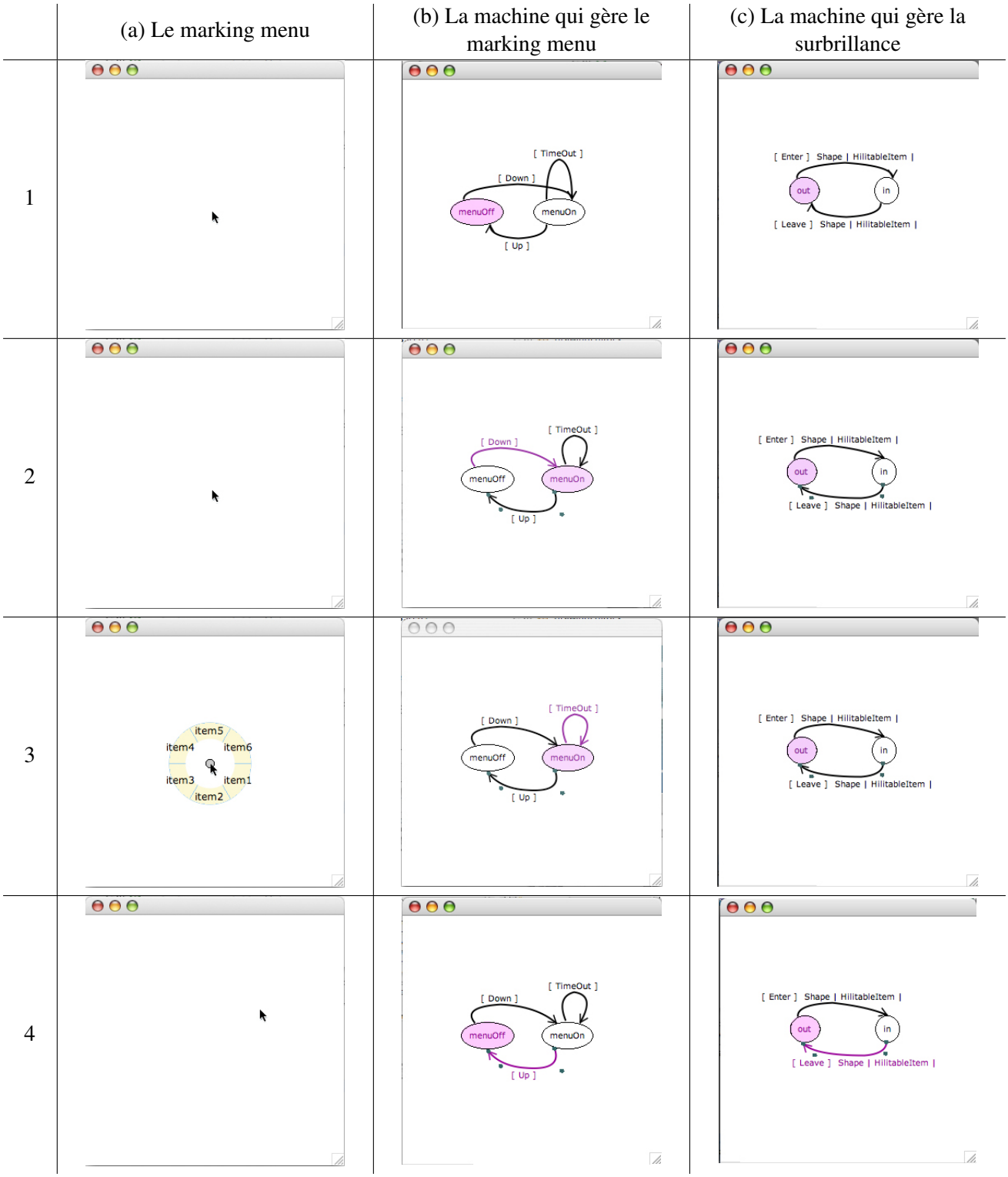


FIG. 5.11 : Visualiser les machines à états du marking menu

La figure 5.11 montre les représentations graphiques de la machine à états spécifique au marking menu (au milieu) et de celle gérant la surbrillance (à droite) qui tournent en parallèle. Ces représentations graphiques ont été réalisées à partir de captures d'écran de notre outil de visualisation de machines à états. Cet outil est utile pour aider à la mise au point d'une machine à états. En effet, un objet de la classe `StateMachineVisualization` prend une machine en paramètre et construit un widget (grâce au canvas de SwingStates) qui affiche cette machine et son état en temps réel. À chaque instant, l'état courant et la dernière transition sont colorés. Fournir automatiquement une bonne mise en page d'un graphe dirigé est un problème très complexe et nous laissons donc à l'utilisateur le soin d'ajuster manuellement la mise en page qui est fournie par défaut (voir la cellule 1-(b) de la figure 5.11). L'utilisateur peut *panner* en utilisant un *drag* dans le fond du canvas pour recentrer la machine par exemple, déplacer les nœuds par *drag* (ses transitions restant accrochées) ou déplacer les transitions grâce à des poignées de manipulation sur leurs points de contrôle. Par exemple, sur la figure 2-(b), l'utilisateur a replacé les transitions afin de mieux distinguer les différentes transitions.

Les opérations de mise en séquence ou en parallèle permettent d'obtenir des programmes modulaires et concis. Nous n'avons cependant pas exploré la relation hiérarchique proposée par Blanch [31] avec laquelle un état peut contenir une machine. Cette relation permet principalement d'automatiser l'activation et la désactivation de plusieurs machines, une machine n'étant active que lorsque son état parent est lui-même actif. En pratique, nous n'avons jamais rencontré de cas dans lequel cette relation aurait été nécessaire. En effet, dans SwingStates, une machine peut être activée et désactivée. Afin d'obtenir le même résultat que la relation hiérarchique, il suffit de placer ces instructions d'activation et de désactivation dans les actions d'entrée et de sortie d'une autre machine comme l'illustre la figure 5.12.

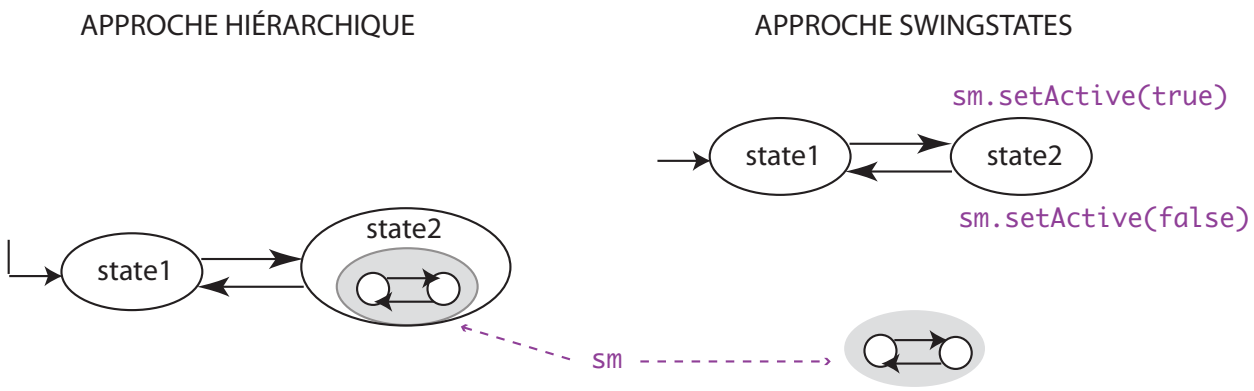


FIG. 5.12 : Approche hiérarchique vs. Approche SwingStates

5.5 Redéfinir l'interaction de composants graphiques existants

Alors que certaines techniques d'interaction requièrent la programmation de nouveaux widgets, certaines peuvent être obtenues en modifiant l'interaction des widgets existants. La classe `JStateMachine`

permet de redéfinir ou d'augmenter l'interaction avec les widgets prédéfinis de Swing.

La classe `CrossingInteraction` ci-dessous permet d'utiliser le franchissement pour activer les widgets Swing d'une classe donnée. Comme dans le cas des formes graphiques d'un canvas, SwingStates permet d'attacher des tags aux widgets Swing. Par défaut, chaque widget porte un tag du nom de sa classe (l.8 et 12). Ainsi, la classe `CrossingInteraction` est générique, paramétrée par le nom de la classe de widgets à rendre franchissable (bouton `javax.swing.JButton`, case à cocher `javax.swing.JCheckBox`, etc.). La méthode `invoke` (l. 22) est surchargée pour spécifier l'effet d'un franchissement sur un widget (appuyer dans le cas d'un bouton, sélectionner/désélectionner dans le cas d'une case à cocher, etc.). Pour rendre les widgets contenus dans un panneau Swing, nous lui attachons la machine à états `crossSM` (l. 20) qui détecte les événements de franchissement. Lorsque l'utilisateur presse le bouton de la souris (l.5), il entre dans le mode franchissement dans lequel les événements d'entrée et de sortie sur un widget sont écoutés. Il quitte ce mode en relâchant le bouton de la souris (l.17).

```
1  public abstract class CrossingInteraction {

2     public CrossingInteraction(JPanel panel, String className) {
3        JStateMachine crossSM = new JStateMachine() {
4           public State crossOff = new State() {
5              Transition press = new Press(BUTTON1, ">> out") { };
6           };

7           public State out = new State() {
8              Transition enter = new EnterOnTag(className, ">> in") { };
9              Transition release = new Release(BUTTON1, ">> crossOff") { };
10          };

11          public State in = new State() {
12             Transition leave = new LeaveOnTag(className, ">> out") {
13                public void action() {
14                   invoke(getComponent()); // activation du widget
15                }
16             };
17             Transition release = new Release(BUTTON1, ">> crossOff") { };
18          };

19       };
20       crossSM.attachTo(panel);
21    }

22    abstract void invoke(Component c) ;

23 }
```

Ainsi, les quelques lignes ci-dessous affecte l'interaction de franchissement à toutes les cases à cocher du panneau `cpane`. Il est à noter que l'on peut modifier plusieurs cases d'un geste continu [22] (Figure 5.13) :

```
new CrossingInteraction(cpane, "javax.swing.JCheckBox") {
   void invoke(Component c) {
      ((JCheckBox)c).setSelected(!((JCheckBox)c).isSelected());
   }
};
```

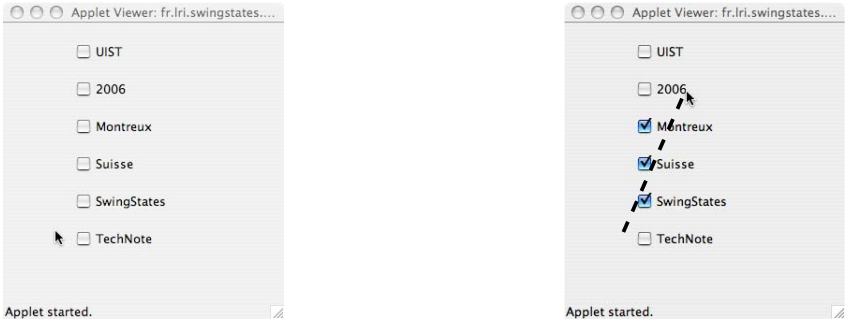


FIG. 5.13 : Des cases à cocher franchissables

Un problème avec l'implémentation ci-dessus est l'absence d'une trace d'encre laissée par le curseur lorsqu'il est en mode franchissement (elle a été ajoutée à la main sur la figure 5.13). SwingStates permet d'attacher une machine à états au *glasspane* de Swing, un panneau transparent affiché au-dessus des autres widgets dans une fenêtre. Il est donc possible d'y dessiner l'encre quand le bouton de la souris est enfoncé et de l'effacer lorsque celui-ci est relâché. La figure 5.14 montre un autre exemple qui utilise le *glasspane* pour entrer une valeur numérique dans un champ texte à l'aide d'une interaction qui rappelle celle que l'on aurait avec un joystick. Lorsque le curseur est en-dessous du champ, la valeur est décrémentée alors que lorsqu'il est au-dessus, la valeur est incrémentée. Plus la distance entre le point de départ et le point courant est grande, plus la valeur change rapidement.

FIG. 5.14 : Une entrée numérique augmentée d'une "interaction joystick"

Il est en bien entendu possible d'ajouter un canvas de SwingStates dans le *glasspane* afin d'y dessiner des scènes interactives plus complexes comme dans l'exemple de la figure 5.15. Le canvas contient un *pie menu* géré par une machine à états qui envoie des événements contenant la couleur choisie à la machine à états qui se charge de faire le *picking* des widgets dans le panneau du dessous et de leur appliquer la couleur reçue.

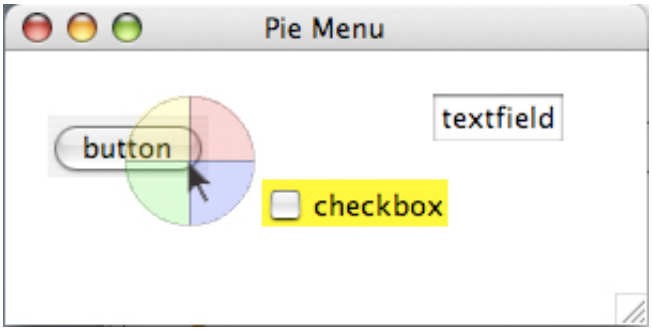


FIG. 5.15 : Un *pie menu* pour colorer des widgets Swing

La possibilité de redéfinir l'interaction des widgets existants permet donc d'explorer de nouvelles techniques avec un effort de programmation réduit. Les trois exemples de cette section sont disponibles sous forme d'applets à l'adresse http ://www.lri.fr/∼appert/SwingStates.

5.6 Évaluer une boîte à outils, un problème complexe

Évaluer une boîte à outils est un problème bien connu mais pour lequel il n'existe pas de solution satisfaisante. Les concepteurs de boîtes à outils rapportent généralement des informations quant à la taille de leur boîte à outils. Le code source de SwingStates contient 9226 lignes de code (sans les commentaires) et la librairie jar exécutable pèse 260 Ko. Bien évidemment, ces informations ne sont pas une mesure de son efficacité et encore moins de son utilisabilité. Les concepteurs rapportent aussi le nombre d'utilisateurs de leur boîte à outils lorsque ceci est possible. Cependant, pour cela, la boîte à outils doit déjà avoir une grande visibilité et les concepteurs doivent collecter un grand nombre de témoignages d'utilisation avant d'obtenir des résultats valides. Ceci étant une tâche considérable, les concepteurs se contentent généralement de rapporter le nombre de fois où leur boîte à outils a été téléchargée, ce qui ne constitue pas une mesure scientifique acceptable. Dans cette section, nous présentons notre processus d'évaluation de SwingStates. En premier lieu, nous montrons que SwingStates a été programmé en suivant certains principes de conception, puis en second lieu, nous rapportons les résultats de son utilisation par nos étudiants sur un benchmark de techniques d'interaction.

5.6.1 Les principes de conception

Notre premier objectif était de satisfaire la fameuse phrase d'Alan Kay : "Simple things should be simple, complex things should be possible". Afin d'atteindre ce but, nous avons utilisé deux principes qui ont déjà fait leurs preuves dans la conception d'interfaces : le polymorphisme et la réification [27]. La réification consiste à transformer des concepts abstraits en objets concrets alors que le polymorphisme consiste à concevoir des fonctions ou objets qui peuvent être utilisés dans des contextes différents. Les langages à objets comme Java sont un cadre de travail particulièrement bien adapté pour satisfaire ces principes puisque la conception consiste à identifier les classes pertinentes et que les langages à objets en général et Java en particulier offrent des mécanismes de réification (les classes, l'introspection) et de

polymorphisme (héritage, inerfaces).

Comme nous l'avons mentionné à la section 5.3.1, les formes graphiques sont des objets à part entière contrairement à l'approche de Java2D qui incite à concevoir une scène graphique en termes de formes graphiques dessinées relativement à un contexte graphique (`Graphics2D`). Dans SwingStates, un objet `CShape` connaît non seulement ses coordonnées relatives et absolues mais aussi ses attributs géométriques et de style.

D'autre part, les tags sont également des objets et offrent un plus grand pouvoir d'expression que les simples chaînes de caractères de Tk. Ils peuvent avoir leur propre comportement (tags actifs) et ont leurs propres méthodes pour les manipuler. Par application du polymorphisme, la plupart des méthodes applicables à une forme graphique sont également applicables à un tag en appliquant la méthode à chaque objet *taggés*. Ces mêmes méthodes sont également applicables à un canvas afin de pouvoir manipuler l'ensemble des formes graphiques d'un canvas comme un simple groupe. Une autre caractéristique intéressante de ces méthodes est qu'elles retournent toujours l'instance appelante. Ainsi, leur résultats peuvent être directement utilisés dans d'autres contextes et les programmes restent concis. Par exemple, les cinq lignes suivantes :

```
CEllipse e = new CEllipse(10,10,20,30);
e.rotateTo(Math.PI/4);
e.setFillPaint(Color.red);
e.setOutlined(false);
e.addTo(canvas);
```

peuvent être remplacées par l'unique instruction suivante (Seules les méthodes `new*` du canvas agissent comme des constructeurs et renvoient l'objet construit et non pas le canvas lui-même) :

```
CEllipse e = (CEllipse)canvas.newEllipse(10,10,20,30)
   .rotateTo(Math.PI/4).setFillPaint(Color.red).setOutlined(false);
```

À la manière des *Interactors* [125] ou des *dispatch agents* de SubArctic [93], SwingStates externalise la gestion de l'interaction dans des objets séparés. L'interaction est réifiée en termes d'objets de contrôle : les machines à états qui contiennent des états contenant eux-mêmes des transitions. Ces classes et leur organisation basée sur les classes internes réifient le formalisme des machines à états en Java. Au-delà de la syntaxe "naturelle" que permet les classes internes, les instances des classes internes présentent l'avantage de pouvoir accéder aux champs et méthodes de l'objet dans lesquelles elles sont déclarées.

5.6.2 L'approche par benchmarks

L'utilisation d'un benchmark est un processus d'évaluation souvent utilisé en informatique en général et aussi dans certains domaines de l'IHM comme la visualisation d'informations [144]. Ici, nous proposons d'appliquer ce type d'évaluation aux boîtes à outils graphiques et reportons les résultats de nos expériences avec des étudiants de master en informatique. Depuis plusieurs années, l'IHM est enseigné aux étudiants de 1ere et 2eme année de master à l'Université Paris-Sud. En 2ème année, ils disposent de deux mois pour lire un article de recherche décrivant une technique et implémenter cette technique en utilisant la boîte à outils de leur choix. Pour la première fois, en 2005, ils avaient la contrainte supplémentaire d'utiliser une version précédente de SwingStates [11]. Le cours incluait une heure de présentation

de la boîte à outils et les étudiants pouvaient consulter la documentation Java et un site wiki qui contient quelques exemples simples de programmes réalisés avec SwingStates[2]. 14 groupes de 2 étudiants ont implémenté les huit techniques d'interaction listées dans la table de la figure 5.16.

Marking Menu [105]	2 groupes
Flow Menu [84]	1 groupe
Reconnaissance de gestes [108]	2 groupes
Side Views [169]	2 groups
Local tools [29]	2 groupes
Alignment stick [149]	2 groupes
Pushpins, rulers and pens [8]	2 groupes
Magnetic guidelines [25]	1 groupe

FIG. 5.16 : Benchmark de techniques utilisé

Contrairement aux résultats décevants obtenus les années précédentes lorsque les étudiants étaient libres d'utiliser la boîte à outils de leur choix, tous les étudiants ont réalisé des prototypes fonctionnels avec peu ou pas d'aide. Ces résultats démontrent le pouvoir et la simplicité du canvas et des machines à états pour implémenter des techniques d'interaction avancées. La figure 5.17 montre trois exemples des projets que les étudiants ont réalisé : *alignment stick*, *local tools* et *magnetic guidelines*.

En examinant les codes source, nous avons observé que tous les étudiants avaient compris les concepts et les possibilités de SwingStates. Chaque groupe n'a utilisé qu'une seule machine à l'exception d'un groupe qui a utilisé une machine pour les règles et une machine pour les stylos dans le projet *pushpins, rulers and pens*. Les projets comportaient 750 lignes de code en moyenne et les machines elles-même représentaient 200 lignes de code en moyenne. Les machines contenaient de 2 à 9 états et les états contenaient de 8 à 32 transitions. Seul un groupe (*SideViews*) a étendu la classe `CShape` afin qu'un objet graphique puisse gérer un historique de ses transformations. Tous les autres groupes ont réalisé leurs scènes à partir des classes d'objets graphiques prédéfinies dans SwingStates. Tous les groupes ont utilisé les tags pour grouper des objets (les outils d'une palette par exemple) et pour gérer l'interaction (pour distinguer un outil d'un objet dans les transitions par exemple). Cependant, certains groupes n'ont pas bien compris l'utilisation des transitions de la forme `*OnTag` et ont utilisé des transitions de la forme `*OnShape` renforcées par des gardes vérifiant les tags portés par la forme graphique. Il est important de préciser que, dans cette première version, seuls les tags extensionnels étaient disponibles et que les tags intentionnels auraient répondu à certains besoins des étudiants. Par exemple, un groupe a rapporté la difficulté de manipuler une forme et ses descendants comme un groupe alors que le tag intentionnel ci-dessous aurait permis de le faire facilement (ce tag est désormais une classe prédéfinie de SwingStates) :

[2]http ://wiki.lri.fr/SMCanvas/ et http ://wiki.lri.fr/SwingStates/

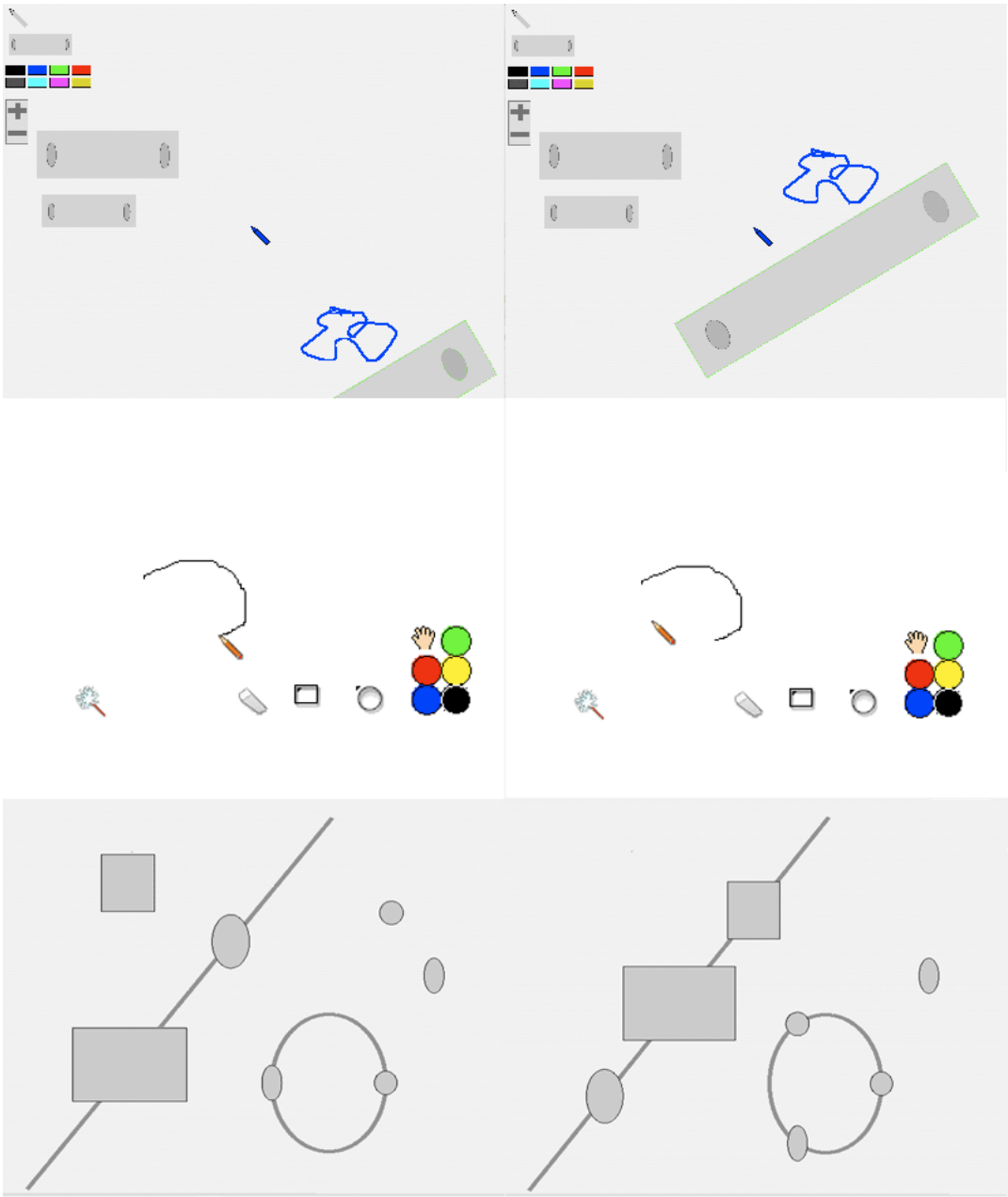


FIG. 5.17 : Exemples de projets réalisés par les étudiants : *alignment stick* (en haut), *local tools* (au milieu), *magnetic guidelines* (en bas)

```
public class Hierarchy extends CIntentionalTag {
   CShape topShape;

   public CHierarchyTag(CShape s) {
      super(s.getCanvas());
      topShape = s;
   }

   public boolean criterion(CShape s) {
      if(topShape == s) return true;
      CShape tmp = s;
      while(tmp.getParent() != null) {
         tmp = tmp.getParent();
         if(tmp == topShape) return true;
      }
      return false;
   }
}
```

Cette première évaluation a montré que les étudiants peuvent implémenter des techniques d'interaction avancées en utilisant SwingStates et donc qu'au moins certaines choses complexes sont possibles ("complex things should be possible"). Elle a également révélé que des étudiants ayant une expérience relativement limitée du langage Java (certains n'en ayant aucune) ont adopté SwingStates sans difficulté.

