Académie de Montpellier

Université Montpellier II

— Sciences et Techniques du Languedoc —

THÈSE

présentée à l'Université des Sciences et Techniques du Languedoc
pour obtenir le diplôme de doctorat

Spécialité	: **INFORMATIQUE**
Formation Doctorale	: **Informatique**
École Doctorale	: **Information, Structures, Systèmes**

Vers des systèmes de fenêtrage distribués : l'évolution du drag-and-drop

par

Maxime Collomb

Soutenue le 1er Décembre 2006 devant le jury composé de :

M. Michel Beaudouin-Lafon, Professeur, Université Paris-Sud ..... Rapporteur
M. Philip Gray, Professeur, University of Glasgow ..... Rapporteur
M. Eric Lecolinet, Maître de conférences, ENST Paris ..... Examinateur
M. Peter King, Professeur, University of Manitoba ..... Examinateur
M. Guy Mélançon, Professeur, Université Montpellier III ..... Directeur de Thèse
Mme. Mountaz Hascoët, Maître de conf., Université Montpellier II ..... Directeur de Thèse

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION

Sommaire

Ce mémoire présente mon travail de thèse qui s'est focalisé sur les difficultés introduites par l'utilisation du mur-écran. Nous nous sommes intéressés aussi bien aux problématiques mono utilisateur que multi utilisateur. Lors de l'utilisation du mur-écran, un utilisateur se retrouve face à deux dispositifs d'affichage de natures différentes. Les différences se situent au niveau des dimensions des dispositifs d'affichage mais également au niveau des dispositifs de saisie associés. Tandis qu'un dispositif de pointage indirect est utilisé sur l'écran, un dispositif de pointage direct est utilisé sur le mur.

Les problématiques qui apparaissent avec l'utilisation du mur-écran sont de différentes natures :

- Les interactions entre les deux vues proposées par le mur-écran.
- Les interactions directes avec le mur.
- Le partage du mur entre plusieurs utilisateurs.

Avant d'aller plus loin dans la description de ces problématiques, nous allons présenter le mur-écran. Nous évoquerons ensuite les objectifs de ce mémoire.

1 Le mur-écran

Le mur-écran (figure 1) est un nouveau dispositif d'affichage conçu pour mettre en pratique des techniques de visualisation et d'interaction innovantes. Le mur-écran se compose de :

- Une surface de projection - le mur - sur laquelle sont rétro-projetées des informations diverses susceptibles d'être utilisées à n'importe quel moment, et
- Un ou plusieurs écrans classiques servant plus particulièrement pour la tâche spécifique sur laquelle un utilisateur travaille.

La surface de projection est de plus dotée d'un dispositif de pointage direct permettant aux utilisateurs d'interagir directement avec les éléments affichés sur la surface.

Les principaux points forts du mur-écran sont :

- Une augmentation et une structuration de la surface d'affichage : avec le mur-écran l'espace d'affichage augmente de manière significative en taille en même temps qu'il change de nature et offre ainsi de nouvelles possibilités d'organisation pour l'espace de travail.
- La rapidité d'accès à l'information : les informations affichées sur le mur sont accessibles directement soit par le biais de dispositif de pointage direct sur le mur soit par l'utilisation plus classique d'une souris.
- La coopération et le partage des informations : l'espace d'affichage mural offre des possibilités de coopération qui peuvent difficilement être mises en œuvre avec un affichage à l'écran. En effet, les informations affichées sur le mur sont susceptibles d'être utilisées, débattues, organisées par plusieurs utilisateurs soit de manière synchrone - lors de réunions s'appuyant sur des informations affichées à l'écran - soit de manière asynchrone - pour le partage d'informations importantes dans un groupe par exemple.

Aspect technique Comme le montre la figure 2, actuellement, une unité centrale contrôle les deux affichages du mur-écran. L'affichage mural est rétro projeté : un vidéo projecteur est positionné en dessous de la surface d'affichage. Son image se reflète dans un miroir puis apparaît sur la surface murale qui est translucide. Le fait de ne pas projeter l'image directement permet de s'assurer que l'utilisateur ne se situe pas entre le vidéo projecteur et la surface d'affichage. Ainsi, l'utilisateur ne fait pas d'ombre sur l'image projetée.

D'autre part, la surface murale possède une système de pointage direct : la barre de capture Mimio^{TM}. Celle ci fonctionne par infrarouge avec des stylos adaptés.

Le mur-écran se destine à différents contextes d'utilisation : mono ou multi utilisateurs.

Fig. 1: *Le mur-écran.*

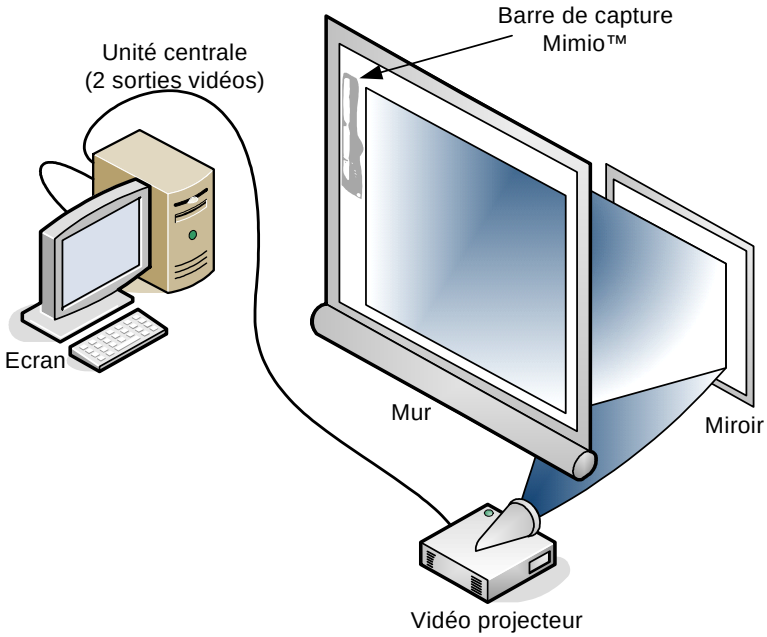

Fig. 2: *L'installation du mur-écran.*

1.1 Contexte mono utilisateur

Dans un contexte mono utilisateur, on peut distinguer 2 types d'utilisation du mur-écran :

- Le mur est utilisé comme un affichage secondaire et permet d'afficher des informations contextuelles. On dispose alors d'un système mur-écran que l'on peut classer dans la catégorie des systèmes focus+contexte [Baudisch et al., 2002].
- Le mur est utilisé comme un affichage principal et l'utilisateur interagit directement avec le mur. En tant que surface augmentée, le mur introduit de nouvelles problématiques au niveau de l'interaction de la visualisation.

1.2 Contexte multi utilisateurs

Le mur-écran a également vocation à être partagé entre les utilisateurs. Une utilisation collaborative du mur-écran, fait apparaître immédiatement les problèmes classiques liés aux collecticiels. Dans un contexte multi utilisateurs, le mur devient une zone de partage et de collaboration. Chaque utilisateur dispose de son propre affichage privé et le mur peut être utilisé par tous les utilisateurs.

Cependant, une telle utilisation du mur-écran ne va pas sans introduire un certain nombre de nouvelles problématiques. En effet, les besoins sont différents si les utilisateurs travaillent simultanément sur des surfaces contrôlées par une unique unité centrale ou si les utilisateurs disposent chacun de leur propre machine. En d'autres termes, est-il nécessaire de disposer de plusieurs systèmes de pointage indépendants sur une même machine ou faut-il conserver un dispositif de pointage par machine et mettre en relation les machines via le réseau ?

2 Les interactions

Le fait de travailler avec le mur-écran introduit également de nouvelles problématiques au niveau des interactions. Il faut tout d'abord permettre à l'utilisateur d'agir sur les deux surfaces mises à sa disposition. Il convient donc de résoudre les problèmes liés aux interactions entre le mur et l'espace de travail habituel affiché sur l'écran.

L'interaction directe avec le mur peut également être problématique. En effet, les interactions habituelles sont parfois délicates à utiliser sur une surface augmentée : le fait de disposer d'une surface physique de grande taille et d'un pointage direct remet en cause certaines des bases des interfaces graphiques actuelles qui sont pensées pour une utilisation sur un matériel classique (i.e. le triptyque écran/souris/clavier).

3 Objectifs

Ce travail de thèse a pour objet l'étude des interactions avec le mur-écran. Et plus généralement des interactions qui interviennent dans les environnements où plusieurs surfaces d'affichage sont mises à disposition du ou des utilisateurs, que ces environnements soient distribués ou non.

Nous nous intéressons plus particulièrement à une technique d'interaction emblématique du modèle de manipulation directe : le *drag-and-drop*. La raison de notre intérêt pour cette technique est que le *drag-and-drop* est largement répandu, que le *drag-and-drop* est délicat à utiliser sur les surfaces augmentées et que plusieurs travaux de recherche se sont appliqués à étendre les possibilités du *drag-and-drop*.

Le premier objectif est de déterminer quelle est la meilleure alternative au *drag-and-drop* dans un environnement aux surfaces d'affichage multiples. Et en particulier, quelle est la meilleure alternative au *drag-and-drop* lorsqu'un dispositif de pointage direct est utilisé.

Notre second objectif est lié aux environnements distribués. En effet, rien dans le *drag-and-drop* tel que nous le connaissons aujourd'hui ne le destine à une utilisation en dehors de l'espace clos qu'est le bureau de l'utilisateur. Un de nos objectifs est donc de faire évoluer le modèle d'interaction de cette technique afin qu'elle puisse être utilisée en environnement distribué.

4 Structure du mémoire

Comme nous l'avons dit précédemment, nous nous intéressons dans ce mémoire aux interactions lors de l'utilisation du mur-écran et plus généralement dans les environnements d'affichages distribués. Il est donc important de commencer par dresser un état de l'art des possibilités offertes par les environnements d'affichages distribués.