Ces résultats satisfaisants nous ont encouragé à utiliser également SwingStates dans le cours d'introduction aux systèmes interactifs du master 1ere année. Les concepts des systèmes interactifs étaient auparavant illustrés en travaux dirigés avec Tcl/Tk [136]. Cependant, le temps de maîtriser un langage de script comme Tcl et le temps imparti pour l'enseignement ne permettait pas de demander aux étudiants d'implémenter des techniques d'interaction avancées et contraignait à rester dans un ensemble de techniques standard. Pour la première fois en 2005, nous avons utilisé une version précédente de SwingStates dans une bonne partie des travaux dirigés, par exemple pour implémenter et comparer différents types de menus. Les étudiants ont également réalisé un widget permettant de faire des requêtes dynamiques [7]. Leurs prototypes étaient fonctionnels et les étudiants ont tous témoigné de la facilité d'utilisation de SwingStates. Pour cette seconde expérience, nous tenons à préciser que les étudiants étaient encadrés par un enseignant qui n'avait pas participé au développement de SwingStates.

5.7 Conclusion

Dans cette section, nous avons présenté SwingStates[3], une extension de la boîte à outils Java Swing qui propose un canvas fondé sur un modèle graphique à base de structure d'affichage pour créer de nouveaux widgets et un modèle d'interaction à base de machines à états afin de programmer l'interaction de nouveaux widgets ou compléter/redéfinir celle de widgets existants. SwingStates met en œuvre les principes de polymorphisme et de réification et exploite les caractéristiques du langage Java pour développer des interactions post-WIMP de façon simple et concise. Les programmes qui reposent sur SwingStates

[3]http ://www.lri.fr/∼appert/SwingStates, https ://sourceforge.net/projects/swingstates

peuvent contenir plusieurs machines qui tournent en parallèle ou qui communiquent entre elles. Ce style de programmation permet de maîtriser le problème de l'explosion du nombre d'états et de garder ainsi des programmes concis et lisibles. SwingStates a été évalué avec des étudiants de Master sur un ensemble de techniques d'interaction avancées, confirmant sa facilité de prise en main et sa puissance d'expression. SwingStates permet de réduire l'effort de programmation nécessaire pour implémenter des techniques originales afin de favoriser l'exploration de l'espace de conception dans les stades avancés de conception des interfaces graphiques en prototypant différentes possibilités.

Mener des expérimentations contrôlées avec TouchStone

Lors de l'introduction d'une nouvelle technique dans la littérature, les chercheurs en IHM évaluent la nouvelle technique implémentée dans le cadre d'une expérimentation contrôlée. Nous avons souligné à quel point il est difficile pour le concepteur d'interfaces de consulter les résultats fournis dans la littérature. En effet, une expérimentation ne pouvant tester toutes les conditions possibles, les chercheurs sont contraints de faire des choix pour ne tester qu'un échantillon des conditions envisageables. Les connaissances requises, le temps nécessaire et l'environnement expérimental sont autant de contraintes auxquelles se heurtent les évaluateurs.

Cet ensemble d'obstacles rencontrés dans la réalisation d'expérimentations contrôlées favorise des plans expérimentaux simples et ad hoc ne comparant la nouvelle technique qu'à la technique "standard" sur une tâche spécifique. Le concepteur d'interfaces devant faire des choix, il est amené à essayer de tirer des conclusions plus générales en regroupant les différents résultats collectés dans la littérature. Les conditions expérimentales variant d'une expérimentation à l'autre, les résultats et surtout les conditions expérimentales étant bien souvent partiellement rapportés, le concepteur d'interfaces se trouve dans l'impossibilité d'unifier les différents résultats. Par exemple, il est impossible de tirer une conclusion à partir de deux expérimentations qui n'ont pas de condition commune ou, si elles ont une condition commune, il est dangereux d'ignorer les détails d'implémentation qui peuvent varier et avoir des conséquences importantes sur les performances.

Dans ce chapitre, nous présentons TouchStone [115], une plateforme pour aider les chercheurs en IHM à mener des expérimentations contrôlées et faciliter l'exploitation de leurs résultats pour un public plus large afin d'augmenter le nombre, la qualité et la lisibilité des résultats rapportés dans la littérature. TouchStone assiste les chercheurs dans les différentes activités que comporte une expérimentation contrôlée : la conception du plan expérimental, l'exécution de l'expérimentation puis l'analyse des résultats. La plateforme de conception guide l'évaluateur étape par étape en permettant une approche exploratoire. La plateforme d'exécution, basée sur une architecture modulaire, facilite le développement de nouveaux composants et la réutilisation de composants existants alors que la plateforme d'analyse fournit des aides pour utiliser les différentes méthodes d'analyse statistique. Afin d'augmenter la lisibilité et la réutilisabilité des expérimentations déjà réalisées, TouchStone agit également comme une base de données des différents éléments d'une expérimentation dans un format standard : le plan expérimental, le programme capable d'exécuter ce plan et les résultats collectés durant l'expérimentation.

6.1 L'architecture de TouchStone

La réalisation d'expérimentations intervient à tous les stades de la recherche scientifique. D'un côté, à partir des théories, les chercheurs opérationnalisent des hypothèses et réalisent des expérimentations contrôlées pour déterminer si ces hypothèses sont supportées en mesurant des effets ou des corrélations par exemple. D'un autre côté, ces expérimentations sont utilisées par d'autres chercheurs pour être répliquées dans des conditions identiques ou différentes afin de les confirmer, les réfuter ou les étendre. Ces expérimentations peuvent alors donner naissance à de nouveaux modèles empiriques, voire de nouvelles théories. Pour le chercheur, l'expérimentation est à la fois un outil de validation et un outil de recherche exploratoire.

Dans les domaines scientifiques empiriques ayant une plus longue histoire que l'IHM, comme la biologie ou la physique, la réalisation d'expérimentations contrôlées est un processus incontournable pour lequel les étudiants sont formés, les pratiques sont standardisées, etc. Quelle que soit la portée et la robustesse des résultats rapportés, une expérimentation n'est jamais une recherche définitive et les chercheurs sont très souvent amenés à baser leurs travaux et propres expérimentations sur les expérimentations des autres chercheurs. L'IHM étant un domaine encore jeune, MacKenzie et al. [117] montrent que, malgré l'abondance des publications en IHM et dans le domaine des facteurs humains, les méthodologies sont ad hoc et que les différentes procédures expérimentales sont bien souvent inconsistantes entre elles, ce qui diminue grandement la possibilité de comparer plusieurs résultats pour les généraliser.

TouchStone est un outil pour mener des expérimentations contrôlées plus facilement et de manière plus automatique et standardisée. TouchStone comprend trois plateformes pour couvrir les trois principales parties que comporte la réalisation d'une expérimentation contrôlée : une plateforme de conception, une plateforme d'exécution et une plateforme d'analyse (Figure 6.1).

La plateforme de conception est une application internet qui découpe la construction d'un plan expérimental en différentes étapes structurées. Lors de ces étapes (spécification des facteurs, de la stratégie de contrebalancement, estimation du temps nécessaire, etc.), l'évaluateur peut tester plusieurs alternatives afin de faire le choix qui lui convient le mieux. Afin d'encourager l'approche de conception exploratoire, l'application permet également à l'évaluateur de revenir sur n'importe quelle étape à chaque instant. La plateforme de conception permet de produire un fichier XML qui décrit l'expérimentation en termes d'instructions séquentielles "exécutables" tout en étant lisibles par un humain. Ce script définit l'enchaînement des différentes conditions et référence les composants logiciels nécessaires à l'exécution de l'expérimentation. La plateforme d'exécution est constituée d'une application Java capable d'interpréter ce script pour l'exécuter et d'un environnement de programmation qui permet de construire des nouveaux composants logiciels. L'application enregistre les valeurs des facteurs et des mesures pour produire des fichiers de logs destinés à être analysés avec la plateforme d'analyse. Pour l'instant, la plateforme d'analyse est un site internet fournissant quelques conseils sur les analyses possibles et des liens vers deux logiciels d'analyse couramment utilisés : JMP [157], un logiciel de SAS Institute, et R [148], un package open-source d'analyse statistique. Bien que développées pour fonctionner ensemble, les trois plateformes peuvent être utilisées indépendamment les unes des autres. Par exemple, l'évaluateur peut éditer le script XML exécutable à la main et se passer de la plateforme de conception ou n'utiliser que la plateforme de conception pour construire son plan expérimental qu'il implémentera dans l'environnement de son choix. TouchStone est un outil *open source* développé par plusieurs membres de l'équipe in|situ| et est encore l'objet de recherches. Ce chapitre donne une vue générale de l'outil en se focalisant sur ce qui a fait l'objet de mes recherches personnelles : la plateforme d'exécution et ses liens avec les deux autres plateformes.

6.2 Concevoir des expérimentations contrôlées

La plateforme de conception de TouchStone [1] est un outil pour concevoir des plans expérimentaux en suivant une approche exploratoire. L'évaluateur utilise l'interface de la plateforme de conception pour

[1] http ://touchstone.lri.fr

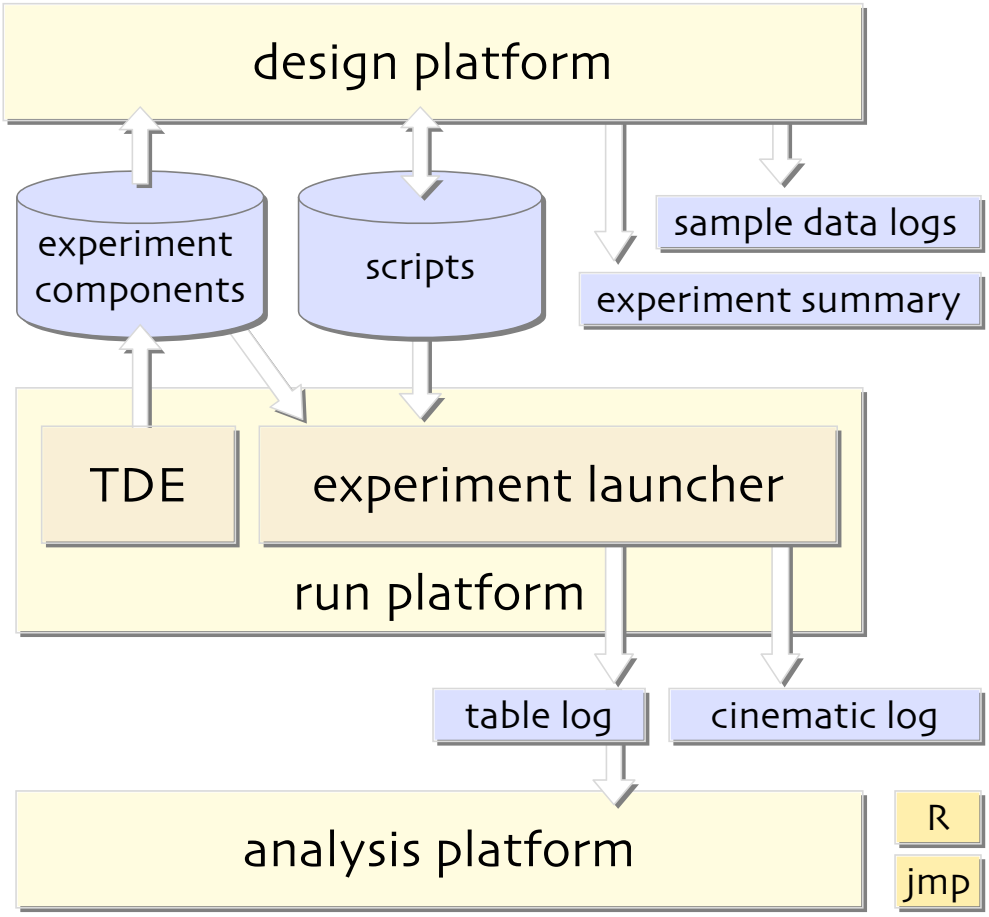


FIG. 6.1 : Architecture de TouchStone

réaliser son plan expérimental étape par étape. À n'importe quelle étape, cette interface permet à l'évaluateur d'envisager différentes alternatives avant de prendre des décisions. Il peut revenir à tout moment sur chaque étape et concevoir son expérimentation en répondant aux différents "Et si... ?" : "Et si j'ajoutais une valeur supplémentaire à ce facteur, combien de participants faut-il ?", "Et si je considérais ce facteur comme un facteur secondaire et que je ne le présentais qu'à certains participants ?", "Et si j'ajoutais une réplication par participant et par condition, l'expérience serait-elle trop longue ?", etc. À chaque instant, l'interface propose une aide explicative et des liens vers d'autres ressources en ligne (cours, livres, sites internet, etc.) car ces choix ne doivent pas être faits "en aveugle". En effet, la réalisation de plans expérimentaux est un processus incluant de nombreuses questions qui ne peuvent être traitées automatiquement. Par exemple, passer d'un plan inter-participant à un plan intra-participant change le

type d'analyses statistiques qui peuvent être faites sur les résultats, etc. En posant les bonnes questions et en permettant d'envisager différentes alternatives, TouchStone doit être utilisé comme un guide et/ou un support pédagogique pour la réalisation de plans expérimentaux. La plateforme de conception produit un script XML décrivant le plan expérimental produit. Le format XML utilisé est lisible par un humain et peut donc aussi être édité "à la main". Il permet aussi d'établir un standard pour la rédaction et le rapport de plans expérimentaux.

Certains sites internet comme WeXtor [152] ou EDGAR [38] et certains logiciels d'analyses statistiques comme JMP [157] proposent des "générateurs d'expérimentations" pour aider à concevoir des types d'expérimentations particuliers. Cependant, aucun système ne permet une approche exploratoire que nous qualifions de "Et si... ?" comme le fait la plateforme de conception de TouchStone. De plus, ces outils couvrent soit une gamme trop large ou trop restreinte d'expérimentations pour être utilisables pour le type d'expérimentations faites en IHM. Pour créer la plateforme de conception, il a fallu trouver le bon arbitrage entre simplicité, c'est-à-dire cibler le sous-ensemble optimal des plans expérimentaux possibles avec une interface minimaliste et efficace, et puissance, c'est-à-dire ne pas se restreindre aux expérimentations "standard" afin de permettre des plans expérimentaux plus "originaux" avec un minimum d'ajustements nécessaires dans le script final. La table de la figure 6.2 montre le découpage de la conception d'une expérimentation selon TouchStone avec un résumé des questions auxquelles l'évaluateur doit répondre au cours de chacune des étapes.

Étape	Décision
2	*Spécifier les facteurs* Quels facteurs et quelles valeurs ? Quels facteurs sont primaires ?
3	*Spécifier les blocs* Quel type de bloc (complet, pur ou mixte) ? Combien de réplications ?
4	*Spécifier le temps* Quels composants d'expérimentation inclure ? Quel temps estimé par événement, essai, pause ? Quel critère de fin pour chaque essai, bloc, entraînement ?
5	*Spécifier l'ordre* Comment ordonner les essais et les blocs ?
6	*Spécifier les mesures* Quelles variables mesurer et sous quelle forme ?

FIG. 6.2 : Résumé des décisions lors de la conception d'une expérimentation

Afin d'encourager le processus itératif de conception d'une expérimentation, chaque étape est présentée dans un onglet séparé afin de permettre à l'évaluateur de naviguer facilement d'une étape à l'autre. Un changement dans un onglet est automatiquement répercuté sur les onglets dépendants. À chaque étape, l'évaluateur peut consulter une aide qui contient les définitions utiles et de nombreux conseils et liens

vers d'autres ressources. En effet, un des buts de la plateforme de conception est aussi d'établir un vocabulaire expérimental bien défini. Par exemple, la littérature en IHM utilise les termes intra-participants (*within participant*) et parfois, mesures répétées (*repeated measures*) qui proviennent de la psychologie ou encore, plus rarement, échantillons corrélés (*correlated samples*) qui provient de l'ingénierie en statistiques qui sont, en réalité, tous synonymes. La plateforme de conception propose un vocabulaire unifié afin de faciliter la clarté et la transparence des plans expérimentaux.

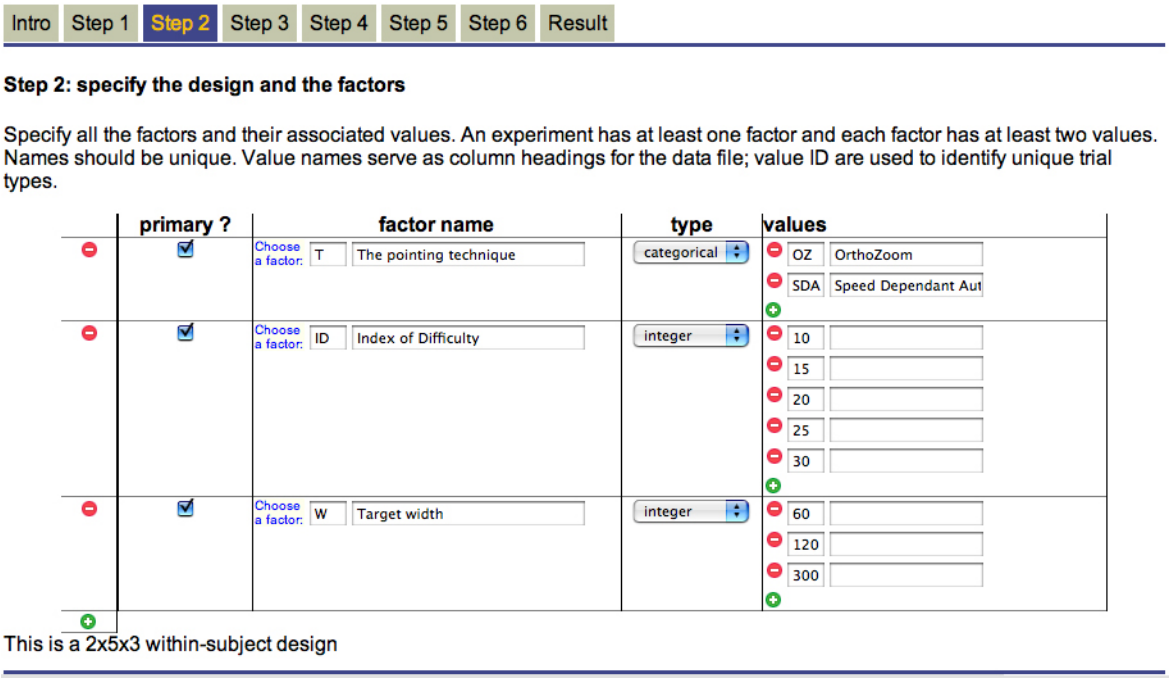


FIG. 6.3 : Plateforme de conception : Spécifier les facteurs

Dans la première étape, l'évaluateur fournit les informations générales de description de son expérimentation : le titre, l'identifiant, les auteurs et une brève description de l'expérimentation. Nous illustrerons les différentes étapes de la plateforme de conception en supposant que nous cherchons à concevoir l'expérience qui nous a permis de comparer les techniques OrthoZoom et Speed Dependant Automatic Zooming sur des tâches de pointage dont l'indice de difficulté varie de 10 à 30 et avec trois largeurs de cible (cf. section 3.2.2).

Dans la seconde étape, l'évaluateur spécifie les facteurs de son expérimentation et les différentes valeurs de ces facteurs (Figure 6.3). L'évaluateur peut sélectionner des facteurs parmi les différents facteurs prédéfinis disponibles dans les menus de l'interface mais il peut également spécifier ses propres facteurs. Les facteurs prédéfinis dépendent des différents *composants* prédéfinis chargés dans TouchStone. Nous reviendrons sur les composants d'expérimentation dans la section 6.3. Dans notre exemple, l'évaluateur sélectionne le facteur prédéfini `T` (Technique) et spécifie ses valeurs. Chaque valeur a un identifiant uti-

lisé pour les fichiers de logs (`OZ` pour OrthoZoom, `SDAZ` pour Speed Dependant Automatic Zooming, etc.). Il ajoute ensuite deux facteurs `ID` et `W` et spécifie leur type (ratio) et leurs valeurs (respectivement {10, 15, 20, 25, 30} et {60, 120, 300}).

L'évaluateur doit maintenant décider quels sont les facteurs primaires et les facteurs secondaires. Les facteurs primaires sont présentés intra-participant (*within participant*), chaque participant étant ainsi soumis à chacune des valeurs de ce facteur, alors que les facteurs secondaires sont présentés inter-participant (*between participant*), un participant étant ainsi soumis à une valeur donnée de ce facteur. Si l'évaluateur choisit au moins un facteur primaire et un facteur secondaire, il s'agit d'un plan expérimental mixte. L'aide de TouchStone décrit les différentes familles de plans existantes (intra-participant, inter-participant et mixte), les avantages et désavantages de chacune et comment le choix d'un type de plan influence le nombre de participants nécessaires pour l'expérimentation.