Dans un premier chapitre, nous dresserons une vue d'ensemble des systèmes permettant le partage des commandes des utilisateurs transmises au travers des dispositifs de saisie d'une part. Et permettant le partage des interfaces utilisateur d'autre part. L'enjeu majeur de ce chapitre sera de mettre en évidence les dimensions adéquates à la caractérisation des systèmes évoqués.

Nous nous intéresserons dans un second chapitre à la manipulation des données. Ce chapitre sera consacré aux techniques destinées à étendre les capacités du *drag-and-drop* dans les environnements distribués.

Dans la suite du mémoire, nous nous concentrerons sur le *drag-and-drop* et ses variantes : la première partie du chapitre III sera consacrée à la présentation de différentes implémentations du *drag-and-drop* tandis que nous présenterons, dans la seconde partie du chapitre, un nouveau modèle pour le *drag-and-drop* qui repose sur le modèle d'interaction instrumentale.

A l'issue de ce chapitre, toutes les caractéristiques du *drag-and-drop* et de ses évolutions auront été exposées et nous pourrons présenter nos études utilisateurs destinées à déterminer la ou les meilleures alternatives au *drag-and-drop* dans un contexte distribué.

Les deux derniers chapitres de ce mémoire seront consacrés à l'implémentation du modèle issu du chapitre III : une API implémentant le modèle instrumental du *drag-and-drop* fera l'objet du chapitre V tandis que la présentation d'une application de gestion collaborative de signets reposant sur cette API fera l'objet du chapitre VI.

Chapitre I

PARTAGE DES DISPOSITIFS D'ENTRÉE ET DE SORTIE

Sommaire

I.1 Introduction

Dans ce premier chapitre, nous allons dresser un état de l'art des systèmes permettant le partage des entrées – dispositifs de saisie – et des sorties – dispositifs d'affichage.

Nous abordons ici un domaine très vaste, le terme « partage » étant très flou. Des systèmes semblant à première vue relativement proches, mettent parfois en œuvre des solutions techniques très différentes. Par exemple, lorsque l'on parle de partage de pointeur, on peut faire référence à un système permettant d'utiliser un dispositif de pointage sur plusieurs machines mais on peut également faire référence à un système permettant d'utiliser plusieurs dispositifs de pointage simultanément sur la même machine.

Il est donc important, avant toute chose, de définir clairement les problématiques du partage des entrées et des sorties. Ceci fera l'objet de la première partie de ce chapitre. Nous pourrons ensuite caractériser un certain nombre de systèmes représentatifs mettant en œuvre le partage des entrées et/ou des sorties. Et au final, nous ferons un bilan présentant une comparaison des différents systèmes présentés.

I.2 Caractérisation des systèmes

La caractérisation des systèmes de partage des dispositifs d'entrée et de sortie a déjà fait l'objet de plusieurs travaux. Nous verrons les limites des classifications issues de ces travaux, après quoi nous proposerons une nouvelle classification plus complète.

I.2.1 Notation UDP/C

Dans l'objectif de caractériser les systèmes utilisant des pointeurs multiples, Lecolinet [2003b] a introduit la notation UDP/C. Bien que ces travaux s'intéressent essentiellement au partage des dispositifs d'entrée, les dispositifs de sortie sont également pris en compte car leur configuration peut influer sur l'utilisation des dispositifs d'entrée.

La notation UDP/C utilise les abréviations suivantes : **U** pour utilisateur, **C** pour unité centrale (*computer*), **D** pour dispositif d'affichage (*display*) et **P** pour pointeur. Ainsi, une configuration de bureau mono utilisateur classique sera notée 1-1-1/1 (un utilisateur ayant à sa disposition un écran et un pointeur sur cet écran et travaillant sur une unité centrale). Cette notation permet une caractérisation assez complète des différentes configurations que l'on peut rencontrer, qu'elles mettent en œuvre plusieurs dispositifs d'entrée, plusieurs dispositifs de sortie ou encore plusieurs utilisateurs.

Cependant, certains aspects fondamentaux n'apparaissent pas au travers de la notation UDP/C. Par exemple, si l'on place plusieurs configurations classiques telles que définies dans le paragraphe précédent (1-1-1/1) côte à côte, on obtient une configuration qu'il est possible de noter N-N-1/N (N utilisateurs utilisant N affichages, chaque affichage ne disposant que d'un seul pointeur. Le tout s'exécutant sur N unités centrales). On pourrait également dire qu'une telle configuration est la combinaison de N configurations de type 1-1-1/1. Ceci est un exemple d'une configuration pour laquelle il est difficile de choisir une notation UDP/C. Il est également fréquent de se confronter à ce genre de dilemme lorsqu'il faut opter pour une cardinalité N ou * pour une des composantes U, D, P ou C. En effet, dans un système N-N-1/N, si une des machines possède deux affichages, les N de D et de C deviennent différents. Faut-il alors utiliser la notation N-*-1/N ou N-N-1/*.

A l'inverse, il arrive également que deux systèmes soient placés dans la même catégorie alors qu'ils mettent en place des mécanismes différents voire opposés. Citons par exemple les systèmes Xdmx (section I.3.7) et EasyLiving (section I.3.6) qui, malgré leurs profondes différences techniques, sont tous deux à ranger dans la catégorie 1-N-1/*.

Au final, malgré le large spectre de systèmes pouvant être caractérisés grâce à la notation UDP/C, celle-ci présente une limite gênante : son ambiguïté. Cette ambiguïté est double : d'une part, un système donné peut parfois correspondre à plusieurs notations différentes. D'autre part, la même notation peut s'appliquer à des systèmes très différents, que ce soit par leurs implémentations, leurs objectifs ou leurs champs d'application.

I.2.2 Surfaces et instruments

Lachenal [2004], en étudiant les différents systèmes permettant d'utiliser des dispositifs de pointage et d'affichage multiples, a proposé une autre classification se basant sur quatre catégories :

- les systèmes **mono** surface **mono** instrument,
- les systèmes **mono** surface **multi** instrument,
- les systèmes **multi** surface **mono** instrument,
- les systèmes **multi** surface **multi** instrument.

Une surface représentant un dispositif d'affichage (sortie) et un instrument représentant un dispositif de saisie (entrée). Tout comme UDP/C, seuls les aspects « visibles » des systèmes sont pris en compte, les mécanismes utilisés étant ignorés.

Comparée à la notation UDP/C, cette classification des systèmes présente l'avantage d'être plus simple. Ainsi, le problème de plusieurs notations pouvant s'appliquer à un système ne se pose plus. En contrepartie, il est encore plus fréquent de classer plusieurs systèmes dans la même catégorie alors que ceux-ci ont des problématiques d'implémentation très différentes. Par exemple, un système multi instrument met-il en œuvre une ou plusieurs unités centrales ? Un réseau est-il nécessaire ?

Pour comprendre les différences et les similarités des systèmes que nous allons présenter dans notre état de l'art, il nous parait nécessaire de prendre en compte le mode de fonctionnement interne de ces systèmes.

I.2.3 Distributed display environments

En 2005, lors d'un *workshop* traitant des environnements d'affichages distribués [Hutchings et al., 2005], les caractéristiques principales des systèmes permettant la distribution de l'affichage ont été mises en évidence.

La première caractéristique est le degré de réplication qu'est capable de mettre en place un système. En effet, distribuer l'affichage peut nécessiter d'afficher la même information sur des affichages différents, en particulier lorsque les affichages ne sont pas à proximité directe les uns des autres.

La seconde caractéristique exhibée est l'utilisabilité, celle-ci pouvant être décrite en trois points :

- L'observabilité des relations entre les affichages. Il s'agit de la topologie des affichages : est-il aisé pour l'utilisateur de construire un espace de travail continu et plan à partir d'un ensemble de dispositifs d'affichage placés dans un espace en trois dimensions ?

- L'aspect dynamique ou statique de ces relations. Comment le déplacement, l'apparition ou la disparition d'un affichage (e.g. l'écran d'un ordinateur portable) est-il pris en compte ?
- L'évolutivité de l'environnement. Est-il possible d'ajouter des dispositifs d'affichage ou de changer leur nature.

Il faut noter que ce travail n'a porté que sur les dispositifs de sortie. De ce fait, les aspects multi utilisateur avaient peu d'importance. En effet, ceux-ci sont importants lorsque plusieurs utilisateurs doivent interagir (utilisation d'un dispositif d'entrée) simultanément avec un système. Mais qu'un ou plusieurs utilisateurs observent un dispositif d'affichage n'a que peu d'importance et a été ignoré lors de ces travaux.

Bien que les notions décrites ici manquent de détails et ne s'appliquent qu'aux dispositifs de sortie, elles ont servies de base aux notions que nous allons utiliser pour caractériser les systèmes de notre état de l'art.

I.2.4 Redirection et multiplexage

En étudiant un certain nombre de systèmes permettant le partage des dispositifs d'entrée et de sortie, nous avons vu apparaître deux mécanismes récurrents que nous nommons *redirection* et *multiplexage*. Ceux-ci peuvent intervenir tant au niveau des dispositifs d'entrée qu'au niveau des dispositifs de sortie.

Redirection au niveau des dispositifs d'entrée

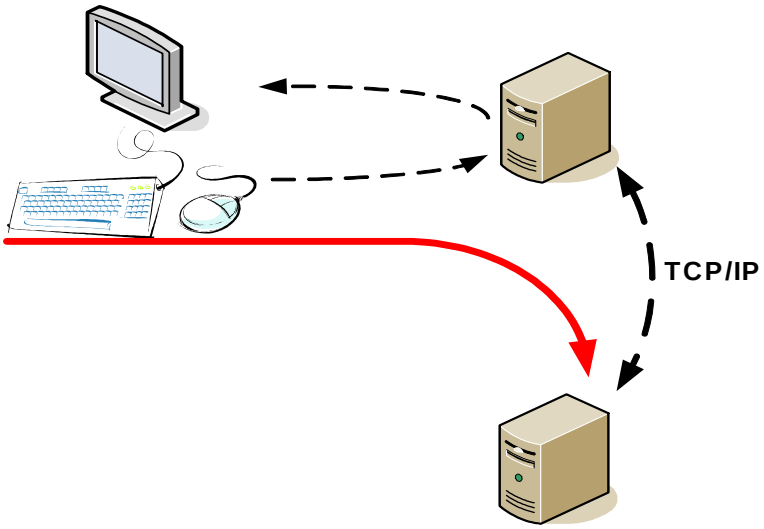

Fig. I.1: *Redirection des évènements issus des dispositifs de saisie.*

Définition I.1 *La redirection d'entrée est la transmission d'un flux d'entrée permettant à celui-ci d'être traité par une machine distante.*

La figure I.1 illustre la redirection d'entrée au travers d'une configuration relativement simple : une machine dispose de dispositifs de saisie et d'un dispositif d'affichage. Elle est également reliée à un réseau TCP/IP lui permettant de communiquer avec une machine distante.