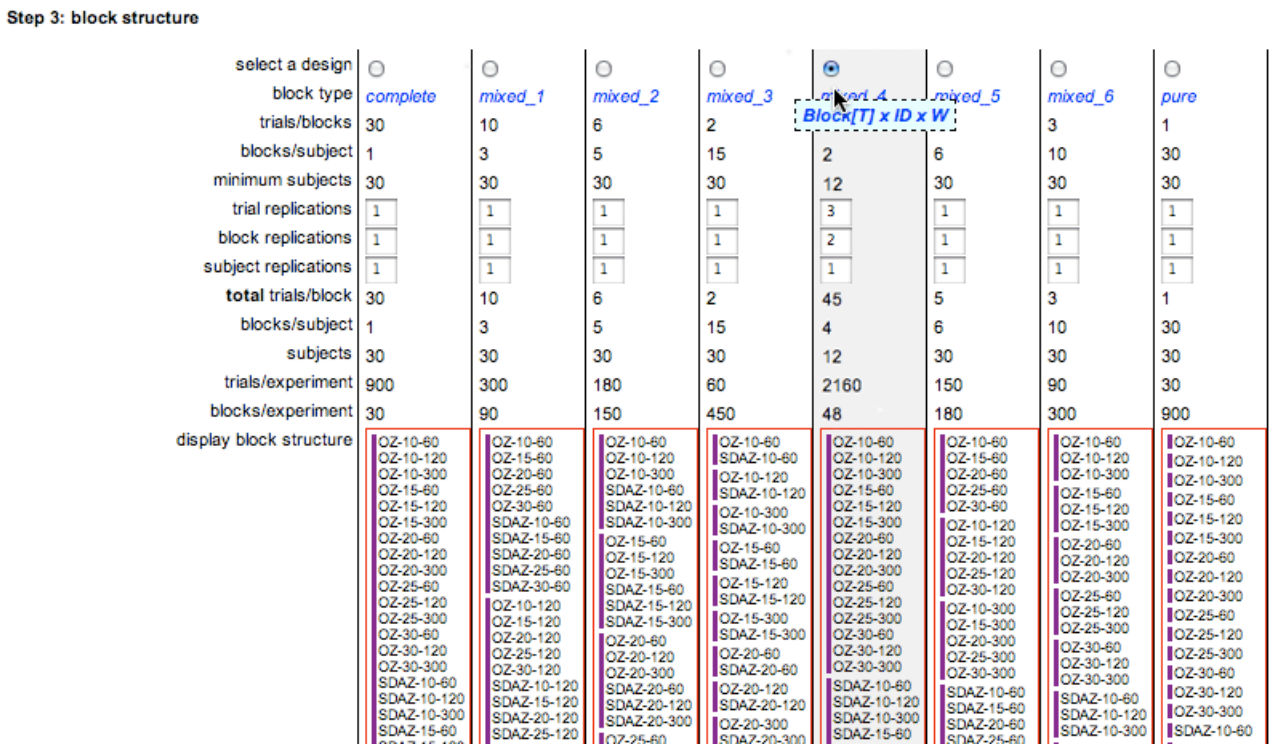


FIG. 6.4 : Plateforme de conception : Structure des blocs

Dans la troisième étape (Figure 6.4), TouchStone rend compte des choix faits à la seconde étape. Ici, l'évaluateur a spécifié un plan 2x5x3 intra-participant, ce qui génère une liste de tous les essais (un essai par condition expérimentale) et un nombre recommandé de participants. TouchStone liste les différentes possibilités de former des blocs parmi l'ensemble des essais depuis le bloc pur (un bloc par condition) jusqu'au bloc complet (un bloc contient toutes les conditions). Dans notre exemple, l'évaluateur choisit une structure de bloc par technique (`T`) afin de présenter les essais d'une même technique en séquence aux

participants. Ce type de groupement est souvent utilisé afin d'éviter de perturber les participants avec des changements successifs. Le plan sélectionné est décrit par l'expression `Block[T] x ID x Width`. Étant donné qu'il s'agit un plan intra-participant, il ne comporte qu'un seul groupe de participants (indiqué par la boîte en rouge) avec deux blocs internes (indiqués par les barres en violet). Cette dernière représentation correspond à une exécution générique de l'expérimentation, c'est-à-dire dont l'ordre des essais n'est pas encore contrebalancé (table de la figure 6.5).

Block 1	Block 2
OZ-10-60	SDAZ-10-60
OZ-10-120	SDAZ-10-120
OZ-10-300	SDAZ-10-300
OZ-15-60	SDAZ-15-60
OZ-15-120	SDAZ-15-120
OZ-15-300	SDAZ-15-300
OZ-20-60	SDAZ-20-60
OZ-20-120	SDAZ-20-120
OZ-20-300	SDAZ-20-300
OZ-25-60	SDAZ-25-60
OZ-25-120	SDAZ-25-120
OZ-25-300	SDAZ-25-300
OZ-30-60	SDAZ-30-60
OZ-30-120	SDAZ-30-120
OZ-30-300	SDAZ-30-300

FIG. 6.5 : Éxécution générique de l'expérimentation

La troisième étape permet aussi à l'évaluateur de spécifier le nombre de réplications de tâches, blocs et participants de l'expérimentation. Une fois de plus, il est invité à consulter les conseils sur les différents choix et leurs conséquences. Par exemple, répliquer les tâches réduit la variabilité intra-participant et donc augmente le probabilité d'obtenir un résultat significatif (si la différence est réellement significative) mais cette réplication augmente aussi le temps d'expérimentation par participant et doit être prise en compte lors de l'analyse des résultats. Dans notre expérimentation, nous avions répliqué trois fois chaque tâche par bloc.

Dans la quatrième étape, l'évaluateur estime le temps d'exécution par participant en incluant les événements pre- et post-expérimentation. Dans notre exemple, nous avions prévu cinq minutes d'explication avant l'expérimentation, un entraînement au début de chaque bloc et un questionnaire de satisfaction à la fin de l'expérimentation ainsi qu'un bloc d'entraînement au début de chaque bloc. L'évaluateur spécifie les *intertitres* qui séparent les blocs et ceux qui séparent les tâches (les messages d'instructions par exemple) et donne une estimation du temps moyen par intertitre. Il spécifie également les critères de fin des blocs et des tâches et fournit une estimation du temps moyen par tâche. Toutes ces informations sont utilisées pour estimer le temps total d'exécution par participant. Si ce temps excède une heure, TouchStone encourage l'évaluateur à reconsidérer le plan qu'il a choisi : le nombre de réplications, de facteurs

et de leurs valeurs ainsi que leur statut (primaire ou secondaire).

Step 5: counter balancing

Design: 2x5x3 within-subject design (Block[T] x ID x W)

	Total = minimium replications ordering				
subject groups	12	12	1		
blocks per subject	4	2	1	random	☑ serial
trials per block	45	15	1	random	☐ serial

	per experiment	per subject	per block
total subjects	12	-	-
total blocks	48	4	-
total trials	2160	180	45
total time	11 h	55 m	13 m 45 s

```
Subject, Block, Trial, T, ID, W
1, 1, 1, "OZ", "10", "60"
1, 1, 2, "OZ", "10", "120"
1, 1, 3, "OZ", "30", "300"
1, 1, 4, "OZ", "10", "300"
1, 1, 5, "OZ", "30", "120"
1, 1, 6, "OZ", "15", "60"
```

FIG. 6.6 : Plateforme de conception : Contrebalancement

Dans la cinquième étape (Figure 6.6), l'évaluateur doit choisir sa stratégie de contrebalancement pour l'ordre de présentation des tâches et des blocs intra- et inter-participant. Le contrebalancement aide à assurer que des effets d'ordre, comme le transfert de connaissances d'une technique à l'autre ou les effets liés à la fatigue, sont distribués à travers les différentes conditions. Le contrebalancement le plus fréquemment utilisé est le carré latin qui assure que chaque participant voit les conditions dans un ordre différent et qui équilibre le rang de présentation et les paires de conditions successives. Si l'expérimentation a un très petit nombre de conditions, l'évaluateur peut opter pour un contrebalancement complet qui présente tous les ordres possibles à chaque participant et, si le nombre de conditions est très élevé (ce qui est le cas de notre exemple : 2x5x3 conditions), l'évaluateur peut choisir de présenter les différentes tâches dans un ordre aléatoire.

À partir de ces choix de contrebalancement, TouchStone génère l'ensemble complet des tâches contrebalancées dans un format de liste qui peut être chargé par un logiciel d'analyses statistiques comme Excel, R ou JMP. L'évaluateur peut donc vérifier le contrebalancement aléatoire proposé par TouchStone afin de tester, par exemple, que le hasard n'a pas conduit à une distribution des conditions ayant des propriétés particulières. L'évaluateur peut également éditer cette liste pour utiliser sa propre stratégie de contrebalancement.

Finalement, dans la sixième étape, l'évaluateur spécifie les mesures (ou *variables dépendantes*) et la granularité de l'enregistrement de ces mesures dans les fichiers de logs : au niveau de l'essai (*trial level*) ou au niveau cinématique (*cinematic level*). Au niveau cinématique, chaque changement de valeur de la mesure provoque l'enregistrement d'une ligne de log. Comme, pour les facteurs, critères de fin et intertitres, l'évaluateur peut choisir les mesures prédéfinies déclarées par les différents composants

chargés ou spécifier ses propres mesures. Le format des enregistrements (ou *log*) est celui utilisé par la majorité des logiciels d'analyses statistiques : chaque ligne est indépendante et liste le participant, les numéros de bloc et de tâche, les facteurs et les mesures. Les logs au niveau cinématique peuvent être utilisés pour rejouer l'expérimentation afin de fournir des interprétations plus détaillées grâce à une analyse plus fine. TouchStone produit des échantillons des fichiers de log afin que l'évaluateur puisse appréhender le format des données qu'il collectera en utilisant la plateforme d'exécution. Dans notre exemple, nous enregistrons le temps de mouvement, le succès et la distance pour chaque tâche alors que nous enregistrons la position du curseur et le facteur de zoom de la vue au niveau cinématique.

Bien que le processus de conception ait été décrit séquentiellement, l'interface par onglets permet à l'évaluateur de procéder dans l'ordre de son choix et de revenir sur ses décisions antérieures. Le dernier onglet présente le résultat de la conception : le fichier XML contenant le script du plan expérimental (dans notre exemple, le script ci-dessous). La première partie du script déclare les différents facteurs et mesures de l'expérimentation tandis que la seconde partie, sur laquelle nous reviendrons à la section suivante, spécifie une exécution du plan expérimental par participant. Une exécution (`<run ...>`) est une séquence de blocs (`<block ...>`) qui peut être entrecoupée d'intertitres (`<interblock ...>`) et chaque bloc est une séquence d'essais (`<trial ...>`) qui peut également être entrecoupée d'intertitres (`<intertrial ...>`). Les attributs `values` spécifient les valeurs des facteurs pour chaque essai et bloc tandis que les attributs `criterion` (ou `criterionTrial`) spécifient les critères d'arrêt des intertitres (ou des essais). Pour améliorer la lisibilité, les balises d'intertitres ont une portée globale afin d'éviter les répétitions : chaque bloc (resp. essai) contenu dans une balise `<interblock ...>` (resp. `<intertrial ...>`) est précédé de l'intertitre spécifié par cette balise. Le code à exécuter pour chaque balise est indiqué par l'attribut `class` qui référence le composant développé grâce à la plateforme d'exécution de TouchStone.

```
<experiment id="MSnav" name="Compare multiscale navigation techniques">
<factor id="T" name="technique" kind="primary" type="nominal" >
  <value id="SDAZ" />
  <value id="OZ" />
</factor>
<factor id="ID" name="Index of Difficulty" kind="primary" type="nominal" >
  <value id="10" />
  <value id="15" />
  <value id="20" />
  <value id="25" />
  <value id="30" />
</factor>
<factor id="W" name="Target width" kind="primary" type="nominal" >
  <value id="60" />
  <value id="120" />
  <value id="300" />
</factor>

<measure id="MT" name="movement time" type="ratio" log="ok" />
<measure id="HIT" name="end trial" type="nominal" log="ok" />
```

```
<measure id="D" name="distance" type="ratio" log="ok" />
<measure id="scale" name="scale ratio" type="ratio" cine_log="ok" />
<measure id="x" name="x-pointer" type="ratio" cine_log="ok" />
<measure id="y" name="y-pointer" type="ratio" cine_log="ok" />
<measure id="xview" name="x-view" type="ratio" cine_log="ok" />
<measure id="yview" name="y-view" type="ratio" cine_log="ok" />

<run id="S1">
  <interblock class="Message({New block})" criterion="Key(Space)">
    <block values="T=OZ"
           criterionTrial="Dwell(1000) | (TimeOut(180000)=>{Too Long})" >
      <intertrial class= "Message({Touch the target as quickly as possible!})"
                  criterion="Press(Mouse.Left)">
      <trial values="ID=15, W=400" class="PointingBlock" />
      <trial ... />
      <trial ... />
      ...
      </intertrial>
    </block>
    <block ... >
      ...
    </block>
  </interblock>
</run>
<run id="S2">
...
</run>
...
</experiment>
```

Ce script peut être utilisé directement dans la plateforme d'exécution, ou sauvé comme un fichier local qui pourra être rechargé dans l'interface de la plateforme de conception afin de réutiliser ou modifier un plan expérimental ultérieurement. Comme nous l'avons précisé précédemment, TouchStone propose ce format de description comme un standard de la spécification d'un plan expérimental et stocke ces plans dans sa base de données d'expérimentations.

6.3 Exécuter des expérimentations contrôlées

Au-delà de l'approche exploratoire offerte par la plateforme de conception, le but de TouchStone est de "stocker des expérimentations", c'est-à-dire non seulement les plans expérimentaux sous forme de scripts XML générés par la plateforme de conception mais aussi les programmes qui exécutent ces expérimentations. C'est pourquoi TouchStone contient non seulement une base de données de scripts décrivant des plans expérimentaux mais aussi une base de données de *composants logiciels* référencés dans ces scripts. Le but de la plateforme d'exécution est d'être la plus modulaire possible afin de pouvoir facilement réutiliser des composants logiciels déjà développés et de ne pas se restreindre à une gamme

trop ciblée d'expérimentations. En effet, les seuls sytèmes, à notre connaissance, qui permettent d'exécuter des expérimentations contrôlées n'assistent pas l'évaluateur de bout en bout lors de la conception d'expérimentations et ont une couverture moins large que celle de TouchStone. Le Generalized Fitts' Law Model Builder [163] ou la plateforme Shakespeare [79] sont tous les deux des systèmes spécifiquement dédiés à la réalisation d'expérimentations qui suivent le paradigme classique de pointage réciproque de Fitts. Ces outils sont très utiles, par exemple, pour mener des expérimentations dans le contexte du standard ISO 9241 [164] sur les dispositifs de pointage mais ne permettent pas d'exécuter des expérimentations plus originales. Récemment, Gray et al. [71] ont proposé un système pour faciliter le développement de variantes de techniques d'interaction pour les comparer en séparant clairement la vue du modèle et en permettant de spécifier une vue qui ne couvre pas tout le modèle (pour comparer des vues qui affichent plus ou moins d'informations par exemple). Ce dernier système a été conçu spécifiquement pour des dispositifs d'interaction portables afin d'adapter facilement l'interface en fonction de l'environnement par exemple. Dans cette section, nous montrons comment la plateforme d'exécution permet d'exécuter un large éventail d'expérimentations utiles en IHM grâce à son architecture modulaire et sa gestion des entrées.

6.3.1 Le module d'expérimentation

Dans cette première section, nous présentons un cas où les différents *composants logiciels* référencés par le script sont déjà dans la base de données de TouchStone afin de nous concentrer sur la description du module d'expérimentation de la plateforme d'exécution. Cette situation serait celle de l'évaluateur qui cherche à répliquer une expérimentation donnée ou qui propose un nouveau plan expérimental n'utilisant que des composants existants. Dans ce cas, l'évaluateur récupère l'application du module d'expérimentation de la plateforme d'exécution[2] et les composants d'expérimentation nécessaires pour exécuter le script de son choix.

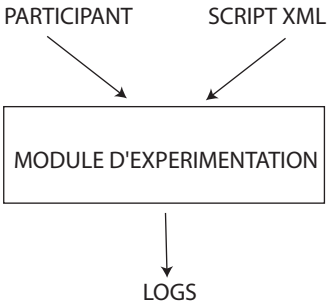


FIG. 6.7 : Entrées/Sorties du module d'expérimentation

[2]La plateforme d'exécution de TouchStone est un projet open source accessible depuis http ://gforge.inria.fr/projects/multiscalenav/

De la plateforme de conception à la plateforme d'exécution

Le module d'expérimentation de la plateforme d'exécution doit "faire tourner" l'expérimentation conçue grâce à la plateforme de conception. La figure 6.7 montre que ce module prend en entrée les instructions du script XML pour gérer l'enchaînement des tâches et les entrées du participant qui déterminent la fin des différentes tâches pour produire, en sortie, des enregistrements des mesures spécifiées par le script XML. Le module d'expérimentation reçoit donc deux types d'entrée : les actions du participant et les instructions du script XML d'expérimentation. Afin de gérer sans conflit ces deux types d'entrée, le déroulement de l'expérimentation a été implémenté selon la machine à états de la figure 6.8 grâce à SwingStates. Cette machine contient deux types d'états : les *états script* et les *états participant*. L'entrée dans un état script reprend l'exécution du script XML et écoute les événements qu'il génère (une balise XML génère un événement de même nom) alors que l'entrée dans un état participant bloque l'exécution du script jusqu'à ce qu'un événement *done* soit reçu. Cet évènement *done* est automatiquement envoyé dès lors que les actions du participant ont satisfait le critère spécifié dans le script. Ainsi, sur la figure 6.8, les états START et BLOCK sont des états systèmes qui attendent des événements pour lancer/terminer un bloc, lancer un intertitre ou une tâche alors que les états INTERBLOCK, INTERTRIAL et TRIAL ne réagissent qu'aux événements *done* qui sont générés lorsque les actions de l'utilisateur ont satisfait un critère donné.

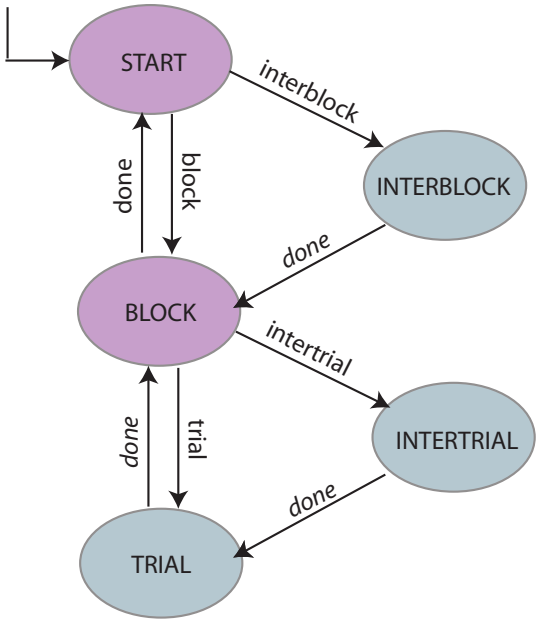


FIG. 6.8 : Machine à états pour les deux sources d'événement d'une expérimentation

Le script généré par la plateforme de conception peut être directement interprété par la plateforme d'exécution dès lors que les composants d'expérimentation utilisés dans le script sont enregistrés dans la plateforme d'exécution. Il y a trois familles de composants d'expérimentations : les blocs, les intertitres et les critères. Afin que ces composants puissent être référencés depuis le script XML par une simple chaîne de caractères, la plateforme d'exécution a été développée en suivant les schémas de conception basés sur le mécanisme de fabriques (*factory* design pattern [69]).

TouchStone contient trois fabriques principales :

1. *Une fabrique de critères*. Cette fabrique permet de construire des critères d'arrêt comme `Key (<key_name>)` qui peuvent être utilisés pour spécifier la fin d'un intertitre (attribut `criterion` des balises `<intertrial ... />` et `<interblock ... />`) ou la fin d'une tâche (attribut `criteriontrial` des balises `<block ... />`). Ces critères peuvent être combinés avec différents opérateurs pour enrichir leur expressivité.
   - Tout d'abord, un critère qui spécifie la fin d'une tâche (`criteriontrial`) peut être un critère de succès (le participant a réussi la tâche de pointage) ou un critère d'échec (le temps imparti pour la tâche est écoulé). Par défaut, un critère est un critère de succès. Pour indiquer qu'il s'agit d'un échec, il suffit d'utiliser l'opérateur => suivi de la raison de l'échec. Dans notre exemple, l'expression `(TimeOut(180000)=>Too Long)` indique qu'au-delà de 3 minutes (180000 ms), la tâche est suspendue et considérée comme un échec car trop longue.
   - Ensuite, il est possible d'appliquer des opérateurs booléens aux critères afin de définir de nouveaux critères. Les quatre opérateurs disponibles sont le *et* (&), le *ou* (|), le *ou exclusif* (^) et la *négation* (!). Ainsi, le critère `Dwell(1000) | (TimeOut(180000)=>Too Long)` de notre exemple indique que soit la tâche se termine en succès lorsque le curseur a été maintenu au moins une seconde dans la cible (`Dwell(1000)`) soit la tâche se termine en échec lorsqu'elle dure trop longtemps (`(TimeOut(180000)=>Too Long)`).
2. *Une fabrique de blocs*. Cette fabrique permet de construire différents types de blocs comme des `PointingBlock` par exemple qui sont des séries de tâche de pointage. Les blocs enregistrés dans cette fabrique peuvent être utilisés dans l'attribut `class` des balises `<block ... />` afin de spécifier le la tâche d'un bloc. Tous les blocs sont des instances de la classe `Block` et contiennent les méthodes :
   - `beginBlock()` et `endBlock()` qui sont appelées respectivement lorsque le bloc commence et se termine. Typiquement, un `PointingBlock` installe une technique au début et la désinstalle à la fin.
   - `beginTrial()` et `endTrial()` qui sont appelées respectivement lorsqu'une tâche de ce bloc commence et se termine. Typiquement, un `PointingBlock` affiche une cible à pointer à chaque nouvelle tâche de pointage et l'efface quand la tâche est terminée.
3. *Une fabrique d'intertitres*. Cette fabrique permet de construire différents types d'intertitres comme un `Message` par exemple pour indiquer des instructions au participant. Les intertitres enregistrés dans cette fabrique peuvent être utilisés dans l'attribut `class` des balises `<intertrial ... />` et `<interblock ... />` afin de spécifier le comportement entre chaque tâche et entre chaque bloc. Tous les intertitres sont des instances de la classe `Intertitle` et contiennent les méthodes :

– `beginIntertitle()` et `endIntertitle()` qui sont appelées respectivement lorsque l'intertitre commence et se termine.