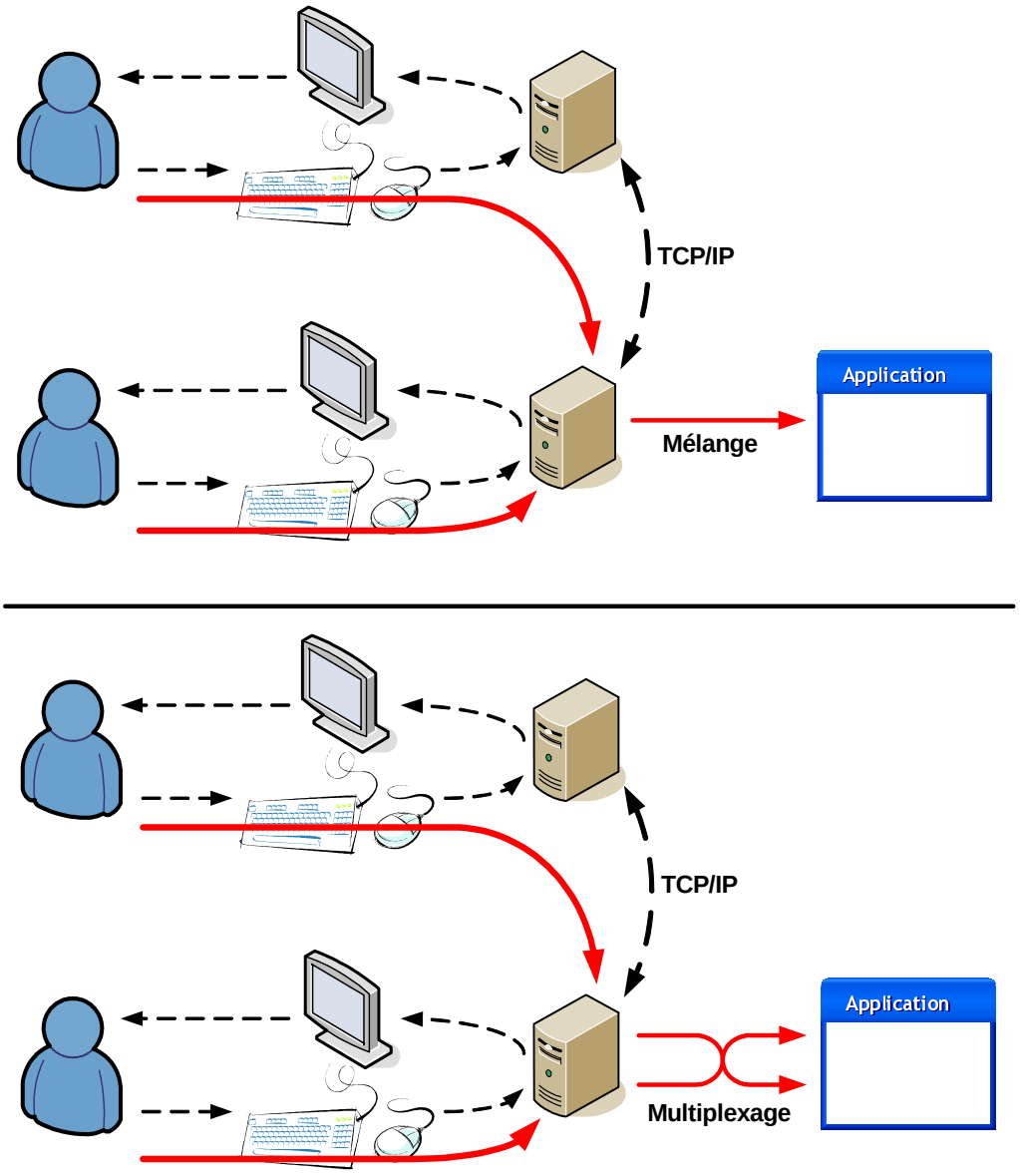

Fig. I.2: *en haut : mélange des flux, l'application ne voit qu'un seul flux d'évènements. en bas : multiplexage des flux, l'application peut identifier le dispositif source des évènements qu'elle reçoit.*

La redirection d'entrée peut se décrire en trois temps :

1. le flux d'évènements d'entrée est capturé sur la première machine,
2. le flux d'évènements transite par le réseau,
3. le flux d'évènements est reconstitué sur la machine distante.

Il faut noter que le format des évènements peut évoluer au cours de ces trois étapes. Il se peut en effet que les systèmes de fenêtrage utilisés sur les deux machines ne soient pas les mêmes. Dans ce cas, un format commun est utilisé pour les communications réseau : lors de l'étape 1, les évènements sont transformés du format utilisé par le système de fenêtrage de la première machine vers un format commun aux deux machines. Lors de l'étape 3, les évènements sont transformés du format commun vers le format utilisé par le système de fenêtrage de la seconde machine.

On peut citer comme exemple des systèmes comme VNC [Richardson et al., 1998] qui permettent de contrôler un bureau distant au travers d'une fenêtre de l'espace de travail local.

Multiplexage au niveau des dispositifs d'entrée

On parle de multiplexage des entrées lorsque plusieurs dispositifs de saisie de même type *agissent* simultanément sur le même espace de travail.

Les systèmes d'exploitation actuels ne permettent pas de gérer correctement plusieurs dispositifs de saisie de même type. En effet, lorsque plusieurs dispositifs de saisie sont connectés sur une machine disposant d'un système d'exploitation courant (*Microsoft Windows*$^{\text{TM}}$, *MacOS X*$^{\text{TM}}$ou *Linux*), les évènements sont mélangés sans prendre en compte le dispositif qui a émis un évènement.

Un exemple très courant est le cas d'un ordinateur portable disposant d'un pavé tactile et auquel on connecte une souris. Les deux systèmes de pointage commandent le même pointeur. Et, du point de vue d'un logiciel, il n'est pas possible de savoir de quel dispositif provient un évènement.

Définition I.2 *Le multiplexage des entrées est le processus par lequel plusieurs flux d'entrée sont transmis par le même canal. Les différents flux pouvant être reconstitués à l'issue de la transmission.*

La figure I.2 illustre deux modes de gestion des flux d'entrée. Dans les deux cas, une machine reçoit deux flux d'évènements provenant de dispositifs de saisie : l'un est connecté directement à la machine, le second est redirigé depuis une machine distante. La différence entre les deux modes de fonctionnement se trouve au niveau de la gestion de ces deux flux d'évènements. Tandis que les systèmes de fenêtrage courants mélangent les flux d'évènements (I.2–haut), le multiplexage des entrées consiste à identifier les différents flux (I.2–bas). Ainsi, l'application recevant un évènement est en mesure de savoir de quel dispositif il provient. Il est également possible pour le système de fenêtrage d'appliquer les évènements à différents pointeurs.

Pour qu'un système supporte le multiplexage des entrées, il est en général nécessaire d'assigner un identifiant à chaque dispositif de saisie et d'augmenter chaque évènement de l'identifiant du dispositif source.

Redirection au niveau des dispositifs de sortie

Définition I.3 *La redirection de sortie est la transmission de tout ou partie d'une vue permettant à celle-ci d'être affichée sur un dispositif distant.*

La notion de *vue* que nous utilisons dans cette définition est issue du modèle MVC [Krasner and Pope, 1988].

Un exemple de redirection de sorties est présenté sur la figure I.3. Une vue (e.g. interface utilisateur d'une application) générée par une machine A est transmise par le réseau à une machine B qui effectue le rendu de cette vue sur son dispositif d'affichage.

De la même manière que pour les entrées, la redirection des sorties peut se décrire en trois étapes :

1. une vue est générée par la première machine,
2. la vue transite par le réseau,
3. la vue est reconstituée et affichée par la seconde machine.

Là encore, le format des données échangées lors de la transmission réseau dépend fortement des systèmes. Nous verrons également dans la section suivante que ces formats peuvent avoir une granularité variable.

Il faut noter que la redirection de sortie implique parfois un mécanisme de réplication. En effet, lorsque la redirection de sortie est utilisée pour afficher une même vue pour différents utilisateurs qui ne se trouvent pas à proximité immédiate les uns des autres, il est nécessaire d'afficher plusieurs fois la même vue. On peut alors plus parler de réplication que de redirection. Cependant, des mécanismes similaires sont mis en œuvre.

Multiplexage au niveau des dispositifs de sortie

Le multiplexage des sorties n'existe pas en tant que tel mais nous utilisons ce terme pour regrouper deux notions : l'intégration et la séparation de vues.

Définition I.4 *L'intégration de sorties est la réunion de vues provenant de différentes sources au sein d'un même espace de travail.*

Définition I.5 *La séparation des sorties est la répartition de l'affichage d'un espace de travail entre différents dispositifs.*

Comme on peut le voir sur la figure I.4, l'intégration fait intervenir des vues provenant de différentes sources. L'intégration des vues peut se faire à des niveaux plus ou moins profonds : si nous revenons à l'exemple de VNC évoqué dans la section précédente, il est impossible de déplacer un fichier d'une machine à l'autre en déplaçant une icone représentant ce fichier entre la vue du bureau distant (application VNC) et le bureau local. L'intégration des vues peut permettre des interactions entre les différentes vues et par conséquent des interactions entre les différentes machines contrôlant ces vues. Donc, VNC ne fait d'intégration des vues.

La séparation des sorties n'est en aucun cas liée à l'intégration. Les processus d'intégration et de séparation peuvent prendre place indépendamment l'un de l'autre.

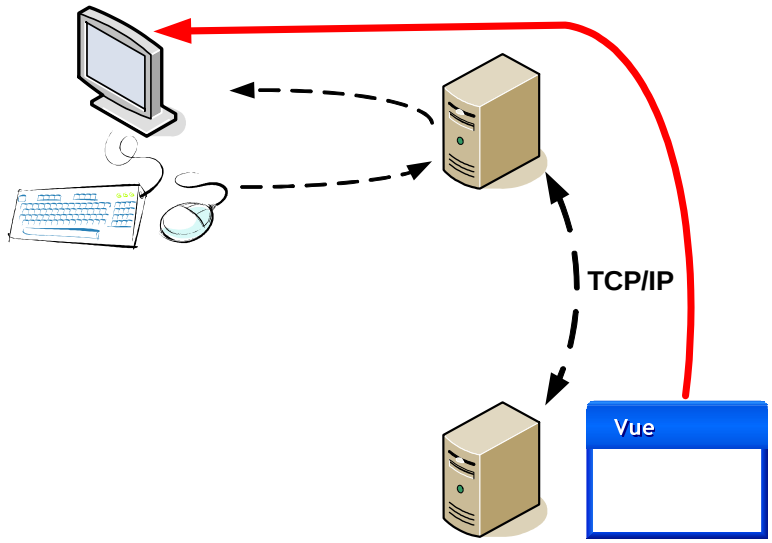

Fig. I.3: *Redirection des sorties.*

Redirection, intégration et séparation des sorties Il faut noter que l'intégration des sorties réunit des vues provenant de différentes sources. Il y a donc de très fortes probabilités qu'un processus de redirection soit nécessaire avant de pouvoir intégrer les vues.

D'autre part, il est également possible d'utiliser conjointement la séparation et la redirection des sorties. Dans l'exemple de la figure I.4, la redirection des sorties est utilisée avant l'intégration et également après la séparation des sorties. C'est ce qu'il se passe par exemple avec un système comme Xdmx (voir la section I.3.7) qui distribue l'afichage d'un espace de travail continu entre plusieurs serveurs X (et donc plusieurs machines).

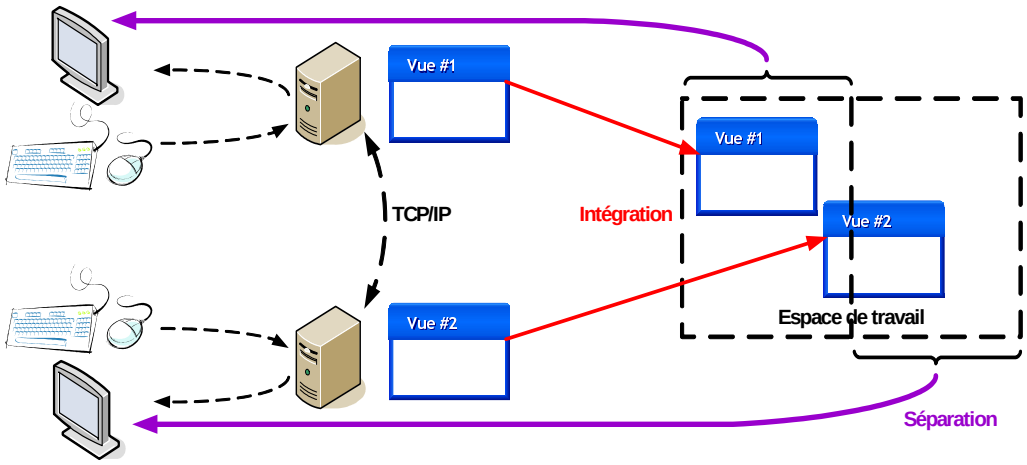

Fig. I.4: *Multiplexage des sorties.*

I.2.5 Granularité des communications

Dans les définitions qui viennent d'être données, nous avons vu apparaître la problématique de la granularité. En effet, nous avons expliqué, sans plus de précision que des informations étaient échangées entre deux machines lors de la redirection et du multiplexage des entrées ou des sorties.

Or, afin de pouvoir correctement comparer les systèmes de notre état de l'art, il convient de se pencher sur ce point. Nous pourrons ainsi comparer la granularité des informations échangées dans les différents systèmes étudiés.

Les différents niveaux de granularité des informations échangées correspondent à différents niveaux d'abstraction. Plus les informations sont abstraites, plus grande sera leur granularité.

Entrées

Tous d'abord, au niveau des entrées, nous avons déjà évoqué le fait qu'il pouvait être nécessaire de convertir les évènements d'entrée dans un format d'échange lorsque l'on se place dans un environnement hétérogène.

D'autre part, lorsque le multiplexage des entrées est utilisé, il est souvent nécessaire d'augmenter les évènements afin de pouvoir identifier le dispositif source. Il est ainsi possible de séparer les évènements provenant de dispositifs différents alors qu'ils transitent par le même canal.

Sorties

C'est au niveau des sorties que les différences de granularité sont les plus importantes. Si nous reprenons l'exemple de la figure I.3 et les trois étapes que nous avons déjà exposées : (1) génération d'un vue par la machine A, (2) transfert de cette vue vers la machine B et (3) reconstitution et affichage de la vue par la machine B.

La différence de granularité des communications entre les machines A et B provient ici de la répartition de la génération de la vue entre les deux machines. En effet, la création de la vue est faite conjointement par la machine A (étape 1) et la machine B (étape 3). Prenons l'exemple d'une application de modélisation 3D, qui s'exécute sur la machine A et dont l'affichage se fait sur la machine B. La répartition du rendu entre les machines A et B peut se faire de différentes manières, impliquant une utilisation différente des ressources des machines A et B.

Utiliser les ressources de la machine A Une première solution est le cas où la machine A effectue un rendu complet de l'interface de l'application. La machine A dispose alors d'un *frame buffer* (données *bitmap*) qu'elle peut transmettre à la machine B. L'application cliente de la machine B se contentera alors d'afficher le contenu du frame buffer. Notons que ce style de transfert peut être optimisé de plusieurs façons : en compressant les données bitmap, où en utilisant les transmissions ayant eu lieu précédemment (rafraîchissement de certaines zone de l'image bitmap seulement). Avec cette solution, un maximum des ressources nécessaires est utilisé sur la machine A.

Le point fort principal de cette solution est son aspect générique. Les interfaces de toutes les applications se résument toutes au final à un tableau de pixels. Au niveau du protocole de communication entre les machines serveur et cliente, les différences entre deux applications mettant en œuvre cette solution sont minimes voire nulles.

Utiliser les ressources de la machine B A l'inverse, une autre solution utilisant cette fois un maximum de ressources sur la machine B, consiste à transmettre depuis la machine A toutes les données décrivant la scène 3D dont il faut faire le rendu. C'est ainsi la machine B qui gère le rendu 3D proprement dit.

Un des avantages de cette solution est de permettre de meilleures performances. En effet, avec un tel fonctionnement, certaines interactions peuvent être traitées directement par le client de la machine B. Cela peut être le cas si, par exemple, l'utilisateur désire se déplacer dans la scène 3D. En effet, la scène n'étant pas modifiée, le nouveau rendu peut être fait directement sur la machine B sans communication avec la machine A, communications qui introduiraient automatiquement un temps de latence.

Cependant, cette solution ne dispose pas du tout de l'aspect générique de la solution précédente. L'application cliente s'exécutant sur la machine B dépend fortement de l'application utilisée sur la machine A et du type de données qu'elle transmet.

Une infinité de solutions intermédiaires Les deux cas évoqués précédemment sont des cas limites et il existe toute une palette de solutions intermédiaires. On pourrait par exemple imaginer que l'interface de l'application soit traitée en deux parties : la partie 2D de l'interface (menus, barres d'outils, etc.) serait rendue sur la machine A et la partie 3D (la scène) serait rendue sur la machine B.

I.2.6 Bilan

Dans cette section, nous avons défini les notions de redirection et de multiplexage des entrées et des sorties. Nous avons également exposé l'importance des différences de granularité des communications entrant en jeu pour l'implémentation de ces différentes notions.

Nous sommes maintenant parés pour comparer un ensemble de systèmes mettant en place le partage des entrées ou des sorties. Les systèmes que nous allons présenter maintenant sont groupés par types.

I.3 Systèmes mono utilisateur

I.3.1 PointRight et Synergy

PointRight [Johanson et al., 2002b] est le système de gestion des dispositifs de saisie du projet *Interactive Workspaces* de Stanford. Il permet à un dispositif de pointage d'agir séquentiellement sur plusieurs machines et donc plusieurs dispositifs d'affichage grâce à un mécanisme de redirection des entrées. Ce mécanisme est géré par le serveur d'évènements du projet *Interactive Workspaces*. La figure I.5 présente une configuration des affichages du projet *Interactive Workspaces* au sein duquel est intégré PointRight. PointRight permet d'utiliser ces différents affichages comme une zone de travail continue. Par exemple, dans un environnement tel que celui présenté sur la figure I.5, lorsque le pointeur atteint le bord gauche du SmartBoard 2, il passe sur le bord droit du SmartBoard 1. S'il atteint le bord inférieur du SmartBoard deux, il passe sur le bord supérieur de la table. Il est même possible de prendre en compte un environnement pseudo-3D en faisant passer le pointeur sur le bord inférieur du mur lorsqu'il atteint le bord gauche de la table.

Cependant, l'espace continu formé par ces affichages n'est qu'une illusion, seul le pointeur peut se déplacer d'un affichage à un autre. Il n'est pas possible de prendre un objet ou une fenêtre (*drag-and-drop*) et de le déplacer d'un affichage à un autre. Notons également que si plusieurs dispositifs de pointage se retrouvent à agir sur un même dispositif d'affichage, alors les évènements sont mélangés et n'agissent que sur un seul pointeur. Il ne peut y avoir au maximum qu'un seul pointeur actif par surface d'affichage. Cela signifie que l'espace d'affichage obtenu n'est pas issue d'un traitement sur les sorties mais uniquement de la mise en place d'une redirection des entrées.