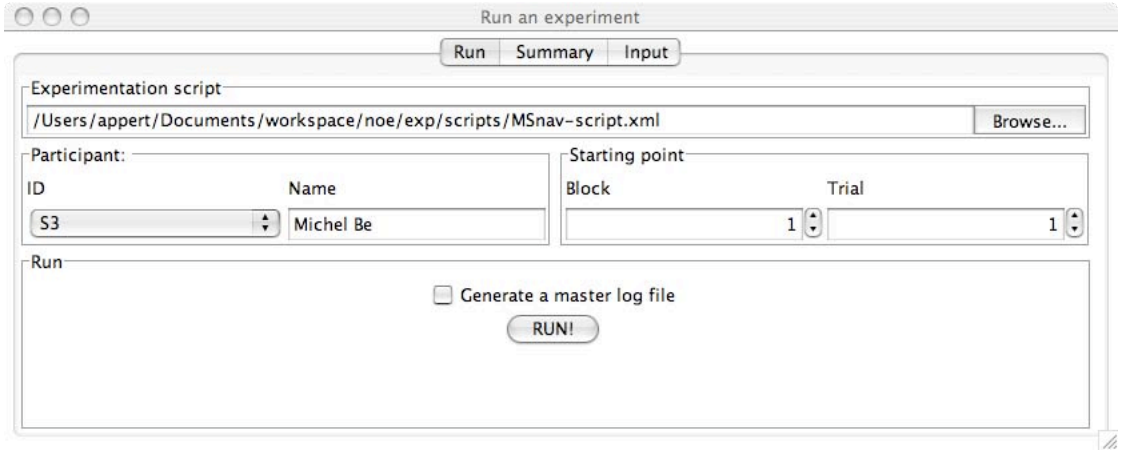


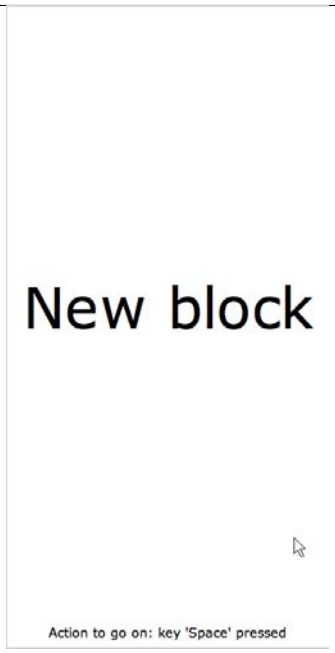
FIG. 6.9 : Onglet *Run* : Lancer une expérimentation

Grâce à ce mécanisme de fabriques, le script est directement exécutable par l'application du module d'expérimentation. L'évaluateur peut charger ce fichier grâce à l'interface du module d'expérimentation (Figure 6.9). Celle-ci contient une liste déroulante des identifiants des participants du script (`S1`, `S2`, ...), une zone de texte pour entrer le nom du participant et deux zones de saisie numérique pour indiquer le point de départ de l'expérimentation (participant, numéro de bloc et numéro de tâche). Ceci est particulièrement pratique pour reprendre une expérimentation qui a été volontairement ou accidentellement interrompue. L'évaluateur peut alors lancer l'exécution de l'expérimentation OZ vs. SDAZ décrite par le script de notre exemple, exécution illustrée par le scénario de la figure 6.10.

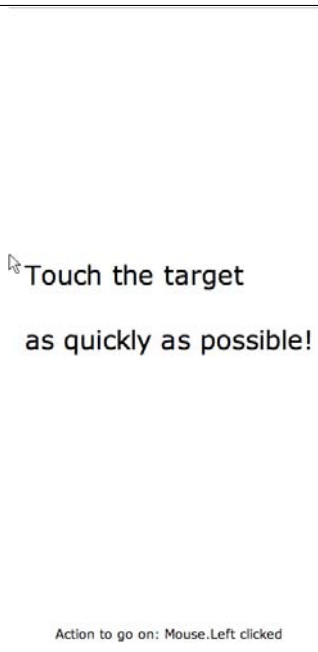
De la plateforme d'exécution à la plateforme d'analyse

L'exécution d'une expérience produit un ensemble de fichiers de log destinés à être exploités avec les outils statistiques recommandés par la plateforme d'analyse de TouchStone. Tous ces fichiers sont stockés dans un répertoire *experiment_id* si l'expérimentation exécutée contient la balise `<experiment id="`*experiment_id*`" ... >`). Ce répertoire contient :

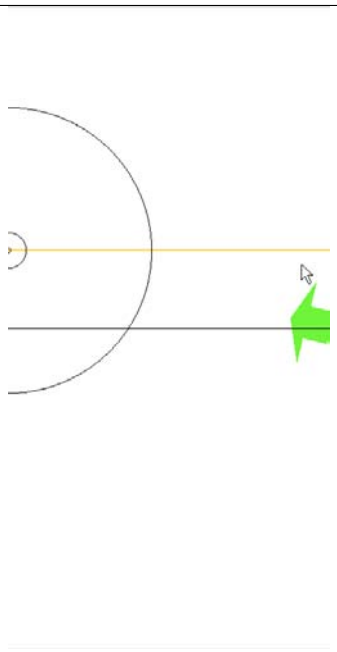
– Deux sous-répertoires contenant respectivement les logs cinématiques et les logs au niveau de la tâche. Les fichier de logs (cinématique et tâche) contiennent un commentaire rappelant le nom de l'expérimentation, les noms des différents identifiants (des facteurs et mesures) utilisés dans le fichier ainsi que la date de l'expérimentation. Le contenu des fichiers est écrit selon un format standard interprétable par tous les logiciels d'analyses statistiques que nous connaissons (R, JMP, Excel, etc.) : des lignes de valeurs séparées par des tabulations, chaque ligne étant indépendante des autres lignes. Pour les logs au niveau de la tâche (figure 6.11), le fichier contient les quatre informations communes à toutes les expériences (le nom de l'expérience, le participant, le numéro de bloc et le numéro de la tâche) puis les facteurs et mesures spécifiées par le script XML (les balises



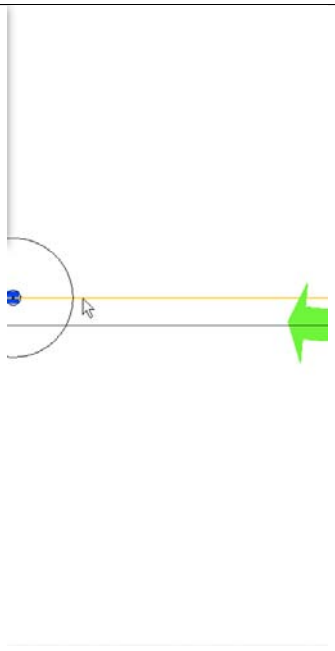
(a) Début de bloc, l'utilisateur presse la touche espace.



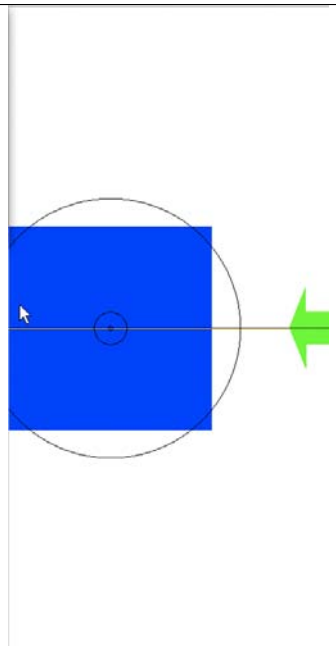
(b) Instructions pour la tâche, l'utilisateur clique.



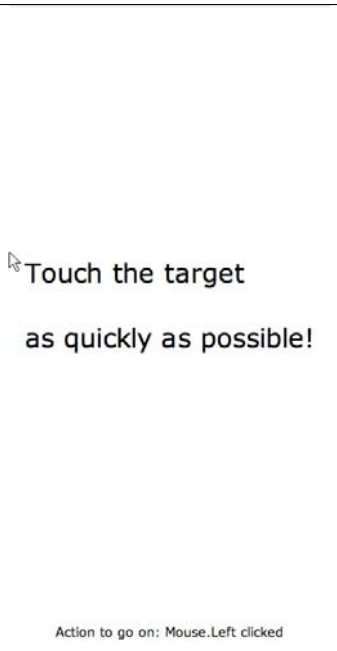
(c) Le participant commence la tâche (ici, la technique est OrthoZoom)



(d) Le participant navigue vers la cible.



(e) Le participant termine la tâche de pointage.



(f) Les instructions pour une nouvelle tâche de pointage sont affichées.

FIG. 6.10 : Exécution de l'expérimentation

`<factor ... />` et `<measure ... />` au début du script **??**). Les fichiers de logs cinématiques ne contiennent eux que les mesures cinématiques spécifiées par le script pour des raisons d'efficacité (chaque évènement sur un périphérique d'entrée pouvant provoquer l'écriture d'une ligne). Nous reviendrons sur la gestion des logs cinématiques à la section 6.3.3. L'exécution d'une expérimentation génère donc deux fichiers de logs par participant, alors que les logiciels d'analyse statistiques prennent en entrée un unique fichier contenant toutes les données de l'expérimentation (pour prendre en compte les effets inter sujets par exemple). C'est pourquoi la plateforme d'exécution permet de générer un unique fichier contenant la totalité des fichiers de logs d'un répertoire ("master log file"), c'est-à-dire les logs de tous les participants. L'évaluateur peut indiquer qu'il désire obtenir ce fichier avant de lancer l'expérience en cochant une case ou, à la fin, grâce à une boîte de dialogue qui apparaît au cas où cette case n'a pas été cochée.

$\#MSnav : Compare\ multiscale\ navigation\ techniques$
$\#scale : scale\ ratio$
$\#x : x - coordinate$
$\#y : y - coordinate$
$\#Date : 2006 - 12 - 05 - 15 - 10 - 01$
$\#Participant : S2$

experiment	*participant*	*block*	*trial*	*T*	*ID*	*W*	*MT*	*HIT*	*D*
nom.	*nom.*	*ratio*	*ratio*	*nom.*	*nom.*	*nom.*	*ratio*	*nom.*	*ratio*
MSnav	*S2*	1	1	*OZ*	10	400	11495	*Dwell*	204600
MSnav	*S2*	1	3	*OZ*	15	200	14462	*Dwell*	6553400

FIG. 6.11 : Un exemple de fichiers de logs au niveau de la tâche

$\#MSnav : Compare\ multiscale\ navigation\ techniques$
$\#scale : scale\ ratio$
$\#x : x - pointer$
$\#y : y - pointer$
$\#xView : x - view$
$\#yView : y - view$
$\#Date : 2006 - 12 - 05 - 15 - 10 - 01$
$\#Participant : S2$

block	*trial*	*scale*	*x*	*y*	*xView*	*yView*
ratio	*ratio*	*ratio*	*ratio*	*ratio*	*ratio*	*ratio*
1	1	1.0	10.0	399.0	10.0	399.0
1	1	0.70275	10.0	398.0	14.2296	398.0

FIG. 6.12 : Un exemple de fichiers de logs au niveau de la cinématique

– Un fichier listant les associations entre l'identifiant du participant et son nom. En effet, avant de lancer l'expérimentation, l'évaluateur entre le nom du participant dans la zone prévue à cet effet mais seul l'identifiant, automatiquement généré par la plateforme de conception (`S1`, `S2`, ...), sera enregistré dans les fichiers de log. La table des associations (identifiant, nom) est stockée à part

car, bien qu'utile à l'évaluateur, elle n'est pas destinée à être publiée dans TouchStone pour des raisons de respect de la vie privée. Pendant l'expérimentation, ce fichier est utilisé pour fournir un résumé de l'expérimentation (partie gauche de la figure 6.13) alors, qu'une fois l'expérimentation terminée, l'évaluateur peut simplement déplacer ou supprimer ce fichier afin d'anonymiser les données avant publication. La partie droite de l'interface montrée sur la figure 6.13 permet de naviguer dans les répertoires contenant les fichiers de log.

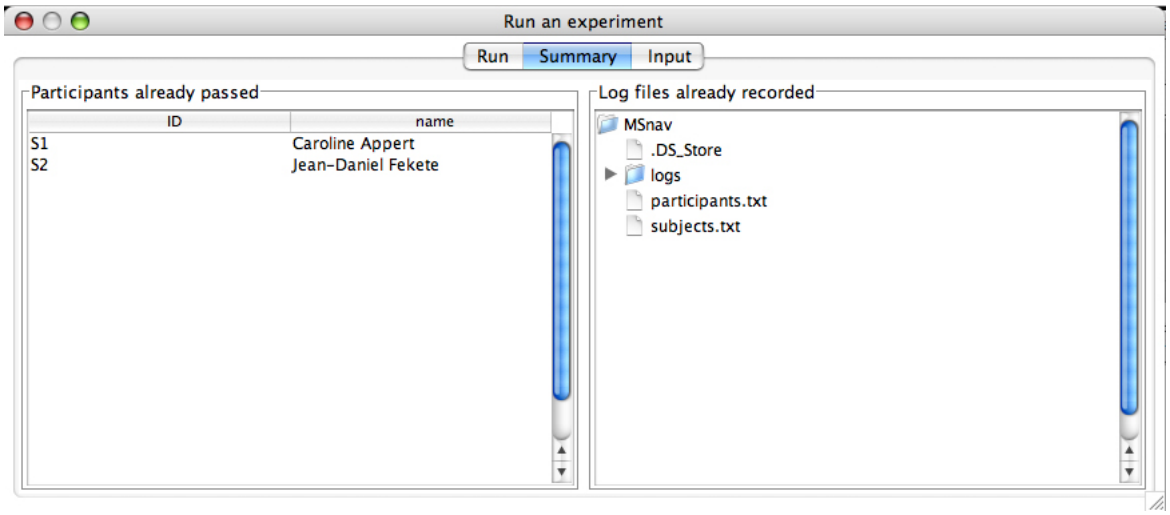


FIG. 6.13 : Onglet *Summary* : À gauche, résumé de l'expérimentation et, à droite, vue sur les fichiers de logs

6.3.2 Définir des composants d'expérimentation

Bien entendu, l'interprétation directe d'une instruction du script XML en code exécutable par le module d'expérimentation de la plateforme d'exécution n'est possible que si chacun des *composants d'expérimentation* référencés par celui-ci ont été développés avec l'interface de programmation de TouchStone. Un composant d'expérimentation est une classe Java correspondant à un bloc, critère, intertitre, facteur et mesure accompagnée de son fichier de description. Dans la section précédente, nous faisions l'hypothèse que les composants logiciels requis par le script étaient présents dans la base de données de TouchStone, ce qui correspond à une situation de réplication ou extension d'une expérience déjà réalisée et exportée. Dans cette section, nous levons cette hypothèse en supposant que les différents composants (`OZ`, `PointingBlock`, etc.) ne sont pas présents. L'évaluateur doit donc développer ses propres composants pour une expérimentation de pointage, ce qui correspond à la situation de réalisation d'une expérience à partir de zéro. Pour développer des composants, l'utilisateur doit récupèrer l'environnement de développement de TouchStone et utiliser l'interface de programmation que nous présentons ci-dessous. Les fichiers de description des composants développés sont automatiquement produits dès lors que cette interface de programmation est respectée.

TouchStone ayant pour but de collecter le maximum d'éléments, nous proposons une interface de programmation pour les composants d'expérimentation qui permette d'enregistrer ces composants auprès des différentes fabriques de TouchStone (afin de les référencer par de simples identifiants) et d'exporter leurs définitions vers la plateforme de conception. Ainsi, ces composants pourront être utilisés par d'autres plans expérimentaux puisqu'ils seront accessibles depuis la plateforme de conception. Un composant d'expérimentation est une classe Java destinée à être utilisée comme valeur pour les attributs `criterion`, `criterionTrial`, `class` et `values` dans les scripts XML.

Afin d'offrir cette possibilité, la plateforme d'exécution repose sur un mécanisme de fabriques (*factories*) de composants couplé avec un *doclet*. D'une part, une fabrique est une classe Java qui gère une table d'association (*identifiant*, *classe Java*) et qui construit des objets java à la demande à partir d'un identifiant. Ainsi, par exemple, on peut définir une fabrique d'intertitres qui se charge de construire un objet de la classe `Message` affichant un message à partir de l'identifiant l'identifiant "Message". Nous heurtant aux limites de l'interface de réflexion de Java, nous avons eu recours au mécanisme de *doclet* afin de spécifier la table d'associations. En effet, en utilisant l'interface d'introspection de Java (`java.lang.reflect`), il est possible d'explorer le contenu des classes et de voir que la classe `Message` est une sous-classe d'`Intertitle` par exemple et ainsi de créer l'association entre l'identifiant "Message" et cette classe. Cependant cette approche oblige à avoir le nom de la classe pour identifiant. On peut alors proposer de surcharger une méthode, `getNames()` par exemple, qui fournit la liste de noms pour une sous classe d'`Intertitle` mais ceci pose encore des problèmes. En effet, si la méthode `getNames()` est une méthode de classe, il est impossible d'obliger le développeur à la définir avec le mécanisme d'héritage et, si c'est une méthode d'instance alors il faut disposer d'une instance. Dans ce deuxième cas, le problème auquel nous sommes confrontés est celui de l'ordre de construction des différents composants logiciels de l'expérimentation. En effet, lors du développement, l'évaluateur peut faire l'hypothèse d'un certain contexte d'exécution et il n'est donc pas garanti que la construction dans l'ordre proposé par l'interface de réflexion puisse se faire avec succès. L'ordre de construction supposé est celui de l'expérimentation or, rappelons qu'ici, l'évaluateur n'est pas supposé avoir complètement spécifié son plan expérimental et donc cet ordre n'est pas connu. Ce sont ces contraintes qui nous ont orienté vers une approche plus déclarative rendue possible grâce au *doclet* de TouchStone qui exploite les informations dans les commentaires de classes Javadoc. Un doclet est un programme Java exécuté au moment de la génération de la documentation et qui, permet, notamment d'accéder facilement à des commentaires et d'y associer un traitement dès lors qu'ils portent une étiquette donnée. Le doclet de TouchStone exploite les étiquettes de la forme `@touchstone.*` pour générer des fichiers de propriétés qui seront utilisés pour spécifier la table d'associations des fabriques et les informations descriptives requises par la plateforme de conception (Figure 6.14).

Définir des intertitres, blocs et critères

Définir un nouvel intertitre, bloc ou critère revient à ajouter une nouvelle classe d'objets avec l'identifiant permettant d'y accéder depuis l'extérieur à l'une des trois fabriques existantes de TouchStone :

La fabrique d'intertitres Elle stocke des sous-classes de la classe `Intertitle` qui contient les méthodes `beginIntertitle` et `endIntertitle` destinées à être surchargées dans les sous-classes

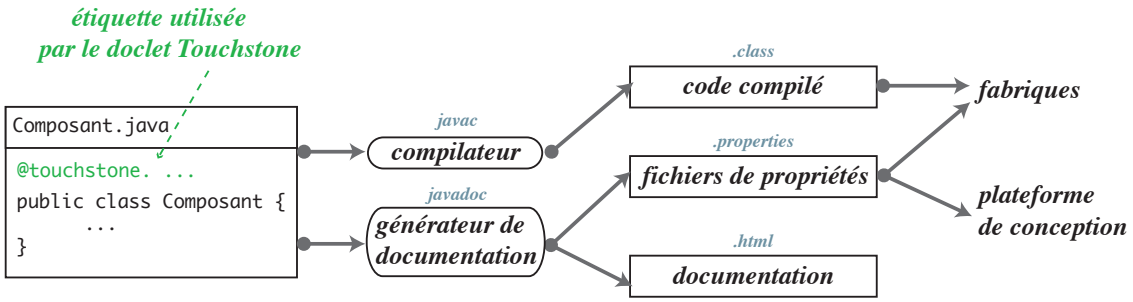


FIG. 6.14 : Mécanisme de fabriques et doclet de TouchStone

pour spécifier l'installation et la désinstallation d'un intertitre. Par exemple, la classe qui définit l'intertitre qui affiche un message que nous utilisons dans notre expérimentation sur les techniques de pointage multi-échelle est codée comme ci-dessous :

```
/**
 * @touchstone.intertitle Message
 */
public class Message extends Intertitle {
     public Message(String text) { super(); }
     public void beginIntertitle() {
      // installe une vue et affiche le message
     }
     public void endIntertitle() {
      // désinstalle la vue
     }
}
```

La fabrique de blocs Elle stocke des sous-classes de la classe `Block` qui contient les méthodes `beginBlock`, `endBlock`, `beginTrial` et `endTrial` destinées à être surchargées dans les sous-classes pour spécifier l'installation et la désinstallation d'un bloc et d'une tâche (c.à.d. *trial*). Par exemple, la classe qui définit le bloc de pointage que nous utilisons dans notre expérimentation sur les techniques de pointage multi-échelle est programmée ainsi :

```
/**
 * ...
 * @touchstone.block FittsBlock
 * @touchstone.block PointingBlock
 */
public class FittsBlock extends Block {
     public FittsBlock() { super(); }

     public void beginBlock() {
          // installe la vue et la technique
```

```
            // en fonction de la valeur du facteur T
        }

        public void beginTrial() {
            // installe la cible en fonction
            // des valeurs des facteurs ID et W
        }

        public void endTrial(EndCondition ec) {
            // efface la cible
        }

        public void endBlock() {
            // désinstalle la vue
        }

}
```

Comme pour les intertitres, le tag `@touchstone.block` permet de définir les identifiants qui pourront être utilisés pour référencer un tel bloc depuis l'extérieur. Notons sur cet exemple qu'il est possible de définir plusieurs identifiants pour accéder aux objets d'une fabrique : les identifiants `FittsBlock` et `PointingBlock` référencent la même classe d'objets par exemple.