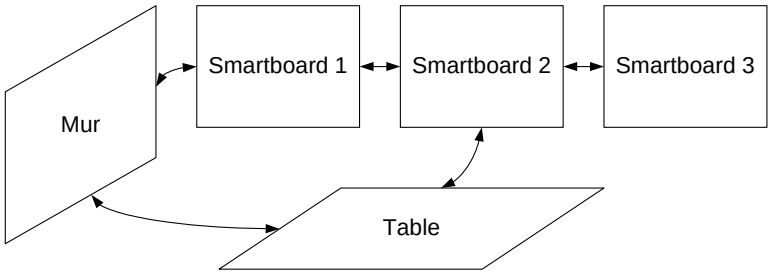

Fig. I.5: *Une configuration des écrans pour PointRight.*

Synergy [Schoeneman] fonctionne de la même manière que PointRight. Il se destine aux utilisateurs disposant de plusieurs machines fonctionnant éventuellement avec des systèmes d'exploitation différents. Chaque machine doit disposer de son propre affichage mais une seule doit disposer d'un clavier et d'une souris. La machine sur laquelle sont connectés le clavier et la souris exécute un serveur Synergy et les autres machines exécutent un client Synergy. La configuration de la topologie des dispositifs d'affichage est similaire à PointRight mais se limite à un agencement en deux dimensions. Synergy possède l'avantage d'être disponible pour les systèmes d'exploitation courants (Windows, Linux, MacOS).

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
PointRight Synergy	✓				

I.3.2 Extension de Whizz

Nous allons maintenant nous intéresser à un système mono utilisateur qui permet la mise en œuvre de plusieurs dispositifs de pointage dans un contexte d'interaction multi modale ou bi-manuelle.

Chatty a proposé une extension pour la boite à outils graphique Whizz [Chatty, 1994] afin de prendre en compte des interactions bi-manuelles. Ce travail propose entre autres la gestion des interactions parallèles. Nous pouvons par conséquent dire que Whizz met en place le multiplexage des entrées.

Un nouveau format d'évènements est proposé qui permet de gérer des évènements de plus haut niveau que les évènements basiques habituels. C'est la boite à outils qui se charge de générer ces évènements de plus haut niveau et ainsi d'éviter les interférences entre les différents dispositifs de pointage.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Ext. de Whizz		✓			

I.3.3 Extensions de l'affichage

Toujours dans les systèmes mono utilisateur, nous allons maintenant voir plusieurs possibilités d'extension de l'affichage d'une machine.

Dans un premier temps, citons les systèmes d'exploitation courants (*Microsoft Windows*™, *MacOS X*™ou *Linux*) qui supportent les sorties multiples. Les systèmes de fenêtrage de ces systèmes d'exploitation sont capable de gérer plusieurs dispositifs d'affichage, qu'il y ait plusieurs cartes graphiques dans une machine ou une seule carte disposant de plusieurs sorties. Les systèmes de fenêtrage gèrent donc la séparation des sorties.

Citons ensuite l'*USB Nivo* [Newnham research] et le *SideCar* [Digital tigers] qui sont des systèmes permettant d'ajouter des dispositifs d'affichage à une configuration en utilisant respectivement des connectiques USB et PCMCIA. Ces systèmes reposent sur l'utilisation de pilotes virtuels : le système d'exploitation fonctionne comme si la machine disposait de plusieurs cartes graphiques (ou d'une carte graphique possédant plusieurs sorties). Les affichages étant connectés sur la même machine, la redirection des sorties n'utilise pas le réseau mais les BUS USB et PCMCIA.

Cependant, dans le même ordre d'idée, *MaxiVista* [Bartels Media] permet d'utiliser un affichage distant de la même manière que s'il était connecté sur une machine locale (figure I.6). MaxiVista a un fonctionnement très proche de celui de VNC bien que ces systèmes n'aient pas le même objectif. MaxiVista repose donc sur une architecture client-serveur : le client fonctionne sur la machine secondaire tandis que le serveur fonctionne sur la machine primaire. Le serveur simule une carte graphique et, au lieu d'envoyer des données à un dispositif d'affichage, il transmet ces données au client en utilisant le protocole TCP/IP.

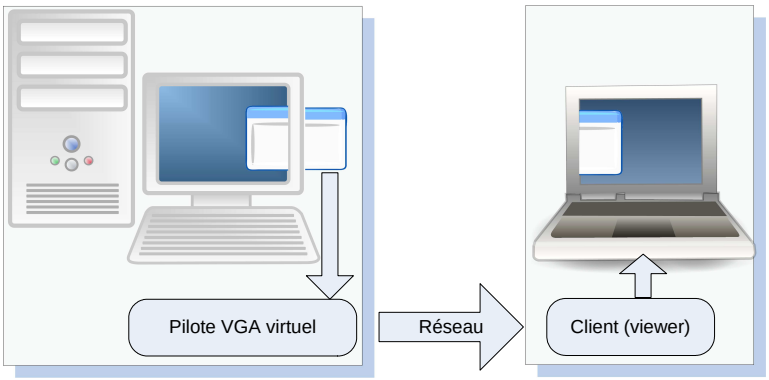

Fig. I.6: *Une fenêtre affichée en partie sur l'affichage de la machine exécutant le programme et en partie sur l'affichage d'une machine cliente [Bartels Media].*

Avec MaxiVista, le système d'exploitation croit gérer plusieurs écrans connectés à une même machine. De ce fait, l'utilisateur dispose des mêmes possibilités que sur un dispositif à plusieurs affichages : clonage des vues et extension du bureau. Le système reste cependant mono utilisateur : seuls les dispositifs de saisie de la machine serveur peuvent agir sur l'espace de travail.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Multi écran					✓
Maxivista			✓		✓

I.3.4 RemoteJFC

RemoteJFC [Lok et al., 2002] est une boite à outils pour les interfaces utilisateurs distribuées. C'est une extension de JFC (Java Foundation Classes) qui permet de développer des applications clientes légères sur un réseau. Cette boite à outils augmente toutes les classes des composants graphiques de JFC pour permettre l'appel de méthode à distance (en utilisant l'API RMI - *Remote Method Invocation* - comme le montre la figure I.7).

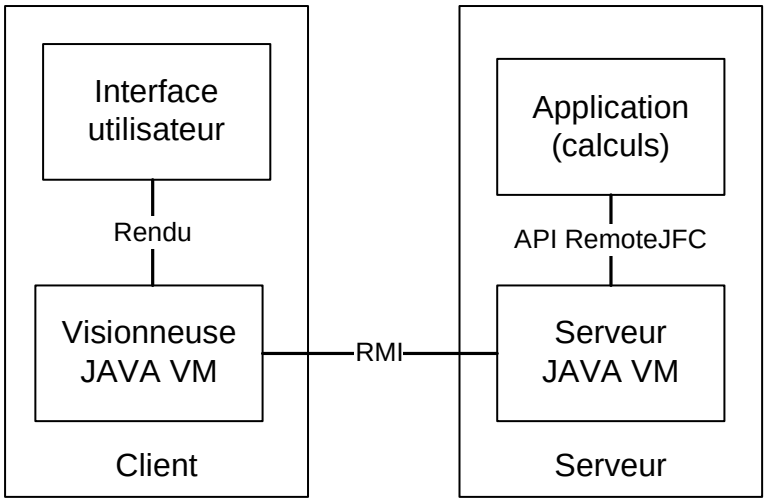

Fig. I.7: *Architecture d'une application écrite avec la boite à outils d'interface utilisateur distribuée RemoteJFC [Lok et al., 2002].*

Il est ainsi possible pour une machine serveur de construire une interface utilisateur sur une machine cliente et la contrôler depuis la machine serveur. Les communications entre les machines serveur et cliente se résument donc à des appels de méthodes (et les paramètres associés) sur les composants de l'interface utilisateurs.

Ainsi, les évènements en entrée de l'interface utilisateur sont redirigés vers la machine serveur pour être traités. De la même manière, les commandes d'affichages sont redirigées vers la machine cliente pour être exécutées.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
RemoteJFC	✓		✓	✓	

I.3.5 INDIGO

Le projet INDIGO (INteractive DIstributed Graphical Object) [Blanch et al., 2005] propose une architecture pour les applications distribuées. Cette architecture s'appuie sur un ensemble de standards (XML, XSLT, SVG, SOAP, etc.) et propose la réalisation d'interfaces distribuées

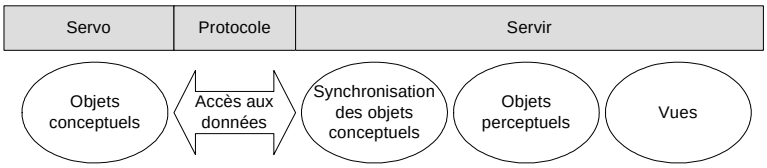

Fig. I.8: *Architecture INDIGO [Blanch et al., 2005].*

au travers de deux types de composants : des serveurs *d'objets* et des serveurs *d'interaction et de rendu* (figure I.8). Un serveur d'objets est en fait le noyau fonctionnel d'une application qui expose ses données à un ou plusieurs serveurs d'interaction et de rendu. Les serveurs partagent un modèle conceptuel que le serveur d'interaction et de rendu transforme en un modèle perceptuel. Le serveur d'interaction et de rendu contrôle donc l'interface utilisateur et peut interpréter les interactions de l'utilisateur sur le modèle perceptuel. Ces interactions donnent lieu à des commandes à appliquer au modèle conceptuel qui sont transmises au serveur d'objets.

Conceptuellement, INDIGO se rapproche du modèle MVC [Krasner and Pope, 1988], le serveur d'objets faisant office de *Modèle*, et les serveurs d'interaction et de rendu faisant office de *vues* et de *contrôleurs*.

La redirection des entrées se fait à un niveau d'abstraction important : les évènements sont traités par le serveur d'interaction et de rendu et sont transformés en commandes. La granularité des informations échangées est donc importante.

Du point de vue des sorties, INDIGO permet d'obtenir plusieurs vues synchronisées sur les mêmes objets. Cependant, on ne peut pas parler de redirection des sorties car les vues sont générées par les serveurs d'interaction et de rendu et ne transitent donc pas par le réseau.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
INDIGO	✓				

I.3.6 EasyLiving

Fig. I.9: *La pièce EasyLiving dans les laboratoires de Microsoft. La pièce est entre autre équipée de plusieurs machines [Brumitt et al., 2000].*

EasyLiving [Brumitt et al., 2000] est un projet d'informatique omniprésente (*ubiquitous computing*) destiné à une utilisation à domicile. Ce projet intègre des aspects domotiques (détection des utilisateurs, gestion de l'éclairage de la pièce, etc.).

De notre point de vue, EasyLiving est intéressant car il permet d'utiliser plusieurs dispositifs d'affichage et de saisie pour une même session de travail. La figure I.9 montre une pièce équipée de plusieurs machines. Il est possible, par exemple, de commencer une session de travail sur la machine se situant au fond de la pièce puis de déplacer la session et de continuer le travail sur l'affichage mural en utilisant la souris et le clavier situés sur la table basse.

D'un point de vue technique, chaque dispositif de saisie est lié à un proxy logiciel qui redirige les évènements vers la machine contrôlant l'affichage utilisé à un moment donné. Un même dispositif de saisie peut donc être utilisé avec différents dispositifs d'affichages. Cependant, il n'est possible d'utiliser qu'un seul dispositif de saisie avec un dispositif d'affichage à un moment donné. C'est pour cette raison que nous avons classé EasyLiving dans la catégorie des systèmes mono utilisateur.

En réalité, les capacités de distribution de EasyLiving sont réduites. Le système repose sur protocole *Remote Desktop Protocol* (RDP) et le composant *Terminal services* de Microsoft Windows. Toutes les applications s'exécutent sur un serveur unique, la machine cliente n'ayant qu'un rôle de terminal. Ce fonctionnement permet de déplacer une session de travail d'un dispositif d'affichage à un autre. En effet, une session de travail peut être interrompue sur une machine, continuer à être exécutée sur le serveur et être rouverte sur une autre machine.

EasyLiving permet donc la redirection des entrées et la redirection de l'affichage au niveau pixel mais ne permet pas le multiplexage ni des entrées ni des sorties. En effet, à un moment donné, une machine ne peut afficher qu'une seule interface : celle de la session de travail ouverte sur le serveur.

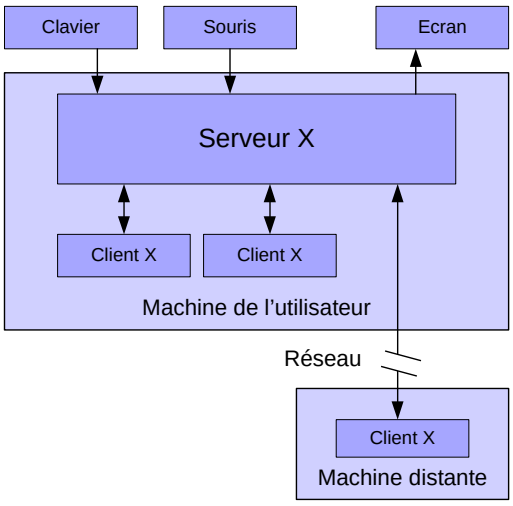

Fig. I.10: *Dans cet exemple, le serveur X reçoit des entrées d'un clavier et d'une souris et dispose d'un écran comme sortie. Deux clients X s'exécutent sur la même machine que le serveur X et un troisième client X s'exécute sur une machine distante [Wikimedia].*

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
EasyLiving	✓		✓		

I.3.7 X-Window

X-Window est un système de fenêtrage qui fournit les outils et un protocole standard pour construire des interfaces graphiques. Bien qu'il ait été originellement développé pour les systèmes Unix, pratiquement tous les systèmes d'exploitation actuels (*Microsoft Windows*™, *MacOS X*™, *Linux*) supportent X-Window. Il est basé sur un modèle client-serveur : Un serveur X s'exécute sur une machine qui dispose de dispositifs de saisie et d'au moins un dispositif d'affichage et communique avec un ensemble d'applications clientes. Le serveur accepte les requêtes d'affichage des clients et leur envoie les commandes de l'utilisateur (évènements souris et clavier).

Le modèle client-serveur de X-Window peut paraître déconcertant du fait que le serveur s'exécute sur la machine de l'utilisateur et que le client peut s'exécuter sur une machine distante. C'est la situation contraire d'une interaction client-serveur classique.

Le point fort de X-Window est de disposer d'un protocole de communication entre le client et le serveur qui soit transparent sur un réseau – exemple sur la figure I.10. Il n'est cependant pas aisé de changer de serveur pour un client donné. C'est-à-dire qu'il n'est, *a priori*, pas possible pour une application cliente de se déconnecter d'un serveur X, puis de se reconnecter à un autre serveur X. La connexion entre le client et le serveur X se fait lors du lancement de l'application cliente et ne cesse qu'à l'arrêt de cette application.

Les communications entre le serveur et les clients X ne se résument pas à des données bitmap. Un ensemble de primitives graphiques simples sont utilisées pour dessiner une interface utilisateur. Par exemple, l'affichage d'une ligne se traduit par l'appel de la primitive *XDrawLine()* par le client et l'exécution de cet primitive se fera sur le serveur X – c'est à dire, la machine de l'utilisateur. X-Window met donc en place la redirection des entrées et des sorties.

Les serveurs X classiques permettent, de la même manière que *Microsoft Windows* ™ou *MacOS X*™, d'utiliser plusieurs affichages connectés sur la machine exécutant le serveur X. X-Window permet donc la séparation des sorties.

Il existe une extension intéressante nommée *Xdmx* (*Distributed Multihead X*) [Xdmx project] qui prend la forme d'un serveur X intermédiaire – *proxy*. Ce serveur X intermédiaire contrôle plusieurs serveurs X classiques s'exécutant chacun sur une machine différente. L'ensemble des affichages ainsi contrôlés peut former un espace de travail continu et il est ainsi possible de déplacer l'affichage d'un client d'un serveur X à un autre. Sachant que l'application cliente n'a pas conscience de changer de serveur X puisqu'elle reste connectée au serveur Xdmx.

Notons enfin que dans le monde *Microsoft Windows*, *Citrix Presentation Server* [Citrix] offre des possibilités comparables à X-Window.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
X-Window	✓		✓	✓	✓

I.4 Systèmes multi utilisateur

I.4.1 VNC

Virtual Network Computing ou VNC [Li et al., 2000; Richardson et al., 1998] est un protocole pour se connecter à un ordinateur distant. Il existe plusieurs systèmes utilisant ce protocole (RealVNC, TightVNC, etc.) qui se décomposent en deux parties : le client et le serveur. Le serveur est le programme sur la machine qui partage son affichage et le client est le programme qui « regarde » et interagit avec le serveur. VNC est un système de fenêtrage virtuel dont l'objectif est de prendre le contrôle d'une machine distante.

VNC est un protocole simple basé sur le réaffichage d'un rectangle de pixels à une position (x, y) donnée. Le serveur envoie donc des petits rectangles au client via une communication TCP/IP. Pour éviter une trop grande utilisation de la bande passante, il existe plusieurs formes d'encodage. L'encodage le plus simple est le *raw encoding* où les pixels sont transmis de gauche à droite par ligne, et après le premier écran transféré en totalité, seuls les rectangles qui changent sont envoyés. Cette méthode fonctionne très bien si seulement une petite partie de l'interface change d'une image à l'autre – exemple : un pointeur de souris qui se déplace. Cependant, le système atteint ses limites lorsque tout l'affichage est en mouvement – une vidéo en plein écran par exemple.

VNC permet donc la redirection des entrées et des sorties mais est limité dans ses capacités de multiplexage. En effet, les capacités d'interaction entre l'espace de travail répliqué et l'espace de travail local sont quasiment inexistantes. Cependant, la plupart des implémentations de VNC autorisent le partage du presse-papier entre les machines serveur et clientes.

Il faut noter qu'au niveau des sorties, VNC ne fait pas seulement de la redirection mais plutôt de la réplication puisque l'interface utilisateur est affichée sur plusieurs machines : la machine serveur et la ou les machines clientes.

On peut également citer ici collaborative VNC [Levitt] qui est une variante de VNC permettant à plusieurs utilisateurs d'agir sur un même espace de travail. Tous les utilisateurs disposent de la même vue. Cette vue peut être contrôlée à tour de rôle par chacun des utilisateurs. Ce contrôle « à tour de rôle » ne permet pas de classer collaborative VNC dans la catégorie des systèmes faisant du multiplexage des entrées.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
VNC Col. VNC	✓		✓		

I.4.2 Wincuts

Wincuts [Tan et al., 2004] est une technique d'interaction qui autorise les utilisateurs à reproduire une partie quelconque d'une fenêtre existante. Un *WinCut* est une reproduction dynamique de la zone initiale : la fenêtre reproduite est interactive.

Wincuts peut être utilisé sur un seul affichage mais, comme on peut le voir sur la figure I.11, il est également possible de répliquer des parties de fenêtre sur un affichage distant.

Au niveau du mixage, les *Wincuts* sont considérés comme des fenêtres classiques mais il n'est pas possible de faire interagir des *WinCuts* provenant de machines différentes.

Fig. I.11: *Trois utilisateurs partagent des informations provenant de leurs ordinateurs portables respectifs en envoyant des WinCuts sur l'écran projeté [Tan et al., 2004].*

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Wincuts	✓		✓		

I.4.3 Le projet Augmented surface

Le projet *Augmented surface* [Rekimoto and Saitoh, 1999] est un environnement augmenté qui permet aux utilisateurs de partager des données entre leurs ordinateurs portables, les affichages augmentés (table et mur) et des objets physiques. Le système repose sur un système de reconnaissance à base de caméras et peut accueillir les ordinateurs portables des utilisateurs qui, en complément des affichages intégrés dans l'environnement, forment un espace de travail continu. En effet, grâce aux techniques d'interaction que sont l'*hyperdragging* et le *pick-and-drop* (voir chapitre II), les utilisateurs peuvent transférer des données d'une machine à une autre en ne connaissant que les positions relatives de ces machines.