La fabrique de critères Elle stocke des classes qui implémentent l'interface `EndCondition` ou qui héritent de la classe `AbstractEndCondition`. Pour indiquer qu'une tâche est terminée en succès ou en échec, le programmeur peut appeler explicitement les méthodes `hit()` et `miss()` au moment de son choix. Il peut également spécifier un critère dans les attributs `criterion` des différentes balises du script XML, le critère sera alors évalué automatiquement par la plateforme d'exécution à chaque événement de l'utilisateur (dans ce cas, les méthodes `hit()` et `miss()` sont automatiquement appelées par la plateforme d'exécution). Pour définir un nouveau critère d'arrêt :

```
/**
 * @touchstone.criterion  Press
 */
public class PressCondition extends AbstractEndCondition {

  protected String button;
  protected long endTime = -1;

  public PressCondition(String button) {
    super();
    this.button = button;
  }

  public boolean isReached(AxesEvent e) {
      if(e.isAxisModified("Mouse."+button)
         && e.getAxisValue("Mouse."+button) == 1) {
        endTime = e.getAxisTime("Mouse."+button);
```

```
        return true;
      }
    }
    return false;
  }

  public boolean isReached(InputEvent e) {
    if(e.getType() == MouseEvent.MOUSE_PRESSED) {
      endTime = e.getAxisTime("Mouse."+button);
      return true;
    }
    return false;
  }

  public String getEndCondition() {
    return "Mouse."+button+" pressed";
  }

  public long getEndTime() {
    return endTime;
  }

}
```

Un critère contient deux méthodes `isReached` afin de pouvoir être utilisé avec le système d'événements de TouchStone (`AxesEvent`), décrit plus loin, soit avec le système d'événements standard Java AWT. Il est important de noter qu'avec le système d'événements de TouchStone, les événements sont entendus quelle que soit la position à laquelle ils surviennent alors qu'avec le système AWT, il devient nécessaire de spécifier le composant à écouter en utilisant l'instruction :

```
Platform.getInstance().registerComponent(component);
```

Définir des facteurs et des mesures

La classe `Platform` est au coeur du développement d'une expérimentation avec TouchStone. C'est une classe singleton (il n'existe qu'une unique instance) qui, d'une part, est la fenêtre principale de l'expérimentation et, d'autre part, sert à répertorier toutes les informations de l'expérimentation. Elle peut être questionnée pour ajuster ou obtenir la valeur d'un facteur ou d'une mesure grâce aux méthodes `getFactorValue(`*factor_id*`)` et `getMeasureValue(`*measure_id*`)`.

```
Platform.getInstance().getFactorValue("T");
Platform.getInstance().getMeasureValue("D");
```

Les facteurs sont les variables contrôlées par le script d'expérimentation. Une instruction du script XML qui contient une expression de la forme `values = "`*factor_id*`= `*factor_value*`"` provoque l'affectation de la valeur *factor_value* au facteur *factor_id*. Par défaut, les valeurs des facteurs sont stockées comme des chaînes caractères. Cependant, ceci peut mener à des programmes comportant de nombreuses comparaisons sur des chaînes de caractères et ne permettant pas la modularité dans un système à base de

composants comme TouchStone. Prenons l'exemple du facteur technique (`T`) dans notre expérimentation de pointage. Le code devra forcément contenir un ensemble de tests pour installer la bonne technique en fonction de la chaîne de caractères spécifiant la valeur ("SDAZ", "OZ", etc.). Non seulement, c'est un mode de programmation propice à l'erreur mais aussi, il est impossible d'ajouter une nouvelle valeur, "PZ" par exemple, sans avoir accès à ce code. TouchStone permet de définir des facteurs de type plus complexe auxquels il est possible d'ajouter des valeurs sans avoir à modifier le code existant. Définir un facteur avancé consiste techniquement à définir une nouvelle fabrique dans TouchStone grâce à l'interface de programmation que nous illustrons ici. Pour définir un nouveau facteur `T`, il faut tout d'abord définir le facteur lui-même :

```
/**
 * @touchstone.factor T
 *      name: "Technique"
 *      help: "The pointing technique"
 */
 public class TechniqueFactor extends Factor {
     public TechniqueFactor() { super("T"); }
     public void setValue(Object value) {
         // installe la technique dans l'interface
         // (value peut être supposé de type Technique)
     }
 }
```

La classe `Factor` est une fabrique de singletons (une même valeur d'un facteur au cours d'une expérimentation est un même objet Java). La méthode `setValue` peut être surchargée pour spécifier un traitement lorsque la valeur de ce facteur est spécifiée par le script XML. Les méta-données *name* ou *help* contenues dans le commentaire d'entête sont utilisées par le doclet de TouchStone qui génère les fichiers de description du composant, d'une part, pour décrire ce facteur lors de son exportation vers la plateforme de conception, d'autre part, pour générer les fichiers nécessaires à l'initialisation des fabriques de TouchStone. Chaque nouveau facteur est ainsi une nouvelle fabrique de TouchStone à laquelle on peut demander de construire des objets valeur. Pour compléter la définition de ce facteur, il faut également définir ses valeurs comme les techniques Speed Dependant Automatic Zooming ou OrthoZoom par exemple.

```
/**
 * @touchstone.value SDAZ
 *      factor: T
 *      name : "SpeedDependantAutomaticZooming"
 *      help: "rate-based scrolling with zoom factor
 *                  adapted to the scrolling speed"
 */
public class SDAZTechnique implements Technique {
     public SDAZTechnique() { super("SDAZ"); }
     ... // méthodes spécifiques à la technique SDAZ
}

/**
```

```
 * @touchstone.value OZ
 *      factor: T
 *      name : "OrthoZoom"
 *      help: "scrolling along the colinear dimension
 *                  while zooming along the orthogonal direction"
 */
public class OZTechnique implements Technique {
     public OZTechnique() { super("OZ"); }
     ... // méthodes spécifiques à la technique OrthoZoom
}
```

`T` est désormais une fabrique de TouchStone, ce qui permet de définir des nouvelles valeurs pour ce facteur dans d'autres expérimentations. Ainsi, si l'évaluateur avait récupéré le composant `T`, il aurait pu définir une nouvelle valeur `PZ` qui implémente la technique classique qui consiste à zoomer avec la molette de la souris et à se déplacer à zoom constant grâce à une interaction de drag.

```
/**
 * @touchstone.value PZ
 *      factor: T
 *      name : "Pan and Zoom"
 *      help: "scroll using panning and
 *                  zoom using the mouse wheel"
 */
public class PZTechnique implements Technique {
     public PZTechnique() { super("PZ"); }
     ... // méthodes spécifiques à la technique Pan and Zoom
}
```

Les mesures sont les variables dont les valeurs sont enregistrées pendant l'exécution d'une expérimentation (à chaque changement de valeur dans le cas d'une mesure cinématique, à chaque fin de tâche sinon). Le script d'expérimentation spécifie quelles variables doivent être enregistrées grâce aux instructions de la forme `<measure id="`*measure_id*`" ... >`. Il faut donc que la description de cette mesure soit disponible dans la plateforme de conception et que la plateforme d'exécution sache calculer la valeur de cette mesure. Pour définir une mesure, il faut qu'un composant enregistre une instance de la classe `Measure` auprès de l'objet `Platform` et définisse la méthode calculant la valeur de cette mesure. La description qui sera exportée vers la plateforme de conception se situe dans le commentaire de la classe qui définit et enregistre la mesure auprès de l'objet `Platform`. Par défaut, tout *axe d'événement*, physique ou virtuel, est une mesure prédéfinie que le script peut référencer sans que le développeur n'ait à programmer quoi que ce soit. Nous reviendrons sur les *axes d'événement* à la section suivante. L'expérimentateur peut également enregistrer des mesures exportées par les composants qu'il utilise. Par exemple, le composant FittsBlock ci-dessous exporte la mesure `D` qui correspond à la distance à la cible dans la tâche de pointage (elle est calculée à partir des valeurs des facteurs `ID` et `W` grâce à la forme $ID = log_2(\frac{D}{W})$).

```
/**
 * @touchstone.measure D
 *      name: "Distance to target"
```

```
 *      help: "Distance to target"
 *      type: ratio
 * ...
 */
public class FittsBlock extends Block {
  public FittsBlock() {
    super();
    Platform.getInstance().addMeasure(
        new Measure("D") {
          public Object getValue() {
            // calcul de la valeur de D en fonction des facteurs
            // ID et W grâce à l'équation ID = log2(D/W)
            double w = Double.parseDouble(
                Platform.getInstance().getFactorValue("W"));
            double id = Double.parseDouble(
                Platform.getInstance().getFactorValue("ID"));
            return new Double(w*(Math.pow(2, ID));
          }
      });
   }
}
```

6.3.3 La gestion de l'entrée généralisée

Comme nous l'avons mentionné précédemment, TouchStone dispose de sa propre gestion des entrées afin de permettre l'utilisation de périphériques d'entrée "non standard" (contrairement au système d'événements AWT, `java.awt.event`, qui n'entend que les événements provenant d'une unique souris et d'un clavier). Nous avons défini un système plus général de gestion des événements grâce à la librairie JInput [147]. Ainsi, TouchStone permet d'étudier l'influence de facteurs humains autres que les facteurs purement moteurs sur l'interaction et de gérer une plus grande variété d'entrées permettant ainsi d'évaluer des techniques bi-manuelles, des techniques basées sur la vision [60], etc.

JInput décrit les périphériques d'entrée en termes de *contrôleurs* (une souris par exemple) et de *composants* qui génèrent des valeurs (pour la souris, les boutons et les axes de déplacement). Par exemple, pour un ordinateur portable équipé d'une souris USB, JInput définit trois contrôleurs : "Trackpad", "Keyboard" et "USB mouse" avec plusieurs composants comme "x", "Space" ou "button1" par exemple. JInput permet de consulter les contrôleurs disponibles et gère une file d'événements par contrôleur (un événement est généré chaque fois qu'un composant génère une valeur. Nous avons défini une interface de programmation permettant d'utiliser cette gestion des entrées avec des écouteurs selon des *axes d'événements* afin que les programmeurs puissent l'utiliser avec les paradigmes de programmation usuels basés sur les écouteurs d'événements. Chaque *axe physique* peut être désigné en utilisant la syntaxe `<contrôleur>.<composant>` ("Trackpad.x", "Keyboard.Space", "USB Mouse.button1", etc.). Un objet peut déclarer un ensemble d'*axes d'intérêt* lors de son inscription auprès de l'*InputManager*. L'*InputManager* consulte régulièrement les files d'événements de tous les contrôleurs et envoie un événement aux objets inscrits lorsque l'un de leurs axes d'intérêt est modifié. L'événement transmis aux différents écouteurs permet de consulter la liste de tous les axes modifiés ainsi que leur valeur. Pour construire

un écouteur d'axe, il suffit d'utiliser l'interface `AxesListener`. La méthode `axesChanged(AxesEvent e)` est la fonction de rappel à implémenter pour récupérer ces événements.

```
public class MyTechnique implements AxesListener {
     public void axesChanged(AxesEvent e) {
          if(e.isAxisModified("Mouse.x") || e.isAxisModified("scale")) {
               int x = (int)e.getAxisValue ("Mouse.x");
          }
     }
}
...

// enregistrement de l'axe ``scale'' pour MyTechnique
Platform.getInstance().addAxesListener("scale",new MyTechnique());
```

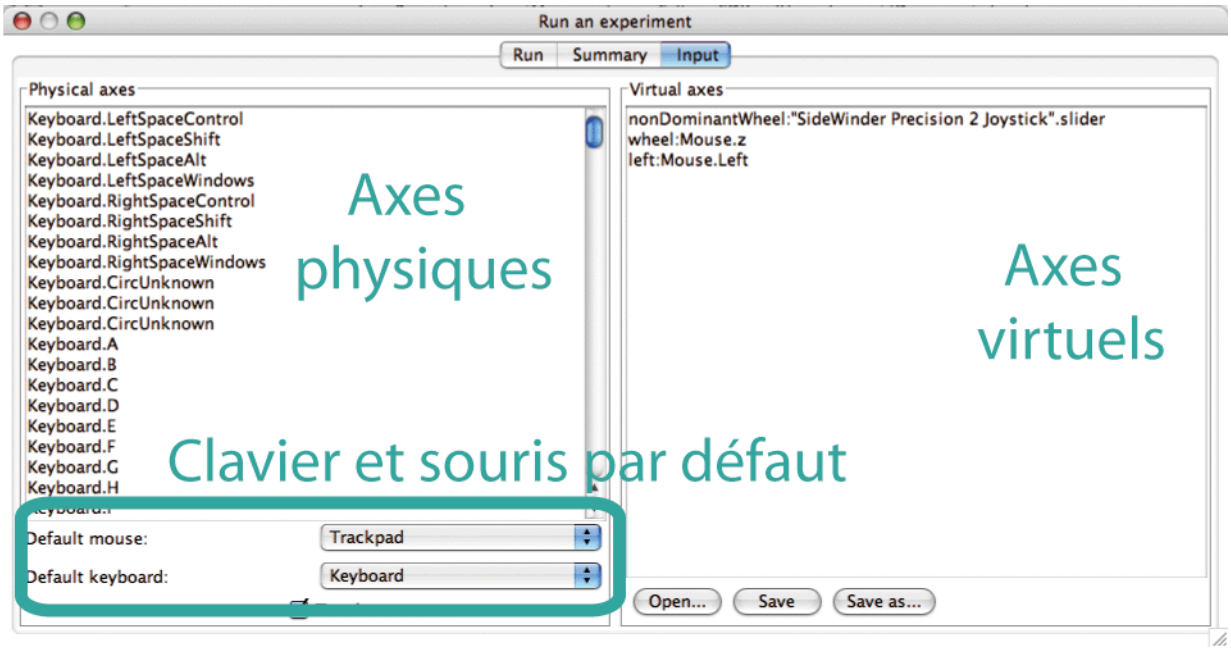


FIG. 6.15 : Onglet *Input* : Gérer les axes d'entrée

Les noms des contrôleurs (et donc des axes physiques) dépendant du pilote du périphérique et du système d'exploitation utilisés, ceux-ci sont consultables lors du lancement de la plateforme d'exécution afin que le programmeur puisse facilement accéder à cette information. La partie gauche de l'onglet *Input* de l'interface du module d'expérimentation présente la liste exhaustive des axes physiques disponibles (Figure 6.15). Il est également possible de définir des *axes virtuels* de plus haut niveau que les axes physiques disponibles. La partie droite de l'interface de lancement montre le fichier de configuration qui définit les *axes virtuels* disponibles pour cette expérimentation. TouchStone permet de définir des axes

virtuels avec une syntaxe simple incluant des opérateurs et des méthodes pour composer différents axes. Ainsi, par exemple, les évènements d'un contrôleur de type souris sont des événements relatifs et il est souvent plus aisé d'accéder à une position absolue de la souris.En utilisant les opérateurs d'addition et de multiplication et la méthode d'intégration (`integrate`), nous pouvons définir un axe virtuel absolu "Mouse.x" à partir d'un axe physique relatif "Trackpad.x" grâce à l'expression suivante :

```
Mouse.x : integrate(Trackpad.x, -Window.width/2, Window.width/2)
          + Window.width/2
```

Grâce à ce mécanisme, il est possible de développer une technique en fonction d'un axe virtuel et de ne changer que la définition de l'axe virtuel pour obtenir plusieurs techniques. Par exemple, le "Pan and Zoom unimanuel" et le "Pan and Zoom bimanuel" sont la même technique qui dépend de l'axe virtuel "zoom". Dans le premier cas, cet axe est défini en fonction de la molette de la souris par défaut alors que, dans le second cas, il est défini en fonction du "slider" du joystick de la main non dominante. Enfin, pour faciliter l'implémentation des techniques n'utilisant que la configuration standard "clavier+souris", TouchStone définit deux contrôleurs virtuels "Mouse" et "Keyboard" pour les souris et clavier par défaut. Les évaluateurs peuvent utiliser ces axes pour développer des techniques sans avoir à les définir dans un fichier de configuration. Pour associer un contrôleur physique à chacun de ces deux contrôleurs virtuels, il suffit à l'évaluateur de sélectionner les deux périphériques parmi une listes des contrôleurs souris et une liste des contrôleurs claviers disponibles (partie bas gauche de la figure 6.15). Grâce à un tel système, il est facile de définir de nouvelles techniques à partir de techniques existantes sans avoir à changer ou recompiler du code existant. Par exemple, la technique Zliding [151] peut être obtenue à partir de la technique de "Pan and Zoom" générique en utilisant le stylet comme souris par défaut et la pression de celui-ci pour définir l'axe virtuel de zoom.

6.4 TouchStone : Validation et exemples d'utilisation

Évaluer TouchStone consiste à mesure son utilité pour mener une expérimentation contrôlée. D'une part, nous menons actuellement une étude d'utilisabilité de la plateforme de conception en se focalisant sur l'approche exploratoire et l'ensemble des expérimentations couvert par TouchStone . D'autre part, nous illustrons comment nous avons utilisé TouchStone pour concevoir deux expérimentations contrôlées utiles dans le cadre de ce travail de thèse : celle qui nous a permis d'ajuster les valeurs des constantes de la loi de Hick-Hyman utilisées par SimCIS et celle nous permettant d'unifier les résultats de différentes techniques de pointage afin d'obtenir des résultats qui pourront être utilisés pour optimiser des actions de CIS.

6.4.1 Évaluation de la plateforme de conception

Une évaluation de la plateforme de conception est actuellement en cours. Elle vise à mesurer le taux de couverture des plans expérimentaux rencontrés en IHM et l'intérêt de l'approche exploratoire offerte par celle-ci. Pour tester la couverture, comme dans le cas de SwingStates, l'approche la plus naturelle semble être l'utilisation de benchmarks, c'est-à-dire tester pour un ensemble de plans expérimentaux si la plateforme de conception de TouchStone permet de les décrire. L'évaluateur demande aux participants de décrire leur propre expérimentation ou une expérimentation extraite de la littérature en utilisant TouchStone. Pour tester l'approche exploratoire, l'évaluateur demande à ces mêmes participants d'étendre les

expérimentations décrites en ajoutant un facteur ou une valeur et d'explorer les différentes alternatives de conception en rapportant par exemple leur impact sur le nombre de participants ou la stratégie de contrebalancement. Nous avons auparavant précisé que la plateforme de conception de TouchStone ne doit pas être utilisée "en aveugle". Aussi ce sont des étudiants en Interaction Homme-Machine qui sont sélectionnés pour participer à cette étude.

Bien que l'étude soit toujours en cours, les premières observations sont encourageantes. Premièrement, dans beaucoup de cas, les informations rapportées dans la littérature ne décrivent pas les plans expérimentaux dans leur totalité : plusieurs participants ont, par exemple, rapporté qu'il leur était impossible de décrire la stratégie de contrebalancement utilisée pour l'expérimentation qu'ils avaient à étudier. L'utilisation de la plateforme de conception de TouchStone devrait encourager la publication des plans et des résultats des expérimentations dans un format "standard", comme celui proposé par touchstone, pour plus de transparence et pour faciliter réplications et extensions des expérimentations publiées. Deuxièmement, l'exploration des plans expérimentaux doit être faite de manière avertie. En effet, malgré les conseils fournis par la plateforme de conception, certains participants, n'ayant jamais réalisé une expérimentation, ont conçu des plans expérimentaux dans lesquels certains effets n'étaient pas contrôlés. La plateforme de conception listant toutes les possibilités peut inciter l'évaluateur à penser que toutes les possibilités sont équivalentes lorsqu'elles ne le sont pas. Par exemple, sélectionner la structure des blocs à la troisième étape ne consiste pas simplement à "obtenir le bon nombre" mais doit être justifié par rapport au rôle des différents facteurs (éviter de perturber le participant par des changements successifs de techniques par exemple). Les remarques collectées lors de cette étude permettront de compléter l'aide de la plateforme de conception de TouchStone.

6.4.2 TouchStone et CIS

Concevoir des expérimentations pour raffiner les prédictions de CIS

Nous avons montré à la section 3.1.1 que les prédictions de CIS provenaient de lois empiriques qui sont étudiées en utilisant des expérimentations contrôlées. En effet la finesse de modélisation de CIS repose sur la connaissance des différentes lois empiriques sous-jacentes à l'interaction (choix, pointage, suivi de tunnel, etc.) qui est acquise par la réalisation d'expérimentations contrôlées. Ainsi, nous avions eu besoin d'une expérimentation pour ajuster les coefficients de la loi de Hick-Hyman afin de raffiner les prédictions de SimCIS. La conception d'une telle expérimentation est très simple avec la plateforme de conception de TouchStone puisqu'il ne s'agit que de définir un nouveau facteur `nbElements` (nombre d'éléments dans l'ensemble) à 8 valeurs (3, 5, 8, 11, 15, 19, 24 et 29) et de choisir une stratégie de contrebalancement. Nous obtenons un script XML dont nous rapportons un extrait (Fragment de code 1).

Fragment de code 1 (Script XML pour l'expérimentation de Hick-Hyman)

```
<experiment id="HickExperiment" name="Hick-Hyman's law experiment">
  <pluginRequired url="http://www.lri.fr/~appert/touchstone/hickHyman.jar" />
  <factor id="N" name="nbElements" kind="primary" type="nominal" >
    <value id="5" />
    <value id="10" />
```

```
    <value id="15" />
  </factor>

  <measure id="MT" name="movement time" type="ratio" log="ok" />
  <measure id="HIT" name="end trial" type="nominal" log="ok" />

  <run id="S1">

  <intertrial class="HickIntertitle" criterion="PressOnTag(Left, start)" >
  <interblock
    class="Message({Read the number of the target to reach.\n
                    Click on the START circle.\n
                    Click the target having the right number.})"
    criterion="Key(Space)" >

  <block
    class="HickBlock"
    criterionTrial="PressOnTag(Left, target) |
                   (PressOnTag(Left, distractor)=>{Distractor})">
    <trial values="N=3"/>
     ...
  </block>

  </interblock>
  </intertrial>

  </run>

</experiment>
```

La programmation d'une expérimentation avec la plateforme d'exécution de TouchStone consiste à identifier et implémenter les composants (blocs, essais, intertitres et critères) nécessaires à l'expérimentation (Figure 6.16). La plateforme de conception est très modulaire, et le programmeur peut en particulier utiliser la boîte à outils de son choix. Ici, nous avons choisi d'utiliser notre boîte à outils SwingStates.

Nous pouvons réutiliser l'intertitre `Message` et le critère `Key(...)` pour séparer les blocs de notre expérimentation. Cependant, nous avons besoin d'un nouveau type de bloc (`ReactionBlock`) qui affiche une cible parmi un ensemble de distracteurs et d'un nouveau type d'intertitre (`StartButton`) qui affiche un bouton au centre de l'écran (pour assurer que le curseur sera à cette position au début de l'essai suivant). Nous définissons également un nouveau critère d'arrêt `PressOnTag(`*mouse_button*`,` *tag_name*`)` en exploitant le mécanisme de tags de SwingStates : lors d'un clic de souris, nous regardons s'il a eu lieu sur une forme possédant le tag *tag_name*. Ainsi, nous pouvons utiliser ce critère pour les intertitres avec le tag "start" et pour les tâches avec les tags "target" et "distractor". Le code complet de ces nouveaux composants est inclus en annexe de ce manuscrit.

1	1. Click on the START circle.
	2. Click the red target.
		interblock
		`Message( ... )`
		criterion
		`Key(Space)`

2	START	intertrial
		`StartButton`
		criterion
		`PressOnTag(start)`

3		trial
		`ReactionBlock`
		trial's criterion
		`PressOnTag(target)`
		`| (PressOnTag(distractor) => {Distractor})`

4	START	intertrial
		`StartButton`
		criterion
		`PressOnTag(start)`

FIG. 6.16 : Composants de l'expérimentation de Hick-Hyman

En utilisant SwingStates et TouchStone, nous avons pu concevoir une expérimentation simple en environ 80 lignes de code. Le squelette d'implémentation que nous avons fourni illustre la lisibilité et la concision du code correspondant à une expérimentation dans la plateforme d'exécution de TouchStone. Nous avons bénéficié du moteur d'expérience de la plateforme d'exécution pour nous affranchir du code nécessaire à l'enchaînement des tâches et à l'enregistrement des données. Nous avons défini les nouveaux composants dont nous avions besoin (`HickBlock`, `HickIntertitle` et `PressOnTag`) tout en ayant pu réutiliser des composants définis à l'extérieur comme l'intertitre `Message` que nous avions déjà utilisé pour notre expérimentation comparant OZ et SDAZ. Les nouveaux composants que nous avons définis peuvent être exportés sous la forme d'un jar (archive Java) et chargés dans la plateforme de conception afin d'être réutilisés. La facilité de réaliser des expérimentations contrôlées offerte par TouchStone permet, entre autres, de raffiner les prédictions de SimCIS.

Des résultats utiles pour optimiser une action

La section précédente illustrait que la plateforme d'exécution permet de couvrir une large gamme d'expérimentation faisant notamment de TouchStone un outil très utile pour raffiner CIS. Les expérimentations sur les actions motrices de base restent incontournables dans le domaine de l'IHM. Dans le cadre de ce travail de thèse, elles le sont d'autant plus qu'elles peuvent fournir des résultats utiles aux concepteurs d'interfaces pour optimiser les actions des graphes d'interaction CIS. Dans cette section, nous démontrons l'intérêt de TouchStone pour enrichir la base de données des résultats sur les différentes actions motrices en prenant l'exemple du pointage multi-échelle.

La réplication et l'extension d'expérimentations existantes est très difficile en pratique même pour les chercheurs confirmés. Pour comparer une nouvelle technique d'interaction aux techniques existantes, ils doivent non seulement développer leur propre technique mais aussi re-développer les techniques existantes à partir des informations publiées. Cette façon de procéder prend beaucoup de temps et ne garantit pas la généralisabilité des résultats puisque de petits détails d'implémentation peuvent engendrer des conséquences importantes sur les performances. En effet, la plupart du temps, les auteurs d'une nouvelle technique cherchent à limiter le nombre de facteurs pour un meilleur contrôle si bien qu'ils se limitent en général à une comparaison de leur technique à une ou deux autres(s) technique(s) permettant de réaliser la même tâche. Par exemple, nous avions comparé OrthoZoom (OZ) à Speed Dependant Automatic Zooming (SDAZ) sur une tâche de pointage, la technique multi-échelle rapportée la plus efficace qui n'utilise qu'une souris, tandis que Guiard et al. [37] avaient comparé le pointage bimanuel (PZB) au pointage unimanuel (PZ). Pour déplacer le plan, l'utilisateur doit utiliser une interaction de drag avec PZ et PZB et, pour changer le facteur de zoom, il doit utiliser le slider d'un joystick tenu par la main non dominante avec PZB et la molette de la souris tenue par la main dominante avec PZ. Ces quatre techniques (OZ, SDAZ, PZ et PZB) permettent toutes de réaliser du pointage multi-échelle mais, les deux expériences OZ vs. SDAZ et PZ vs. PZB ayant été réalisées dans des conditions différentes, il est impossible pour le concepteur de déterminer laquelle choisir pour obtenir l'interface la plus performante. Dans cette section, nous illustrons comment TouchStone peut faciliter ce processus de réplication de l'expérimentation OZ vs. SDAZ et d'extension avec les conditions PZ et PZB.