L'*hyperdragging* est une extension du drag-and-drop qui autorise le déplacement de données au delà des limites de la zone d'affichage d'une machine. Pour commencer, un utilisateur déplace un objet en faisant un drag-and-drop classique. Quand le pointeur atteint un bord de la zone d'affichage, il « saute » sur la surface partagée la plus proche (l'emplacement relatif des différents affichages étant déterminé par le système de caméras vidéo). L'utilisateur peut alors déposer l'objet sur la surface partagée.

On a donc clairement affaire à un système utilisant la redirection des entrées. Et il utilise également le multiplexage des entrées. En effet, plusieurs utilisateurs ont la possibilité de faire des *hyperdragging* simultanément.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Augmented surfaces	✓	✓			

I.4.4 Logiciels collaboratifs synchrones

Cette catégorie regroupe les logiciels permettant une collaboration entre des utilisateurs n'étant pas à proximité immédiate les uns des autres. Chaque utilisateur dispose donc de sa propre machine, toutes les unités centrales étant reliées via un réseau. Chaque utilisateur dispose d'une vue sur un espace de travail commun et peut voir les actions des autres utilisateurs.

Lecolinet [2003b] différencie ici les pointeurs actifs et passifs. En fait, sur un dispositif d'affichage donné, on est susceptible de voir N pointeurs. Un seul de ces pointeurs (celui qui est lié au dispositif d'affichage) sera actif alors que les N-1 autres seront passifs. Sur un dispositif d'affichage donné, les pointeurs passifs permettent à l'utilisateur du pointeur actif de savoir ce que font les autres et ainsi d'anticiper les conflits pour mieux les éviter.

Ces systèmes gèrent plusieurs pointeurs provenant de différentes machines sur le même espace de travail et font donc de la redirection et du multiplexage d'entrées. Au niveau des sorties, il est nécessaire de répliquer certains éléments de l'interface utilisateur ce qui nécessite la mise en place d'un mécanisme de redirection des sorties.

DistView

DistView [Prakash and Shim, 1994] est une boite à outils destinée aux collecticiels synchrones permettant à plusieurs utilisateurs de partager la vue et les interactions sur une fenêtre (au sens de fenêtre dans un système de fenêtrage). La réplication de la vue d'une fenêtre partagée est réalisée au niveau des objets de l'interface (*textfields*, *sliders*, etc.).

Cette réplication nécessite l'utilisation d'un serveur de fenêtre partagé et passe par la création, pour chaque objet, d'un proxy. Ce proxy permet de synchroniser tous les messages que reçoit un objet entre la version originale et les versions répliquées de cet objet.

COAST

De même que DistView, COAST [Schuckmann et al., 1996] est une boite à outils pour le développement de collecticiels synchrones. Cependant, là où DistView considère le partage d'une fenêtre, l'architecture de COAST s'appuie sur la notion de document.

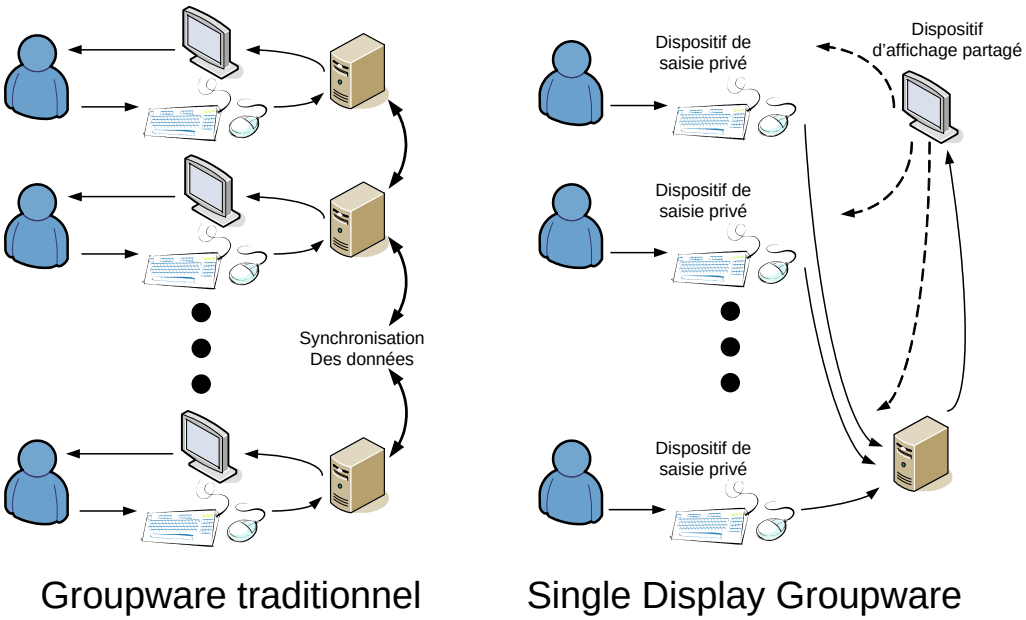

Fig. I.12: *Un Single Display Groupware permet à plusieurs utilisateurs d'agir sur un système qui dispose d'un unique affichage [Stewart et al., 1999].*

Les utilisateurs partagent un document – en réalité, les parties d'un document – et disposent chacun d'une vue sur ce document. Les utilisateurs peuvent donc se focaliser sur des parties différentes du document.

La réplication des objets se traduit par la réplication complète des données d'un document entre les différentes instances de l'application (généralement une par machine et par utilisateur). Ces objets peuvent être de n'importe quel type et leur réplication est gérée par un *gestionnaire de réplication.*

VIRTUS

Toujours dans le domaine des collecticiels synchrones, VIRTUS [Saar, 1999] propose une plateforme pour la manipulation collaborative de scènes VRML. VRML est un langage d'échange spécialisé dans la représentation d'univers virtuels en trois dimensions. Afin de minimiser les communications nécessaires à la synchronisation des objets de la scène 3D, la scène est décomposée en objets qui peuvent eux aussi être décomposés en une hiérarchie de sous-objets. Les objets traités par VIRTUS sont exactement de la même granularité que les objets 3D d'une scène VRML.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
DistView COAST VIRTUS	✓	✓	✓		

I.4.5 Single display groupware

Les *Single Display Groupwares* (SDG) sont des systèmes qui permettent à plusieurs utilisateurs de travailler simultanément sur un même affichage. En général, comme le montre la figure I.12, de tels systèmes mettent en œuvre plusieurs utilisateurs, chacun disposant d'un dispositif de saisie, et un seul dispositif d'affichage commun.

Comme l'indique son nom, cette catégorie regroupe des systèmes qui ne mettent en œuvre qu'un seul dispositif d'affichage. Il n'y a donc ni redirection ni multiplexage des sorties.

MMM (multi-device multi-user multi-editor)

MMM [Bier et al., 1992] propose deux éditeurs aux fonctionnalités très simples. L'intérêt est de permettre à plusieurs utilisateurs de travailler sur ces éditeurs simultanément. Les évènements gérés dans le cadre de MMM sont augmentés et intègrent notamment une identification du dispositif de pointage qui génère chaque évènement.

Les éditeurs développés prennent en compte cet identifiant et ne mélangent pas les suites d'évènements des différents dispositifs de pointage (3 au maximum).

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
MMM		✓			

Pebbles

Le projet Pebbles [Myers et al., 1998] consiste à utiliser de manière collaborative une application s'affichant sur un grand écran de type *whiteboard*. L'interaction se fait grâce à des PDAs (Personal Digital Assistants). Chaque utilisateur du système dispose de son PDA et tous les PDAs sont reliés à une même machine qui contrôle l'affichage du whiteboard.

Ce dispositif est destiné à être utilisé en salle de réunion pour des sessions de conception ou de brainstorming. Il fonctionne avec l'ensemble des applications existantes grâce à l'application *remote commander* qui transforme les actions de l'utilisateur sur le PDA en évènements du système. On a ici affaire à de la redirection d'évènements, redirection qui nécessite une certaine transformation pour générer des évènements de type souris/clavier à partir d'évènements provenant d'une surface tactile et d'un clavier virtuel (affiché sur l'écran du PDA). La redirection dont on parle ici n'est cependant pas du même type que celle des autres systèmes décrits dans ce chapitre puisque aucune communication réseau n'entre en compte et que les PDA sont reliés à l'unité centrale par des ports séries.

D'autre part, une application multi utilisateurs est présentée : *PebblesDraw* est un SDG qui permet à plusieurs utilisateurs possédant chacun leur PDA de dessiner sur une zone partagée du whiteboard. A chaque PDA correspond un pointeur de couleur différente. *PebblesDraw* met donc en place le multiplexage des entrées puisqu'il est capable de gérer des évènements provenant de différentes sources et de les identifier afin de ne pas les mélanger.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Pebbles	(✓)	✓			

MID

MID [Hourcade and Bederson, 1999; Hourcade et al., 2004] est un paquetage Java permettant de faciliter la conception de SDG. MID gère un ensemble de dispositifs de pointage et génère des évènements proches des évènements habituels de Java mais qui disposent en plus de la possibilité d'identifier le dispositif source d'un évènement.

Il faut noter que MID permet, au choix, de faire du multiplexage ou de mélanger les évènements pour permettre le support d'applications existantes qui ne sont pas capables de gérer plusieurs pointeurs.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
MID		✓			

MightyMouse

MightyMouse [Booth et al., 2002] est destiné à une utilisation particulière : l'objectif est de partager l'interaction sur un grand affichage entre les utilisateurs. On se place ici dans le contexte de la salle de réunion et on suppose que chaque participant dispose de son ordinateur portable personnel et qu'un de ces portables est visible de tous (via un projecteur).