Facteurs et stratégie de contrebalancement : Nous avons conduit une expérimentation selon un plan expérimental 4x2x2 intra-participant pour comparer les 4 techniques (OZ, SDAZ, PZ et PZB) pour des

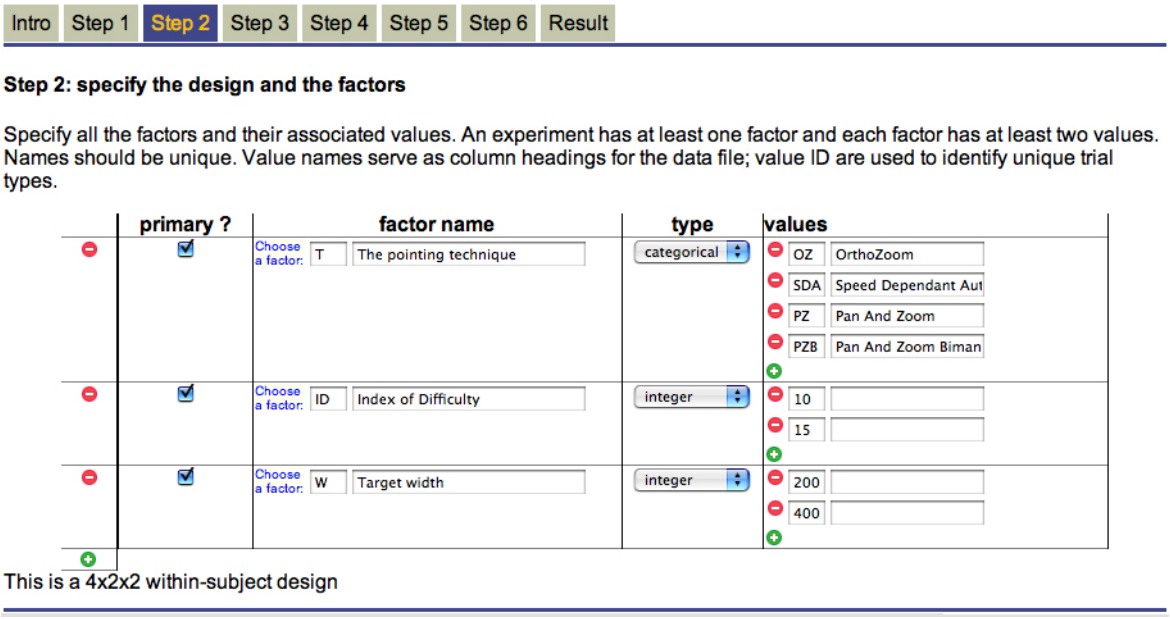


FIG. 6.17 : De la plateforme d'exécution à la plateforme de conception : les deux nouvelles valeurs sont visibles

tâches de pointage multi-échelle. Nous nous sommes contentés de reprendre le plan expérimental OZ vs. SDAZ que nous avions conçu avec TouchStone (cf. section 6.3.2) en ajoutant simplement deux valeurs au facteur `Technique` selon la méthode présentée à la sous-section 6.3.2. La figure 6.17 montre la sélection de ces deux nouvelles valeurs dans la plateforme de conception.

L'ajout de deux nouvelles valeurs rallonge la durée d'exécution de l'expérimentation, nous incitant à changer le plan expérimental sur les conseils de la plateforme de conception. Nous réduisons le nombre de valeurs de `ID` et de `W`, nous ramenons ainsi à un plan contenant 16 types de tâches : 4 `Techniques` (OZ, SDAZ, PZ et PZB) x 2 `IDs` (10 et 15) x 2 `W` (200 et 400). Comme dans le plan OZ vs. SDAZ, nous avons groupé les tâches selon les techniques pour obtenir 4 blocs, un par technique. Toutes les combinaisons de IDxW apparaissaient dans chaque bloc, répliquées 4 fois, pour obtenir des blocs comportant 32 tâches de pointage.

Face au choix d'un tel plan, la plateforme de conception recommande 16 participants pour obtenir un contrebalancement parfait des valeurs des différents facteurs selon un carré latin. Cette condition étant trop contraignante pour nous au moment de la conception de cette expérience, nous avons décidé d'essayer plusieurs alternatives pour le ramener à 12 participants. La plateforme de conception permet, par exemple, d'envisager de supprimer le facteur `W` en lui assignant une valeur constante. Cependant, un tel choix nous amènerait à ne pas prendre en compte l'effet de la largeur de la cible contrairement aux recommandations faites par Hinckley et al. [89] sur l'interaction potentielle entre l'effet d'une technique et l'effet de la largeur de la cible. Une meilleure alternative pour limiter le nombre de participants semble

être la dévaluation de `W` au niveau du bloc. Avant de lancer l'expérimentation, nous avons vérifié la distribution des tâches induite par ce choix pour s'assurer que la stratégie de contrebalancement proposée par la plateforme de conception de TouchStone était satisfaisante. Les tests post-expérimentaux ont de plus rapporté qu'il n'y avait pas d'effet d'ordre, validant ainsi la stratégie proposée.

Mesures : Nous mesurons deux variables dépendantes au niveau d'une tâche : le temps de mouvement `MT` et le succès/échec `HIT`. Un échec se produit lorsque l'utilisateur n'a pas réussi à atteindre la cible au bout de trois minutes (ce qui n'est jamais arrivé au cours de notre expérimentation). Au niveau cinématique, nous enregistrons la position et le facteur de zoom de la vue afin de pouvoir rejouer l'interaction si nécessaire lors des analyses.

Participants : Douze participants, 9 hommes et 3 femmes, agés de 26 ans en moyenne, ont pris part à l'expérimentation.

Équipement : Les deux nouvelles techniques ont été développées dans l'environnement de développement de TouchStone pour concevoir notre plan expérimental grâce à la plateforme de conception en référençant ces nouveaux composants. L'expérimentation s'est déroulée sur un PC équipé d'un processeur 2 GHz Pentium 4 avec un moniteur LCD 1280x1024, une souris optique pour la main dominante et un joystick SideWinder Precision 2 pour la main non-dominante.

Procédure : Pour chaque participant, l'évaluateur commence par expliquer la tâche en deux minutes. Ensuite, chaque bloc est précédé d'un bloc d'entraînement contenant 8 tâches de pointage dont les valeurs des conditions `ID` et `W` sont tirées au hasard. Un intertitre indique au participant d'atteindre la cible le plus vite possible, une nouvelle cible apparaissant chaque fois que la cible courante vient d'être atteinte. Le scénario au sein d'un bloc était identique à celui d'un bloc lors de l'expérimentation PZ vs. OZ, il est rappelé par à la figure 6.10.

Hypothèses : Nous nous attendons à des résultats cohérents avec ceux rapportés dans la littérature. Rappelons que les expérimentations originales OZ vs. SDAZ [15] et PZ vs. PZB [37] rapportent que, pour le temps d'exécution d'une tâche de pointage, les performances relatives sont : OZ $<$ SDAZ (H_1) et PZB $<$ PZ (H_2) . Cependant, nous n'avons pas d'hypothèses sur les comparaisons entre les autres paires, ce qui est la motivation de cette expérimentation.

Résultats : Comme dans l'expérimentation OZ vs. SDAZ, l'analyse de variance n'a pas révélé d'interaction significative entre les effets des facteurs `Technique` et `W` sur le temps de mouvement `MT` ($F_{3,173} < 1$). Nous nous assurons également qu'il n'y a pas eu d'effet de l'ordre de présentation des techniques : l'effet du numéro de bloc et l'effet de la technique n'ont pas d'interaction significative. L'analyse de l'effet du numéro de tâche au sein d'un bloc ne montre un effet d'apprentissage que pour la technique SDAZ ($F_{3,177} = 3.36$ et $p = 0.02$).

L'analyse de variance en utilisant la méthode REML a révélé un effet significatif de la `Technique` sur le temps de mouvement `MT` ($F_{3,177} = 44.9$ et $p < 0.0001$) et un effet significatif de l'indice de difficulté `ID` sur `MT` ($F_{3,173} = 0.2$ et $p = 0.9$). Le test post-hoc de Tukey nous a permis d'établir l'ordre suivant entre techniques pour le temps d'exécution : OZ $<$ PZB $<$ {PZ $\simeq$ SDAZ} (Figure 6.18).

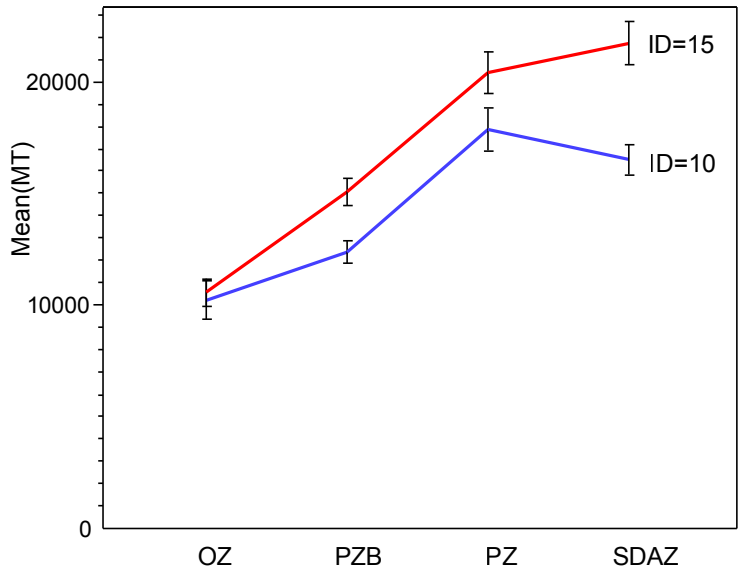


FIG. 6.18 : PZ vs. PZB vs OZ. vs. SDAZ : Temps de mouvement moyen en s.

Ces résultats supportent nos hypothèses H_1 et H_2 et fournisse un résultat plus général pour comparer les quatre techniques pour une tâche de pointage multi-échelle en 1 dimension. Ces résultats montrent également que SDAZ semble plus sensible à l'effet de l'`ID` que les autres techniques. En exportant le code de ces deux techniques sous forme de composants d'expérimentation, TouchStone nous a permis de répliquer et étendre une expérience afin de fournir des résultats plus généraux et plus lisibles pour les concepteurs d'interfaces graphiques. De plus, nous avons mis à disposition dans la plateforme les composants que nous avons réalisé pour cette expérimentation afin d'encourager d'autres chercheurs à répliquer et étendre ces études.

6.5 Conclusion

Dans ce chapitre nous avons présenté la plateforme d'expérimentation TouchStone, un projet dans lequel plusieurs membres de l'équipe in|situ| sont impliqués, en insistant tout particulièrement sur la description de la plateforme d'exécution sur laquelle j'ai concentré mes recherches.

TouchStone est un outil pour guider les chercheurs dans la réalisation d'expérimentations contrôlées ayant pour but d'augmenter le corpus des résultats empiriques ainsi que la lisibilité des plans expérimentaux et de leurs résultats pour une plus large communauté de concepteurs d'interfaces. TouchStone permet aux évaluateurs de concevoir et réaliser leurs expériences dans un format encourageant les recherches autour de ces résultats. TouchStone est organisé en deux principales plateformes (conception et exécution), toutes deux modulaires, pour faciliter la réutilisation de composants. TouchStone est un système en ligne et basé sur une architecture modulaire pour permettre aux évaluateurs de réutiliser les composants existants et de partager ceux qu'ils réalisent. Ainsi, TouchStone nous a permis de réaliser à

moindre coût des expérimentations en réutilisant des plans expérimentaux et des composants d'expérimentation existants et de rapporter ainsi de nouveaux résultats. Nous espérons que ce système partagé permettra à TouchStone de devenir une base de données de plans expérimentaux, de composants d'expérimentation et de résultats d'expérimentation pour améliorer la qualité de l'état de l'art en IHM et faciliter ainsi les choix des concepteurs d'interfaces. L'expérimentation que nous avons réalisée avec TouchStone illustre cette propriété : en répliquant et unifiant deux expérimentations publiées antérieurement et dans des conditions expérimentales différentes, nous fournissons un résultat sur la comparaison des différentes techniques de pointage multi-échelle plus général et plus facile à lire.

Conclusion et Perspectives

Le but de cette thèse était de former les bases d'une approche générative de la conception d'applications graphiques interactives. Nous avons proposé une approche longitudinale allant de la modélisation de l'interaction avec CIS jusqu'aux outils de mise en œuvre avec SwingStates en passant par l'évaluation avec TouchStone. Nous avons montré que ces outils permettent aux concepteurs d'interfaces graphiques de faire des choix informés dans un espace de conception plus balisé et aux développeurs de les mettre en œuvre avec leurs outils de développement habituels. Pour conclure ce manuscrit, nous rappelons les contributions de cette thèse, les caractéristiques des outils développés et la façon dont ils s'articulent dans le processus de conception des applications graphiques. Enfin, nous identifions deux pistes précises pour des travaux futurs.

7.1 Les contributions

7.1.1 Complexity of Interaction Sequences

Le problème En amont de la conception d'applications graphiques, il s'agit d'identifier les alternatives possibles et de ne garder que celles qui semblent prometteuses. Les modèles de l'interaction disponibles actuellement ne permettent pas de baliser un espace de conception dans lequel décrire différentes possibilités et les évaluer. Soit ils sont d'un niveau d'abstraction trop faible pour être utilisable pour faire des choix préliminaires, soit ils ne permettent pas de répondre à la simple question "Laquelle de ces techniques est la plus efficace dans ce contexte d'utilisation ?".

La contribution Complexity of Interaction Sequences (CIS) [13], est un modèle pour aider le concepteur à explorer les différentes possibilités qui s'offrent à lui et faire des choix informés dans les stades préliminaires de conception d'applications graphiques. CIS identifie un espace de conception dans lequel décrire des techniques d'interaction à un niveau d'abstraction élevé qui est approprié pour les phases en amont de la conception. CIS permet d'évaluer ces différentes possibilités en prédisant leur efficacité au sein d'une séquence d'interaction qui opérationnalise un contexte d'utilisation réel. De plus, la structure morphologique d'une technique exhibée par CIS permet d'identifier les briques de base d'une technique pour les optimiser avec l'aide de l'entrepôt de résultats proposé par TouchStone. Ainsi nous avons conçu deux nouvelles techniques : OrthoZoom [15] pour optimiser les pointages multi-échelle et ControlTree [14] pour optimiser la sélection d'un nœud dans une très grande représentation nœud-lien.

L'évaluation Afin de valider CIS, nous avons réalisé une expérimentation contrôlée afin de comparer les prédictions de CIS à des observations empiriques. Cette expérimentation compare trois techniques ayant des structures CIS différentes sur quatre contextes d'utilisation. Les différences empiriques entre techniques et entre contextes sont les mêmes que les différences prédites. Cependant, les performances absolues se sont révélées sous-estimées par les prédictions de CIS. Nous avons donc profité de la collecte de ces résultats empiriques pour mieux calibrer les différents coefficients empiriques des lois sur lesquelles reposent CIS et avons ainsi amélioré la finesse des prédictions. Ces coefficients étant dépendants des conditions d'utilisation (des périphériques d'entrée utilisés par exemple), nous avons utilisé TouchStone et SwingStates pour développer des expérimentations permettant de collecter facilement les données empiriques nécessaires à l'ajustement de ces coefficients.

7.1.2 SwingStates

Le problème D'un côté, les boîtes à outils qui sont largement utilisées pour développer des applications graphiques, comme Java Swing [114], Gtk [113] ou Qt [61] ont été spécialement conçues pour la construction d'interfaces en assemblant des widgets dont la cohérence doit être assurée à travers des spaghettis de callbacks [128]. De l'autre côté, de nombreuses boîtes à outils expérimentales ont été développées pour faciliter la programmation de l'interaction avancée mais très peu de développeurs les utilisent pour les applications courantes.

La contribution SwingStates [11, 12] est une boîte à outils qui introduit dans le langage Java un modèle de dessin et des structures de contrôle adaptées à la programmation de techniques d'interaction avancées, les machines à états. Par rapport aux autres boîtes à outils expérimentales, SwingStates est une extension de Java Swing, une boîte à outils largement utilisée pour le développement d'interfaces graphiques, afin de faciliter son adoption par les développeurs. SwingStates permet de modifier l'interaction de widgets Swing prédéfinis et de définir de nouveaux widgets qui s'intègrent parfaitement dans n'importe quelle interface Swing. Les machines à états, dont la syntaxe reste naturelle tout en étant directement interprétable par la machine virtuelle Java, permettent de spécifier l'interaction facilement. Elle offre donc plus de possibilités aux développeurs tout en leur permettant de rester dans leur cadre de travail habituel.

L'évaluation Afin de valider SwingStates, nous l'avons soumis à un usage *in situ* en utilisant une approche par benchmark. Nos étudiants de master l'ont utilisée pour implémenter un éventail de techniques publiées dans les quinze dernières années et représentatives de l'interaction dite post-WIMP. Les projets qu'ils ont réalisé sont fonctionnels et nous continuons à utiliser SwingStates pour l'enseignement de l'IHM aux étudiants de Master à l'Université Paris-Sud depuis maintenant deux ans.

7.1.3 TouchStone

Le problème De nombreuses techniques d'interaction naissent dans les laboratoires de recherche et sont évaluées par des expérimentations contrôlées. Les difficultés rencontrées par les évaluateurs lors de la réalisation d'expérimentations contrôlées favorisent des plans expérimentaux simples et ad hoc ne comparant la nouvelle technique qu'à la technique "standard" sur une tâche spécifique. Le concepteur se trouve donc face à une littérature dans laquelle il est difficile d'identifier la technique qui serait la plus efficace pour son problème réel.

La contribution TouchStone [115] est un outil pour assister les chercheurs dans la réalisation d'expérimentations contrôlées. La plateforme de conception guide l'évaluateur étape par étape afin de proposer des standards pour la réalisation et la présentation de plans expérimentaux. Nos travaux ont porté sur la plateforme d'exécution qui facilite le développement des composants logiciels requis par une expérimentation et la réutilisation de composants existants grâce à son architecture modulaire. L'assistance et le caractère exploratoire de la plateforme de conception et la modularité de la plateforme d'exécution favorisent non seulement l'extension ou la réplication d'expérimentations existantes mais aussi la réalisation de nouvelles. TouchStone a donc pour but d'augmenter le corpus des résultats empiriques et de fournir un guide de lecture de ces résultats accompagnés de leurs plans expérimentaux.

L'évaluation Afin de valider TouchStone, nous avons réalisé une expérimentation qui unifie les résultats de deux expérimentations antérieures : (i) OrthoZoom vs. Speed Dependant Automatic Zooming et (ii) Pan Zoom standard vs. Pan Zoom bi-manuel. L'expérimentation (i) est celle que nous avions mené pour évaluer OrthoZoom qui s'est avéré être deux fois plus efficace que Speed Dependant Automatic Zooming, la technique la plus efficace auparavant. Nous avons également montré l'utilité de la plateforme d'exécution en réalisant un code concis pour une expérimentation de type Hick-Hyman. Pour faciliter la réalisation d'expérimentations contrôlées, CIS peut venir assister TouchStone pour le choix des tâches expérimentales valides en envisageant différentes séquences d'interaction et en jugeant de leur pertinence grâce à SimCIS.

7.1.4 Ce qu'il manque...

Les perspectives de travaux futurs sont nombreuses. Nous nous focalisons ici sur la couverture de l'approche proposée dans cette thèse en nous posant les deux questions suivantes :

1. Peut-on programmer avec SwingStates tout ce que CIS permet de modéliser ?
2. Peut-on modéliser avec CIS tout ce que SwingStates permet de programmer ?

Sans chercher à identifier les deux ensembles "Ce que CIS permet de modéliser" et "Ce que SwingStates permet de programmer", nous pouvons donner deux exemples rencontrés pendant nos travaux qui montrent que les deux réponses sont négatives :

1. CIS permet de modéliser l'interaction bi-manuelle alors que SwingStates ne gère que les configurations d'entrée standard clavier+souris.
2. SwingStates permet de programmer des interfaces gestuelles alors que CIS ne permet pas de les modéliser actuellement.

Dans les deux sections suivantes, nous abordons chacun de ces deux points de façon plus détaillée.

7.2 SwingStates et l'entrée généralisée

Nous avons vu que CIS permet de modéliser des techniques qui requièrent des entrées non standard comme la toolglass qui nécessite de pouvoir gérer deux dispositifs de pointage. Or, SwingStates a été construit au-dessus du système d'évènements `java.awt.event` qui ne permet pas de dissocier les différents dispositifs de pointage ou les différents claviers. En effet, même si un stylet et une souris sont disponibles dans la configuration d'entrée, un déplacement du stylet ou de la souris provoquera le même événement générique `mouseMoved`.

La plateforme d'exécution de TouchStone, qui a été développée après SwingStates, permet l'entrée généralisée grâce à l'utilisation de la librairie JInput. TouchStone propose un objet, l'`InputManager`, qui est capable de générer des événements de la forme `axesChanged` et auquel il est possible de demander à chaque instant quels axes ont effectivement changé et quelles sont leurs valeurs actuelles. N'importe quel objet qui implémente l'interface `AxesListener` peut s'abonner auprès de cet `InputManager` pour entendre les changements sur un ensemble d'axes. Une façon simple d'intégrer l'entrée généralisée dans SwingStates est de faire implémenter cette interface à nos machines à états et de définir un nouveau type de transition `AxesChanged` qui puisse être déclenchée lors du changement de la valeur d'un axe.

Il serait cependant intéressant de proposer une intégration plus forte avec les types de machines existants afin de proposer des machines plus expressives. Ainsi, il faudrait pouvoir d'utiliser toutes les transitions prédéfinies de SwingStates (`Press*`, `Drag*`, `Key*`, etc.) et de pouvoir les paramétrer par un dispositif de pointage pour les transitions pour l'instant dédiées à la souris ou par un dispositif d'entrée de texte pour les transitions pour l'instant dédiées au clavier. Ceci nécessite de modifier le mécanisme de *picking* actuel de SwingStates pour qu'il puisse prendre en compte un ensemble de *pickers*. Chaque picker devant avoir deux axes *x* et *y* pour déclencher des transitions de type `Move*` et un axe *left* afin que les combinaisons de valeurs (*x*, *y*, *left*) nous permettent de déclencher des transitions de type `Press*`, `Drag*`, etc.

Cette piste de travail va être abordée très prochainement. L'utilisation d'entrées généralisées permettra d'utiliser SwingStates dans des environnements plus divers que le simple "desktop" (PDAs, tableaux interactifs, tables interactives, etc.), ce qui amène naturellement à se poser la question du support à l'interaction collaborative ("single display groupware" par exemple [165]) et/ou multi-surface (interactions de type pick-and-drop" [154]). Dans de tels contextes, d'autres modes de composition des machines à états devront peut-être enrichir les compositions parallèles et séquentielles que nous avons étudiées. Dans le même souci d'améliorer l'approche générative en augmentant la couverture des outils proposés ici, nous envisageons d'étendre le modèle CIS en commençant par étudier les lois empiriques sous-jacentes à l'interaction basée sur la reconnaissance de gestes.

7.3 CIS et l'interaction gestuelle

À l'heure actuelle, CIS permet de décrire et d'évaluer des techniques d'interaction pour des interfaces graphiques dans lesquelles les objets d'interaction sont affichés et manipulés (P). Par exemple, CIS permet de modéliser les techniques courantes basées sur le pointage mais aussi des interfaces plus novatrices comme celles qui sont basées sur le franchissement de but ou sur le suivi de chemin. Toutes ces interfaces sont modélisées par des suites de choix, acquisitions et validations. Dans ces interfaces, la propriété de visibilité des objets d'interaction suggère à l'utilisateur les interactions possibles (interfaces à feed-forward) et nous permet donc de négliger les effets cognitifs complexes tout en gardant des prédictions proches de l'usage réel. Les choix visuels sont évalués grâce à la loi de Hick-Hyman [88, 94] tandis que les actions motrices sont évaluées grâce à la loi de Fitts [64] et aux lois de franchissement et de suivi de chemin [4, 2]. Cependant, nous avons vu au cours de ce manuscrit que, pour l'instant, CIS ne permet pas de décrire les interfaces gestuelles alors que nous avons présenté un programme réalisé avec SwingStates permettant de copier, couper et coller des formes graphiques avec des commandes gestuelles.

La couverture de CIS dépend donc des lois empiriques sous-jacentes aux sélections et actions de CIS et elle pourrait être étendue par la connaissance d'un plus grand nombre de lois empiriques. Nous avons déjà étendu CIS au cas particulier des interfaces multi-échelle (ou interfaces zoomables). Dans ce cas, le choix d'un objet par l'utilisateur ne relève pas uniquement de la perception visuelle mais d'un ensemble d'actions motrices pour naviguer et rendre visible les objets graphiques [141] alors que, bien qu'étant dans un espace multi-échelle, la loi de Fitts sous-tend toujours la navigation vers un objet donné [78].

D'une façon similaire, nous projetons d'étendre CIS afin de pouvoir modéliser les interfaces gestuelles, cadre dans lequel la propriété (P) n'est plus valide. Récemment, Cao et Zhai [44] ont démontré empiriquement qu'un geste composite peut-être décrit par un ensemble de traits primitifs (courbes, droites et angles) et que le temps pour exécuter ce geste peut être prédit en sommant le temps d'exécution de chacun des traits primitifs. Alors que ce résultat fournit une loi motrice pour un geste isolé, CIS requiert encore une loi pour évaluer le *temps de choix d'un geste*. Alors que certains concepts ont une représentation assez directe (dessiner un chiffre pour entrer la valeur numérique d'une commande par exemple), d'autres n'ont pas une association aussi directe (dessiner un symbole représentant une commande pour copier ou coller par exemple) [1]. Pour le premier cas, nous pouvons considérer un temps de choix quasi nul alors que, dans le second cas, l'utilisateur doit au préalable identifier le bon geste avant d'exécuter les actions motrices nécessaires au dessin. La création d'une loi de choix d'un geste requiert :

1. L'identification des variables qui influencent le temps de choix.
2. La construction d'un cadre expérimental pour tester l'effet de ces variables.

Une première étape serait de tester l'effet des deux variables suivantes : la taille du vocabulaire (c'est-à-dire le nombre de gestes) et les distances entre paires de gestes. Pour définir la distance entre deux gestes, nous pourrions utiliser l'algorithme de Rubine pour la reconnaissance de gestes [156], qui apprend un vocabulaire à partir d'un ensemble d'exemples. Cet algorithme définit un geste comme un vecteur de caractéristiques (angle de départ, taille de la boîte englobante, etc.) et utilise la distance entre ces vecteurs pour construire les classes de gestes, puis pour classifier un nouveau geste dans une des classes ainsi définies. Pour tester l'effet de ces deux variables, il nous faut opérationnaliser la notion de tâche de choix d'un geste, c'est-à-dire la définir comme une fonction des variables identifiées précédemment vers des mesures de performance. Une première idée consiste à demander à l'utilisateur d'apprendre un ensemble d'associations couleur/geste et de mesurer le temps d'exécution et le nombre d'erreurs lorsque l'utilisateur doit accomplir le geste correct en réponse à un stimulus coloré. La difficulté ne réside pas tant dans la définition de la tâche en elle-même mais dans le contrôle des effets qui peuvent intervenir pour bruiter nos observations. En effet, choisir un geste parmi un ensemble de gestes mémorisés dépend de la mémoire de chaque utilisateur qui ne peut être contrôlée a priori. C'est pourquoi notre plan expérimental devra limiter l'impact de ce facteur afin d'attribuer les effets observés aux seules variables testées (taille du vocabulaire et distance entre deux gestes).

Ces résultats peuvent avoir des implications directes pour la conception d'interfaces pour les utilisateurs en fournissant des guides pour construire un "bon" vocabulaire pour les commandes abstraites disponibles dans une application. En particulier, nous essaierons de répondre aux questions suivantes :

- Est-ce que la notion de geste pré-attentif au sein d'un vocabulaire existe (un geste qui est mémorisé plus facilement que les autres, par exemple celui qui est le plus éloigné de tous les autres gestes du vocabulaire) ? Si oui, les concepteurs d'interface y gagneraient à l'attribuer à la commande la plus fréquente.
- Est-ce que le vocabulaire doit être épars (c'est-à-dire une large distance entre chaque paire de gestes) ou compact ? Est-ce que cette dernière propriété doit être liée à la taille du vocabulaire ? Les réponses à ces questions permettraient d'évaluer la qualité d'un vocabulaire dans son ensemble.

7.4 À plus long terme

Les perspectives immédiates portent donc sur l'extension de la couverture de l'approche générative proposée dans ce manuscrit en restant dans le même champ d'application, celui des applications graphiques avec un seul utilisateur. Bien qu'étant le plus répandu, ce style d'interfaces n'est pas l'unique existant. À plus long terme, il serait donc intéressant de voir comment une telle approche générative peut passer à l'échelle pour couvrir un spectre plus large des interfaces existantes : tangibles, ambiantes, collaboratives, multi-surface, etc.

Sur le plan plus théorique, nous projetons d'approfondir la notion de sensibilité au contexte d'une technique par l'étude de sa complexité afin qu'une technique puisse être évaluée selon sa complexité au mieux, au pire et en moyenne à la manière d'un algorithme. À l'heure actuelle, les cas au mieux et au pire sont donnés de manière ad hoc et nous n'étudions pas la complexité en moyenne. Ce travail nécessite de pouvoir caractériser la distribution des données d'entrée, c'est-à-dire des différentes interactions des utilisateurs.

Bibliographie

[1] Jr. A. C. Long, J. A. Landay, and L. A. Rowe. Implications for a gesture design tool. In *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 40–47, New York, NY, USA, 1999. ACM Press.

[2] J. Accot and S. Zhai. Beyond Fitts' law : models for trajectory-based HCI tasks. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 295–302, 1997.

[3] J. Accot and S. Zhai. Performance evaluation of input devices in trajectory-based tasks : an application of the steering law. In *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 466–472, New York, NY, USA, 1999. ACM Press.

[4] J. Accot and S. Zhai. More than dotting the i's—foundations for crossing-based interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems : Changing our world, changing ourselves*, pages 73–80, 2002.

[5] J. Accot and S. Zhai. Refining fitts' law models for bivariate pointing. In *CHI '03 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 193–200, New York, NY, USA, 2003. ACM Press.

[6] C. Ahlberg and B. Shneiderman. The alphaslider : a compact and rapid selector. In *CHI '94 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 365–371, New York, NY, USA, 1994. ACM Press.

[7] C. Ahlberg, C. Williamson, and B. Shneiderman. Dynamic queries for information exploration : an implementation and evaluation. In *CHI '92 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 619–626, New York, NY, USA, 1992. ACM Press.

[8] R. St. Amant and T.E. Horton. Characterizing tool use in an interactive drawing environment. In *SMARTGRAPH '02 : Proceedings of the 2nd international symposium on Smart graphics*, pages 86–93, New York, NY, USA, 2002. ACM Press.

[9] J.R. Anderson. ACT : A simple theory of complex cognition. *American Psychologist*, 51(4) :355–365, 1996.

[10] G. Apitz and F. Guimbretière. CrossY : a crossing-based drawing application. *Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 3–12, 2004.

[11] C. Appert and M. Beaudouin-Lafon. Smcanvas : augmenter la boîte à outils java swing pour prototyper des techniques d'interaction avancées. In *IHM '06 : Proceedings of the 18th international conference on Association Francophone d'Interaction Homme-Machine*, pages 99–106, New York, NY, USA, 2006. ACM Press.

[12] C. Appert and M. Beaudouin-Lafon. Swingstates : adding state machines to the swing toolkit. In *UIST '06 : Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 319–322, New York, NY, USA, 2006. ACM Press.

[13] C. Appert, M. Beaudouin-Lafon, and W.E. Mackay. Context matters : Evaluating interaction techniques with the CIS model. In *Proc. of HCI 2004, Leeds, UK*, pages 279–295. Springer Verlag, September 2004.

[14] C. Appert and J.D. Fekete. Controltree : Navigating and selecting in a large tree. In *UIST '06 : Proceedings of the 19th annual ACM symposium on User interface software and technology*, pages 47–48, 2006.

[15] C. Appert and J.D. Fekete. Orthozoom scroller : 1d multi-scale navigation. In *CHI '06 : Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 21–30, New York, NY, USA, 2006. ACM Press.

[16] R. Baecker and I. Small. Animation at the interface. *The Art of Human-Computer Interface Design*, pages 251–267, 1990.

[17] K.B. Bærentsen. Intuitive User Interfaces. *Scandinavian Journal of Information Systems*, 12 :29–60, 2000.

[18] R. Balakrishnan. "Beating" Fitts' law : virtual enhancements for pointing facilitation. *International Journal of Human-Computer Studies*, 61(6) :857–874, 2004.

[19] L.J. Bannon. From human factors to human actors : the role of psychology and human-computer interaction studies in system design. *Design at work : cooperative design of computer systems table of contents*, pages 25–44, 1992.

[20] L.J. Bannon and S. Bødker. *Beyond the Interface : Encountering Artifacts in Use*. Computer Science Department, Aarhus Universitet, 1991.

[21] R. Bastide, D. Navarre, and P. Palanque. A model-based tool for interactive prototyping of highly interactive applications. In *CHI '02 : CHI '02 extended abstracts on Human factors in computing systems*, pages 516–517, New York, NY, USA, 2002. ACM Press.

[22] P. Baudisch. Don't click, paint ! using toggle maps to manipulate sets of toggle switches. In *UIST '98 : Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 65–66, New York, NY, USA, 1998. ACM Press.

[23] P. Baudisch, E. Cutrell, D. Robbins, M. Czerwinski, P. Tandler, B. Bederson, and A. Zierlinger. Drag-and-Pop and Drag-and-Pick : Techniques for Accessing Remote Screen Content on Touch- and Pen-operated Systems. *Proceedings of Interact*, 3 :57–64, 2003.

[24] M. Beaudouin-Lafon. Instrumental interaction : an interaction model for designing post-WIMP user interfaces. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, 2000.

[25] M. Beaudouin-Lafon. Designing interaction, not interfaces. In *AVI '04 : Proceedings of the working conference on Advanced visual interfaces*, pages 15–22, New York, NY, USA, 2004. ACM Press.

[26] M. Beaudouin-Lafon and H.M. Lassen. The architecture and implementation of cpn2000, a post-wimp graphical application. In *UIST '00 : Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 181–190, New York, NY, USA, 2000. ACM Press.

[27] M. Beaudouin-Lafon and W. E. Mackay. Reification, polymorphism and reuse : three principles for designing visual interfaces. In *AVI '00 : Proceedings of the working conference on Advanced visual interfaces*, pages 102–109, New York, NY, USA, 2000. ACM Press.

[28] B. Bederson, J. Grosjean, and J. Meyer. Toolkit design for interactive structured graphics. *IEEE Trans. Softw. Eng.*, 30(8) :535–546, 2004.

[29] B. B. Bederson, J. D. Hollan, A. Druin, J. Stewart, D. Rogers, and D. Proft. Local tools : an alternative to tool palettes. In *UIST '96 : Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 169–170, New York, NY, USA, 1996. ACM Press.

[30] E. A. Bier, M. C. Stone, K. Pier, W. Buxton, and T. D. DeRose. Toolglass and magic lenses : the see-through interface. In *SIGGRAPH '93 : Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80, New York, NY, USA, 1993. ACM Press.

[31] R. Blanch. Architecture logicielle et outils pour les interfaces hommes-machines graphiques avancees. *These de doctorat, Universite Paris-Sud XI Orsay*, 2005.

[32] R. Blanch and M. Beaudouin-Lafon. Programming rich interactions using the hierarchical state machine toolkit. In *AVI '06 : Proceedings of the working conference on Advanced visual interfaces*, pages 51–58, New York, NY, USA, 2006. ACM Press.

[33] R. Blanch, Y. Guiard, and M. Beaudouin-Lafon. Semantic pointing : improving target acquisition with control-display ratio adaptation. In *CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 519–526, New York, NY, USA, 2004. ACM Press.

[34] T. Bleser and J. Sibert. Toto : a tool for selecting interaction techniques. In *UIST '90 : Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, pages 135–142, New York, NY, USA, 1990. ACM Press.

[35] S. Bødker. *Through the Interface : A Human Activity Approach to User Interface Design*. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 1991.

[36] F. Bourgeois. Le pointage multi-échelles : analyse cinématique de la navigation dans les environnements d'information. *Université de la Méditerranée de Marseills*, 2002.

[37] F. Bourgeois, Y. Guiard, and M. Beaudouin Lafon. Pan-zoom coordination in multi-scale pointing. In *CHI '01 : CHI '01 extended abstracts on Human factors in computing systems*, pages 157–158, New York, NY, USA, 2001. ACM Press.

[38] J. K. M. Brown. Edgar. `http://www.uea.ac.uk/nrp/jic/edgar`.

[39] W. Buxton. Lexical and pragmatic considerations of input structures. *ACM SIGGRAPH Computer Graphics*, 17(1) :31–37, 1983.

[40] W. Buxton. A three-state model of graphical input. In *INTERACT '90 : Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 449–456. North-Holland, 1990.

[41] F. Bérard. Gmlcanvas. `http://iihm.imag.fr/projects/gml`.

[42] J. Callahan, D. Hopkins, M. Weiser, and B. Shneiderman. *An empirical comparison of pie vs. linear menus*. ACM Press New York, NY, USA, 1988.

[43] G. Calvary, J. Coutaz, and L. Nigay. From single-user architectural design to PAC* : a generic software architecture model for CSCW. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 242–249, 1997.

[44] X. Cao and S. Zhai. Modeling human performance of pen stroke gestures. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*. In press, 2007.

[45] S. K. Card, W. K. English, and B. J. Burr. *Evaluation of mouse, rate-controlled isometric joystick, step keys, and text keys, for text selection on a CRT*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[46] S.K. Card. *The Psychology of Human-Computer Interaction*. Lawrence Erlbaum Assoc Inc, 1983.

[47] S.K. Card, J.D. Mackinlay, and G.G. Robertson. A morphological analysis of the design space of input devices. *ACM Trans. Inf. Syst.*, 9(2) :99–122, 1991.

[48] S.K. Card, T.P. Moran, and A. Newell. The keystroke-level model for user performance time with interactive systems. *Commun. ACM*, 23(7) :396–410, 1980.

[49] S.K. Card, T.P. Moran, and A. Newell. The model human processor- An engineering model of human performance. *Handbook of perception and human performance.*, 2 :45–1, 1986.

[50] M. S. T. Carpendale and C. Montagnese. A framework for unifying presentation space. In *UIST '01 : Proc. ACM Symposium on User Interface Software and Technology*, pages 61–70. ACM Press, 2001.

[51] D.A. Carr. Specification of interface interaction objects. In *CHI '94 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 372–378, New York, NY, USA, 1994. ACM Press.

[52] Olivier Chapuis and Nicolas Roussel. Metisse is not a 3D desktop ! In *UIST '05 : Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 13–22, New York, NY, USA, 2005. ACM Press.

[53] S. Chatty. La Construction d'Interfaces Homme-Machine Animées. *These de doctorat, Universite Paris-Sud XI Orsay*, 1992.

[54] A. Cockburn and J. Savage. Comparing Speed-Dependent Automatic Zooming with Traditional Scroll, Pan and Zoom Methods. *People and Computers XVII (Proceedings of the 2003 British Computer Society Conference on Human-Computer Interaction.)*, pages 87–102, 2003.

[55] M. Collomb, M. Hascoët, P. Baudisch, and B. Lee. Improving drag-and-drop on wall-size displays. *Proceedings of the 2005 conference on Graphics interface*, pages 25–32, 2005.

[56] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan. Context is key. *Communications of the ACM*, 48(3) :49–53, 2005.

[57] S.A. Douglas, A.E. Kirkpatrick, and I.S. MacKenzie. Testing pointing device performance and user assessment with the iso 9241, part 9 standard. In *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 215–222, New York, NY, USA, 1999. ACM Press.

[58] P. Dragicevic. Un modele d ?interaction en entree pour des systemes interactifs multi-dispositifs hautement configurables. *These de doctorat, Universite de Nantes*, 2004.

[59] P. Dragicevic and J.D. Fekete. Input device selection and interaction configuration with icon. In *People and Computer XV - Interaction without frontier (Joint proceedings of HCI 2001 and IHM 2001)*, pages 543–558. Spring Verlag, 2001.

[60] J. Eisenstein and W. Mackay. Interacting with communication appliances : an evaluation of two computer vision-based selection techniques. In *CHI '06 : Proceedings of the SIGCHI conference on Human Factors in computing systems*, pages 1111–1114, New York, NY, USA, 2006. ACM Press.

[61] E. Eng. Qt GUI Toolkit : Porting graphics to multiple platforms using a GUI toolkit. *Linux Journal*, 1996(31es), 1996.

[62] J.D. Fekete. The InfoVis Toolkit. *InfoVis'04*, pages 167–174, 2004.

[63] J.D. Fekete. The InfoVis Toolkit. *InfoVis'04*, pages 167–174, 2004.

[64] P.M. Fitts. The Information Capacity of the Human Motor System in Controlling the Amplitude of Movement. *Journal of Experimental Psychology*, 47 :381–391, 1954.

[65] J.D. Foley and V.L. Wallace. The art of natural graphic man ?Machine conversation. *Proceedings of the IEEE*, 62(4) :462–471, 1974.

[66] J.D. Foley, V.L. Wallace, and P. Chan. The human factors of computer graphics interaction techniques. *IEEE Comput. Graph. Appl.*, 4(11) :13–48, 1984.

[67] G. Furnas and B. Bederson. Space-scale diagrams : understanding multiscale interfaces. In *CHI '95 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 234–241, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[68] K. Gajos and D.S. Weld. SUPPLE : Automatically Generating User Interfaces. *Proceedings of IUI-2004*, 2004.

[69] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1995.

[70] J.J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Assoc Inc, 1987.

[71] P. Gray, J. Goodman, and J. Macleod. A Lightweight Experiment Management System for Handheld Computers. In *CADUI 2004*, Madeira, Portugal, 2004.

[72] T.R.G. Green. Instructions and descriptions : some cognitive aspects of programming and similar activities. In *AVI '00 : Proceedings of the working conference on Advanced visual interfaces*, pages 21–28, New York, NY, USA, 2000. ACM Press.

[73] T.R.G. Green and A. Blackwell. Cognitive Dimensions of Information Artefacts : a tutorial. *Web-based document. URL : http ://www. ndirect. co. uk/thomas. green/workStuff/Papers*, 1998.

[74] T. Grossman and R. Balakrishnan. The bubble cursor : enhancing target acquisition by dynamic resizing of the cursor's activation area. In *CHI '05 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 281–290, New York, NY, USA, 2005. ACM Press.

[75] T. Grossman and R. Balakrishnan. A probabilistic approach to modeling two-dimensional pointing. *ACM Trans. Comput.-Hum. Interact.*, 12(3) :435–459, 2005.

[76] Y. Guiard. Asymmetric Division of Labor in Human Skilled Bimanual Action : The Kinematic Chain as a Model. *Journal of Motor Behavior*, 19(4) :486–517, 1987.

[77] Y. Guiard and M. Beaudouin-Lafon. Target acquisition in multiscale electronic worlds. *International Journal of Human-Computer Studies*, 61(6) :875–905, 2004.

[78] Y. Guiard, M. Beaudouin-Lafon, J. Bastin, D. Pasveer, and S. Zhai. View size and pointing difficulty in multi-scale navigation. *Proceedings of the working conference on Advanced visual interfaces*, pages 117–124, 2004.

[79] Y. Guiard, M. Beaudouin-Lafon, Y. Du, C. Appert, J.D. Fekete, and Olivier Chapuis. Shakespeare's complete works as a benchmark for evaluating multiscale document navigation techniques. In *BELIV '06 : Proceedings of the 2006 AVI workshop on BEyond time and errors*, pages 1–6, New York, NY, USA, 2006. ACM Press.

[80] Y. Guiard, M. Beaudouin-Lafon, and D. Mottet. Navigation as multiscale pointing : extending fitts' model to very high precision tasks. In *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 450–457, New York, NY, USA, 1999. ACM Press.

[81] Y. Guiard, R. Blanch, and M. Beaudouin-Lafon. Object pointing : a complement to bitmap pointing in guis. In *GI '04 : Proceedings of the 2004 conference on Graphics interface*, pages 9–16, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2004. Canadian Human-Computer Communications Society.

[82] Y. Guiard, F. Bourgeois, D. Mottet, and M. Beaudouin-Lafon. Beyond the 10-bit barrier : Fitts' law in multiscale electronic worlds. *Proc IHM-HCI 2001*, pages 573–587, 2001.

[83] Y. Guiard, Y. Du, and O. Chapuis. Quantifying degree of goal directedness in document navigation : Application to the evaluation of the perspective-drag technique. In *CHI '07 : Proc. Human Factors in Computing Systems*. ACM Press, 2007. This volume.

[84] F. Guimbretière and T. Winograd. Flowmenu : combining command, text, and data entry. In *UIST '00 : Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 213–216, New York, NY, USA, 2000. ACM Press.

[85] C. Gutwin. Improving focus targeting in interactive fisheye views. In *CHI '02 : Proc. Human Factors in Computing Systems*, pages 267–274. ACM Press, 2002.

[86] D. Harel. Statecharts : A visual formalism for complex systems. *Science of Computer Programming*, 8(3) :231–274, 1987.

[87] H.R. Hartson, A.C. Siochi, and D. Hix. The UAN : a user-oriented representation for direct manipulation interface designs. *ACM Trans. Inf. Syst.*, 8(3) :181–203, 1990.

[88] W.E. Hick. On the rate of gain of information. *Quarterly Journal of Experimental Psychology*, 4(1) :11–46, 1952.

[89] K. Hinckley, E. Cutrell, S. Bathiche, and T. Muss. Quantitative analysis of scrolling techniques. In *CHI '02 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 65–72, New York, NY, USA, 2002. ACM Press.

[90] K. Hinckley, M. Czerwinski, and M. Sinclair. Interaction and modeling techniques for desktop two-handed input. *Proceedings of the 11th annual ACM symposium on User interface software and technology*, pages 49–58, 1998.

[91] K. Hornbæk, B. B. Bederson, and C. Plaisant. Navigation patterns and usability of zoomable user interfaces with and without an overview. *ACM Trans. Comput.-Hum. Interact.*, 9(4) :362–389, 2002.

[92] K. Hornbæk and E. Frøkjær. Reading of electronic documents : the usability of linear, fisheye, and overview+detail interfaces. In *CHI '01 : Proc. Human Factors in Computing Systems*, pages 293–300. ACM Press, 2001.

[93] S. E. Hudson, J. Mankoff, and I. Smith. Extensible input handling in the subarctic toolkit. In *CHI '05 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 381–390, New York, NY, USA, 2005. ACM Press.

[94] R. Hyman. Stimulus information as a determinant of reaction time. *Journal of Experimental Psychology*, 45(3) :188–96, 1953.

[95] T. Igarashi and K. Hinckley. Speed-dependent automatic zooming for browsing large documents. In *UIST '00 : Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 139–148, New York, NY, USA, 2000. ACM Press.

[96] R. J. K. Jacob, L. Deligiannidis, and S. Morrison. A software model and specification language for non-wimp user interfaces. *ACM Trans. Comput.-Hum. Interact.*, 6(1) :1–46, 1999.

[97] R.J.K. Jacob, L.E. Sibert, D.C. McFarlane, and M.P. Mullen. Integrality and separability of input devices. *ACM Trans. Comput.-Hum. Interact.*, 1(1) :3–26, 1994.

[98] B.E. John and D.E. Kieras. The goms family of user interface analysis techniques : comparison and contrast. *ACM Trans. Comput.-Hum. Interact.*, 3(4) :320–351, 1996.

[99] B.E. John and D.E. Kieras. Using goms for user interface design and evaluation : which technique ? *ACM Trans. Comput.-Hum. Interact.*, 3(4) :287–319, 1996.

[100] S. Jul and G. W. Furnas. Critical zones in desert fog : Aids to multiscale navigation. In *UIST'98 : Proc. ACM Symposium on User Interface Software and Technology*, pages 97–106, 1998.

[101] P. Kabbash, W. Buxton, and A. Sellen. Two-handed input in a compound task. *Proceedings of the SIGCHI conference on Human factors in computing systems : celebrating interdependence*, pages 417–423, 1994.

[102] D.E. Kieras, D.E. Meyer, J.A. Ballas, and E.J. Lauber. Modern computational perspectives on executive mental processes and cognitive control : Where to from here. *Control of Cognitive Processes : Attention and Performance XVIII*, pages 681–712, 2000.

[103] A. Kobsa. User Experiments with Tree Visualization Systems. *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, pages 9–16, 2004.

[104] G. Kurtenbach and W. Buxton. The limits of expert performance using hierarchic marking menus. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 482–487, 1993.

[105] G.P. Kurtenbach, A.J. Sellen, and W.A.S. Buxton. An Empirical Evaluation Articulatory and Cognitive Marking Menus. *Human-Computer Interaction*, 8 :1–23, 1993.

[106] J. Lamping, R. Rao, and P. Pirolli. A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *CHI '95 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 401–408, New York, NY, USA, 1995. ACM Press/Addison-Wesley Publishing Co.

[107] T.K. Landauer and D.W. Nachbar. Selection from alphabetic and numeric menu trees using a touch screen : breadth, depth, and width. *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 73–78, 1985.

[108] J. A. Landay. Silk : sketching interfaces like krazy. In *CHI '96 : Conference companion on Human factors in computing systems*, pages 398–399, New York, NY, USA, 1996. ACM Press.

[109] E. Lecolinet. A Brick Construction Game Model for Creating Graphical User Interfaces : The Ubit Toolkit. *Proc of the IFIP Conference on Human-Computer Interaction (INTERACT)*, pages 510–518, 1999.

[110] J. Lehikoinen and I. Salminen. An empirical and theoretical evaluation of binscroll : A rapid selection technique for alphanumeric lists. *Personal Ubiquitous Comput.*, 6(2) :141–150, 2002.

[111] A.N. Leont'ev. *Activity, consciousness, and personality*. Prentice-Hall Englewood Cliffs, NJ, 1978.

[112] F.J. Lerch, M.M. Mantei, and J.R. Olson. Skilled financial planning : the cost of translating ideas into action. *ACM SIGCHI Bulletin*, 20 :121–126, 1989.

[113] S. Logan. *Gtk+ Programming in C*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001.

[114] M. Loy and R. Eckstein. *Java Swing*. O'Reilly, 2003.

[115] W. Mackay, C. Appert, M. Beaudouin-Lafon, O. Chapuis, Y. DU, J.D. Fekete, and Y. Guiard. Touchstone : Exploratory design of experiments. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*. In press, 2007.

[116] W.E. Mackay. Which Interaction Technique Works When ? Floating Palettes, Marking Menus and Toolglasses support different task strategies. *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 203–208, 2002.

[117] I. Scott MacKenzie and Shaidah Jusoh. An evaluation of two input devices for remote pointing. In *EHCI '01 : Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction*, pages 235–250, London, UK, 2001. Springer-Verlag.

[118] I.S. MacKenzie. Fitts'law as a research and design tool in human-computer interaction. *Human-Computer Interaction*, 7(1) :91–139, 1992.

[119] I.S. MacKenzie and W. Buxton. Extending fitts' law to two-dimensional tasks. In *CHI '92 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 219–226, New York, NY, USA, 1992. ACM Press.

[120] I.S. MacKenzie, T. Kauppinen, and M. Silfverberg. Accuracy measures for evaluating computer pointing devices. In *CHI '01 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 9–16, New York, NY, USA, 2001. ACM Press.

[121] T. Masui. Lensbar - visualization for browsing and filtering large lists of data. In *INFOVIS '98 : Proceedings of the 1998 IEEE Symposium on Information Visualization*, pages 113–120, Washington, DC, USA, 1998. IEEE Computer Society.

[122] T. Masui, K. Kashiwagi, and IV G. Borden. Elastic graphical interfaces to precise data manipulation. In *CHI '95 : Conference companion on Human factors in computing systems*, pages 143–144, New York, NY, USA, 1995. ACM Press.

[123] M. McGuffin and R. Balakrishnan. Acquisition of expanding targets. In *CHI '02 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 57–64, New York, NY, USA, 2002. ACM Press.

[124] T. Moscovich and J. Hughes. Navigating documents with the virtual scroll ring. In *UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 57–60, New York, NY, USA, 2004. ACM Press.

[125] B. A. Myers. A new model for handling input. *ACM Trans. Inf. Syst.*, 8(3) :289–320, 1990.

[126] B. A. Myers, D. A. Giuse, R. B. Dannenberg, D. S. Kosbie, E. Pervin, A. Mickish, B. Vander Zanden, and P. Marchal. Garnet : Comprehensive support for graphical, highly interactive user interfaces. *Computer*, 23(11) :71–85, 1990.

[127] B. A. Myers, R. G. McDaniel, R. C. Miller, A. S. Ferrency, A. Faulring, B. D. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The amulet environment : New models for effective user interface software development. *IEEE Trans. Softw. Eng.*, 23(6) :347–365, 1997.

[128] B.A. Myers. Separating application code from toolkits : eliminating the spaghetti of call-backs. In *UIST '91 : Proceedings of the 4th annual ACM symposium on User interface software and technology*, pages 211–220, New York, NY, USA, 1991. ACM Press.

[129] B.A. Nardi. *Context and Consciousness : Activity Theory and Human-Computer Interaction*. MIT Press, 1996.

[130] D. Navarre, P. Palanque, R. Bastide, and O. Sy. Structuring Interactive Systems Specifications for Executability and Prototypability. *7th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS*, 2000.

[131] D. Nekrasovski, A. Bodnar, J. McGrenere, F. Guimbretière, and T. Munzner. An evaluation of pan & zoom and rubber sheet navigation with and without an overview. In *CHI '06 : Proc. Human Factors in Computing Systems*, pages 11–20. ACM Press, 2006.

[132] W. M. Newman. A system for interactive graphical programming. *Seminal graphics : poineering efforts that shaped the field table of contents*, pages 409–416, 1968.

[133] C. North and B. Shneiderman. Snap-together visualization : a user interface for coordinating visualizations via relational schemata. In *AVI '00 : Proc. working conference on Advanced Visual Interfaces*, pages 128–135. ACM Press, 2000.

[134] J.R. Olson and E. Nilsen. Analysis of the Cognition Involved in Spreadsheet Software Interaction. *Human-Computer Interaction*, 3(4) :309–349, 1987.

[135] J.R. Olson and G.M. Olson. *The growth of cognitive modeling in human-computer interaction since GOMS*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[136] J.K. Ousterhout. *Tcl and the Tk toolkit*. Addison-Wesley Reading, Mass, 1994.

[137] P. Palanque and R. Bastide. Petri net based design of user-driven interfaces using the cooperative object formalism. In F. Paternó, editor, *Design, Specification and Verification of Interactive Systems '94*, pages 383–400, Heidelberg, 1994. Springer-Verlag.

[138] S.J. Payne and T.R.G. Green. Task-Action Grammars : A Model of the Mental Representation of Task Languages. *Human-Computer Interaction*, 2(2) :93–133, 1986.

[139] C.A. Petri. Kommunication mit Automaten. *Disseration, Technische Universitat Darmstadt*, 1962.

[140] E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pages 145–152, Dallas, USA, 2005.

[141] E. Pietriga, C. Appert, and M. Beaudouin-Lafon. Pointing and beyond : an operationalization and preliminary evaluation of multi-scale searching. In *CHI '07 : Proceedings of the SIGCHI conference on Human factors in computing systems*. In press, 2007.

[142] C. Plaisant, J. Grosjean, and BB Bederson. SpaceTree : supporting exploration in large node link tree, design evolution and empirical evaluation. *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 57–64, 2002.

[143] C. Plaisant, J. Grosjean, and BB Bederson. SpaceTree : supporting exploration in large node link tree, design evolution and empirical evaluation. *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, pages 57–64, 2002.

[144] Catherine Plaisant. The challenge of information visualization evaluation. In *AVI '04 : Proceedings of the working conference on Advanced visual interfaces*, pages 109–116, New York, NY, USA, 2004. ACM Press.

[145] P.G. Polson, C. Lewis, J. Rieman, and C. Wharton. Cognitive walkthroughs : a method for theory-based evaluation of user interfaces. *International Journal of Man-Machine Studies*, 36(5) :741–773, 1992.

[146] M. Posner and S. Petersen. The attention system of the human brain. *Annual Review of Neuroscience*, 13 :25–42, 1990.

[147] JInput Project. Jinput, the java input api project. `http://jinput.dev.java.net/`.

[148] R Project. The r project for statistical computing. `http://www.r-project.org`.

[149] R. Raisamo and K. J. Räihä. A new direct manipulation technique for aligning objects in drawing programs. In *UIST '96 : Proceedings of the 9th annual ACM symposium on User interface software and technology*, pages 157–164, New York, NY, USA, 1996. ACM Press.

[150] G. Ramos and R. Balakrishnan. Fluid interaction techniques for the control and annotation of digital video. In *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 105–114, New York, NY, USA, 2003. ACM Press.

[151] G. Ramos and R. Balakrishnan. Zliding : fluid zooming and sliding for high precision parameter manipulation. In *UIST '05 : Proceedings of the 18th annual ACM symposium on User interface software and technology*, pages 143–152, New York, NY, USA, 2005. ACM Press.

[152] U.D. Reips and C. Neuhaus. Wextor. `http://psychwextor.unizh.ch/wextor/`.

[153] P. Reisner. Formal Grammar and Human Factors Design of an Interactive Graphics System. *IEEE Transactions on Software Engineering*, 7(2) :229–240, 1981.

[154] J. Rekimoto. Pick-and-drop : a direct manipulation technique for multiple computer environments. In *UIST '97 : Proceedings of the 10th annual ACM symposium on User interface software and technology*, pages 31–39, New York, NY, USA, 1997. ACM Press.

[155] Y. Rogers. New theoretical approaches for HCI. *ARIST : Annual Review of Information Science and Technology*, 38 :87–143, 2004.

[156] D. Rubine. Specifying gestures by example. In *SIGGRAPH '91 : Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 329–337, New York, NY, USA, 1991. ACM Press.

[157] SAS. Jmp. `http://www.jmp.com`.

[158] S.C. Seow. Information Theoretic Models of HCI : A Comparison of the Hick-Hyman Law and Fitts' Law. *Human–Computer Interaction*, 20 :315–352, 2005.

[159] B. Shneiderman. Direct manipulation : A step beyond programming languages. *Proceedings of the joint conference on easier and more productive use of computer systems.(Part-II) on Human interface and the user interface table of contents*, 1981.

[160] A.C. Siochi and H.R. Hartson. Task-oriented representation of asynchronous user interfaces. In *CHI '89 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 183–188, New York, NY, USA, 1989. ACM Press.

[161] D.C. Smith, C. Irby, R. Kimball, and E. Harslem. The star user interface : an overview. *on AFIPS Conference Proceedings ; vol. 55 1986 National Computer Conference table of contents*, pages 383–396, 1986.

[162] G. Smith and M. Schraefel. The radial scroll tool : scrolling support for stylus- or touch-based document navigation. In *UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 53–56, New York, NY, USA, 2004. ACM Press.

[163] R. William Soukoreff and I. Scott MacKenzie. Generalized fitts' law model builder. In *CHI '95 : Conference companion on Human factors in computing systems*, pages 113–114, New York, NY, USA, 1995. ACM Press.

[164] R. William Soukoreff and I. Scott MacKenzie. Towards a standard for pointing device evaluation, perspectives on 27 years of fitts' law research in hci. *Int. J. Hum.-Comput. Stud.*, 61(6) :751–789, 2004.

[165] J. Stewart, B. B. Bederson, and A. Druin. Single display groupware : a model for co-present collaboration. In *CHI '99 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 286–293, New York, NY, USA, 1999. ACM Press.

[166] R. Stockli, E. Vermote, N. Saleous, R. Simmon, and D. Herring. The Blue Marble Next Generation - A true color earth dataset including seasonal dynamics from MODIS. Published by the NASA Earth Observatory, 2005.

[167] P. S. Strauss. Iris inventor, a 3d graphics toolkit. In *OOPSLA '93 : Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 192–200, New York, NY, USA, 1993. ACM Press.

[168] S. H. Tang and M. A. Linton. Pacers : time-elastic objects. In *UIST '93 : Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 35–43, New York, NY, USA, 1993. ACM Press.

[169] M. Terry and E. D. Mynatt. Side views : persistent, on-demand previews for open-ended tasks. In *UIST '02 : Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 71–80, New York, NY, USA, 2002. ACM Press.

[170] B. H. Thomas and P. Calder. Applying cartoon animation techniques to graphical user interfaces. *ACM Trans. Comput.-Hum. Interact.*, 8(3) :198–222, 2001.

[171] L.S. Vygotsky. *Mind in society*. Harvard University Press, 1980.

[172] C. Ware and M. Lewis. The DragMag image magnifier. In *CHI '95 conference companion, Human Factors in Computing Systems*, pages 407–408. ACM Press, 1995.

[173] S. Zhai, S. Conversy, M. Beaudouin-Lafon, and Y. Guiard. Human on-line response to target expansion. In *CHI '03 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 177–184, New York, NY, USA, 2003. ACM Press.

[174] S. Zhai and B. Smith. Multistream input : An experimental study of document scrolling methods. *IBM Systems Journal*, 38(4) :642–651, 1999.

Annexes

8.1 Code complet de la balle rebondissante avec SwingStates

```
AnimationTranslateBy animBall = new AnimationTranslateBy(-10, 10);
AnimationScaleTo animCollide = new AnimationScaleTo(0.5, 1.5);
CEllipse ball = canvas.newEllipse(50, 50, 20, 20);
animBall.setAnimatedElement(ball);
animCollide.setDurationLap(200).setNbLaps(2).setAnimatedElement(ball);

CStateMachine smBall = new CStateMachine(canvas) {

  public State idle = new State() {
    Transition collideX = new CElementEvent(ball, ">> collide") {
      public boolean guard() {
        boolean out = ball.getMinX() < 0;
        if(out) ball.setReferencePoint(0, 0.5)
                  .translateTo(0, ball.getCenterY());
        else {
          out = out || ball.getMaxX() > canvas.getWidth();
          if(out) ball.setReferencePoint(1, 0.5)
                    .translateTo(canvas.getWidth(), ball.getCenterY());
        }
        return  out;
      }
      public void action() {
        animBall.setDelta(-animBall.getDx(), animBall.getDy());
        animBall.suspend();
        animCollide.setScaleTarget(0.3, 1.3).start();
      }
    };
    Transition collideY = new CElementEvent(ball, ">> collide") {
      public boolean guard() {
        boolean out = ball.getMinY() < 0;
        if(out) ball.setReferencePoint(0.5, 0)
                  .translateTo(ball.getCenterX(), 0);
        else {
          out = out || ball.getMaxY() > canvas.getHeight();
          if(out) ball.setReferencePoint(0.5, 1)
                     .translateTo(ball.getCenterX(), canvas.getHeight());
        }
        return  out;
      }
      public void action() {
        animBall.setDelta(animBall.getDx(), -animBall.getDy());
        animBall.suspend();
        animCollide.setScaleTarget(1.3, 0.3).start();
      }
    };
  };
```

```
  public State collide = new State() {
    Transition endCollide = new AnimationStopped(animCollide, ">> idle") {
      public void action() {
        animBall.resume();
      }
    };
  };
};
```

8.2 Code complet de l'expérimentation de Hick-Hyman

8.2.1 Le bloc `ReactionBlock`

```
/**
 * @touchstone.block "ReactionBlock"
 */
public class ReactionBlock extends Block {

  Canvas canvas;

  public ReactionBlock() {
    super();
    Rectangle r = GraphicsEnvironment.getLocalGraphicsEnvironment()
      .getDefaultScreenDevice().getDefaultConfiguration()
      .getBounds();
    canvas = new Canvas(r.width, r.height);
    canvas.setBounds(r);
  }

  /**
   * Displays a set of distractors around the starting shape.
   * @param sizeTarget The size of the distractors.
   * @param angleAvailable The angle to use around the starting shape
   *                       to spread the distractors.
   * @param nbElements The number of distractors.
   */
  public void digitTargets(int sizeTarget,
                           double angleAvailable,
                           int nbElements) {
    double angleStep = angleAvailable/(nbElements);
    int alea = (int)(Math.random()*(nbElements-1));
    for(int i = 0; i < nbElements; i++) {
      if(alea == i)
        canvas.newEllipse(
            canvas.getWidth()/2 - sizeTarget/2,
            canvas.getHeight()/2 - sizeTarget/2,
            sizeTarget,
            sizeTarget)
```

```
          .setOutlined(false).setFillPaint(Color.RED)
          .translateBy(
            Math.cos(i*angleStep)*225,
            -Math.sin(i*angleStep)*225)
          .addTag("target");
      else
        canvas.newEllipse(
            canvas.getWidth()/2 - sizeTarget/2,
            canvas.getHeight()/2 - sizeTarget/2,
            sizeTarget,
            sizeTarget)
          .setOutlined(false).setFillPaint(Color.BLACK)
          .translateBy(
            Math.cos(i*angleStep)*225,
            -Math.sin(i*angleStep)*225).addTag("distractor");
    }
  }

  public void beginTrial() {
    digitTargets(40, 2*Math.PI,
      Integer.parseInt(getPlatform().getMeasureValue("N").toString()));
    Platform.getInstance().add(canvas);
    Platform.getInstance().repaint();
  }

  public void endTrial(EndCondition ec) {
    canvas.removeShapes(canvas.getTag("target"))
      .removeShapes(canvas.getTag("distractor"))
      .removeShapes(canvas.getTag("textTarget"));
    Platform.getInstance().remove(canvas);
  }

}
```

8.2.2 L'intertitre `StartButton`

```
/**
 * @touchstone.intertitle StartButton
 */
public class StartButtonIntertitle extends Intertitle {

  Canvas canvas;
  public static Font font = new Font("verdana", Font.PLAIN, 36);

  public StartButtonIntertitle() {
    super();
    Rectangle r = GraphicsEnvironment.getLocalGraphicsEnvironment()
      .getDefaultScreenDevice().getDefaultConfiguration()
```

```
      .getBounds();
    canvas = new Canvas(r.width, r.height);
    canvas.setBounds(r);
    CEllipse start = (CEllipse) canvas.newEllipse(
        canvas.getWidth()/2-20, canvas.getHeight()/2-20, 40, 40)
      .addTag("start").setOutlined(false).setFillPaint(Color.YELLOW);
    canvas.newText(0, 0, "START", font)
      .setPickable(false).setReferencePoint(0.5, 0.5)
      .translateTo(start.getCenterX(), start.getCenterY());
  }

  public void beginIntertitle() {
    Platform.getInstance().add(canvas);
    Platform.getInstance().repaint();
  }

  public void endIntertitle() {
    Platform.getInstance().remove(canvas);
  }

}
```

8.2.3 Le critère `PressOnTag`

```
/**
 * @touchstone.criterion PressOnTag
 */
public class PressOnTag extends PressCondition {

  private String tag = null;
  private CShape pickedShape = null;

  public PressOnTag(String button, String tag) {
    super(button);
    this.tag = tag;
  }

  public String getEndCondition() {
    return "Press on "+tag;
  }

  public boolean isFinished(Timer arg0, long arg1) {
    return false;
  }

  // Comme nous utilisons le système d'événements standard
  // pour cette expérimentation, nous n'avons pas
  // développé la vérification pour le système d'événements
```

```
  // de TouchStone
  public boolean isReached(AxesEvent arg0) {
    return false;
  }

  public boolean isReached(InputEvent arg0) {
    if(! super.isReached(arg0)) return false;
    Canvas canvas = (Canvas)Platform.getInstance().getComponent(0);
    pickedShape = canvas.pick(((MouseEvent)arg0).getPoint());
    if(pickedShape == null) return false;
    return pickedShape.hasTag(tag);
  }

  public void start() {
    super.start();
    pickedShape = null;
    // Informe la plateforme que le canvas est le composant graphique
    // à traquer pour détecter si le critère est vérifié
    Platform.getInstance().registerComponent(
      Platform.getInstance().getView().getComponent());
  }

  public void stop() {
    super.stop();
    Platform.getInstance().unregisterComponent(
      Platform.getInstance().getView().getComponent());
  }

}
```

Résumé

La recherche en Interaction Homme-Machine a produit de nombreuses techniques d'interaction pour améliorer l'utilisabilité des applications graphiques. Cependant, les produits industriels n'en tirent que très rarement profit, leurs interfaces restant des compositions de "widgets" classiques. Ce constat est dû à un manque d'outils appropriés pour la conception d'applications graphiques. D'une part, les concepteurs d'interfaces doivent pouvoir mesurer l'efficacité d'une technique d'interaction afin de faire des choix informés. D'autre part, les développeurs ont besoin d'environnements permettant de programmer les techniques choisies qu'elles soient classiques ou innovantes.

Cette thèse propose trois outils utilisables en synergie pour favoriser l'adoption de techniques d'interaction avancées, depuis l'imagination d'une technique jusqu'à son implémentation. Le premier outil, Complexity of Interaction Sequences (CIS), est un modèle pour décrire une technique d'interaction et prédire son efficacité dans un contexte d'utilisation donné. Le niveau d'abstraction élevé de CIS en fait un outil utilisable en amont de la conception et de l'évaluation afin de pouvoir envisager plusieurs techniques selon les besoins réels et d'apprécier leur efficacité à moindre coût. Le second outil, SwingStates, est une boîte à outils qui introduit un modèle de dessin et des structures de contrôle adaptées à la programmation de techniques d'interaction avancées tout en minimisant l'effort nécessaire à la maîtrise de cet outil par les développeurs. En effet, SwingStates est une extension de Java Swing, une boîte à outils largement utilisée pour le développement d'interfaces graphiques, et offre ainsi de nouvelles possibilités au développeur tout en restant dans leur cadre de travail habituel. Enfin, Touchstone, est une plateforme pour aider à la conception d'expérimentations contrôlées. Son architecture modulaire qui permet la réutilisation et son aspect exploratoire facilitent le processus complexe de réalisation d'expérimentations contrôlées. Touchstone est non seulement destinée aux évaluateurs, mais également aux concepteurs grâce à sa fonction d'entrepôt de résultats empiriques.