MightyMouse se base sur VNC et permet de contrôler le pointage de l'écran partagé grâce aux dispositifs de pointage des différents ordinateurs portables. Cependant, l'affichage partagé ne dispose que d'un seul pointeur et les utilisateurs ne peuvent contrôler l'affichage partagé qu'à tour de rôle.

La différence entre MightyMouse et Collaborative VNC dont on a parlé précédemment (voir p. 24) se situe au niveau de l'affichage : alors que MightyMouse est pensé pour être utilisé sur un écran partagé, collaborative VNC se destine à des utilisateurs ne se trouvant pas dans une proximité immédiate et disposant chacun d'un affichage personnel.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
MightyMouse	✓				

Dynamo

Dynamo [Izadi et al., 2003] est un système pour le support des réunions informelles rapides. Il a d'ailleurs été testé dans une cafétéria. Le système permet principalement à un petit groupe d'utilisateurs d'échanger des fichiers ou de visualiser des documents.

Fig. I.13: *L'espace de partage de Dynamo : trois utilisateurs sont connectés sur le système (ils apparaissent dans la partie inférieure). Différentes applications sont lancées et deux zones de l'espace sont des sous espaces privés. Les icones dans le coin supérieur gauche de chaque zone privée indiquent que le premier utilisateur peut accéder à la zone de gauche, le second à la zone de droite et le troisième aux deux zones [Izadi et al., 2003].*

Dynamo permet un échange rapide entre les utilisateurs et s'appuie pour cela sur le fait que chaque utilisateur dispose de sa clé USB personnelle. Ainsi, un usager se connecte en branchant sa clé USB et est identifié grâce à celle-ci.

Dynamo reprend les caractéristiques de la plupart des SDG : plusieurs dispositifs de pointages et un seul dispositif d'affichage. Ce système met donc en œuvre le multiplexage des entrées.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Dynamo		✓			

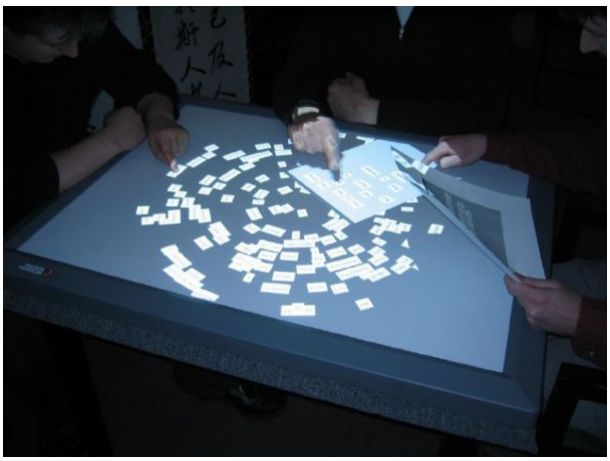

Fig. I.14: *Un groupe de trois personnes collaborent pour assembler un poème [Shen et al., 2004].*

DiamondTouch et DiamondSpin

Le système composé de DiamondTouch (partie matérielle) et DiamondSpin (partie logicielle) [Shen et al., 2004] est original pour plusieurs raisons : d'une part, du point de vue matériel, DiamondTouch est une surface tactile qui autorise plusieurs interactions simultanées. De plus, la surface tactile de DiamondTouch permet d'identifier chacun des utilisateurs grâce à leurs positions autour de surface.

D'autre part, DiamondSpin, la partie logicielle du système est elle aussi originale. En effet, DiamondTouch et DiamondSpin sont pensés pour être utilisés sur une table.

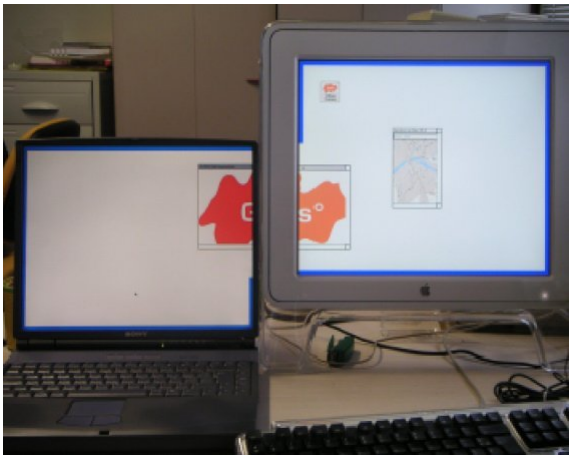

Fig. I.15: *Avec le système I-AM, la fenêtre d'une application s'exécutant sur une machine A peut être affichée en partie sur le dispositif d'affichage de la machine A et en partie sur le dispositif d'affichage d'une machine B, les machines A et B étant reliées via un réseau. Il est bien sur également possible d'afficher cette fenêtre uniquement sur un des deux dispositifs d'affichage [Lachenal, 2004].*

Dans cette configuration, il est courant que des utilisateurs se positionnent en face l'un de l'autre, et une interface classique qui comporte un haut et un bas devient inappropriée. DiamondSpin propose donc une boite à outils permettant à des utilisateurs de collaborer autour d'une table grâce à diverses interfaces qui sont utilisables quelque soit la position des utilisateurs (figure I.14).

Au même titre que les autres SDG, DiamondSpin met en œuvre le multiplexage des entrées.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
DiamondSpin		✓			

I.4.6 Systèmes complets

Nous nous intéressons dans cette section aux systèmes qui mettent en œuvre à la fois de la redirection et le multiplexage des entrées et des sorties. Ces systèmes permettent à un ou plusieurs utilisateurs d'interagir librement avec différents dispositifs d'interaction sur différents dispositifs d'affichage.

Les systèmes présentés dans cette section ne sont pas forcément multi utilisateurs. Cependant, ils permettent tous de gérer des dispositifs de pointage multiples. Ils présentent de ce fait les bases nécessaires à la réalisation de collecticiels.

Ubit Mouse Server

Ubit [Lecolinet, 2003a] est une boite à outils graphique basée sur X-Window qui utilise les graphes de scènes pour créer des interfaces graphiques. Cette architecture moléculaire rend possible la création d'interfaces graphiques distribuées (intégration et séparation des sorties).

L'extension Ubit Mouse Server (UMS) [Lecolinet, 2003b] permet de gérer plusieurs pointeurs sur une même surface. Il étend les fonctionnalités du serveur X. Il peut fonctionner avec toutes les applications X, mais seules les applications compatibles (c'est à dire les applications capables d'utiliser les structures d'évènements étendues de UMS) peuvent en tirer pleinement partie. En d'autres termes, les applications compatibles vont pourvoir identifier les évènements provenant des différents dispositifs de pointage alors que les autres applications vont les mélanger.

D'autre part, l'UMS fait également de la redirection des entrées : si deux surfaces sont couplées, il est possible de déplacer un pointeur d'une surface à l'autre. Notons que, grâce à l'architecture moléculaire de Ubit, il est également possible de déplacer une interface d'une surface à l'autre.

Au niveau des sorties, Ubit permet d'associer plusieurs serveurs X. Il devient alors possible, grâce au fonctionnement moléculaire de Ubit, de déplacer une interface d'un serveur X à un autre. Ubit permet donc la redirection, l'intégration et la séparation des sorties.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Ubit (+UMS)	✓	✓	✓	✓	✓

Machine abstraite d'interaction

I-AM (Interaction Abstract Machine) [Lachenal, 2004] est un gestionnaire de ressources d'interaction. Il assure le partage de ces ressources entre les différentes applications (figure

I.15). Il permet entre autre d'afficher la fenêtre d'une application à cheval entre deux dispositifs d'affichage gérés par I-AM (même si ces deux dispositifs d'affichages sont liés à des machines différentes).

Du point de vue des dispositifs de saisie, I-AM permet la gestion de plusieurs dispositifs de pointage. Ces dispositifs peuvent à priori être utilisés sur tous les dispositifs d'affichages gérés par I-AM (exception faite des dispositifs de pointage directs qui restent liés à une surface d'affichage). Ainsi, I-AM gère de manière transparente la réplication des évènements issus des dispositifs de pointage. Les applications clientes de I-AM reçoivent alors les évènements provenant des différents dispositifs de pointage grâce au multiplexage des entrées.

Bien que I-AM ne soit pas pensé pour une utilisation collaborative, il fournit les bases nécessaires au développement de collecticiels.

De la même manière que Ubit, I-AM met en œuvre la redirection, l'intégration et la séparation des sorties. La différence réside dans la gestion dynamique de l'environnement que permet I-AM. Tandis que les serveurs X sont liés « en dur »pour Ubit, I-AM – qui n'est pas basé sur le serveur X – permet une découverte dynamique des ressources d'affichage et peut les associer à la volée.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
I-AM	✓	✓	✓	✓	✓

I.5 Conclusion

I.5.1 Récapitulatif

Le tableau I.1 propose un récapitulatif des observations faites dans ce chapitre.

On peut faire plusieurs observations sur ces résultats. Tout d'abord, on constate que la redirection des entrées et des sorties est implémentée dans un certain nombre de systèmes, dont certains sont des produits matures et commerciaux.

A l'inverse, le multiplexage des entrées n'est implémenté que dans des travaux de recherche, ce qui témoigne d'un manque de maturité du domaine. Le frein principal au développement de ces systèmes est l'incompatibilité des interfaces graphiques actuelles. En effet, les *toolkits* graphiques qui sont utilisées à grande échelle ne sont pas capable de traiter plusieurs flux d'évènements d'entrée.

De la même manière que pour les entrées, l'intégration et la séparation des sorties sont beaucoup moins bien supportées que la redirection des sorties. Il faut cependant noter les grandes possibilités qu'offre X-Window puisqu'il permet l'intégration et la séparation des sorties ce qui en fait un système très en avance sur les autres systèmes de fenêtrages.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
Mono utilisateur					
PointRight Synergy	✓				
Ext. de Whizz		✓			
Multi écran					✓
Maxivista			✓		✓
RemoteJFC	✓		✓	✓	
INDIGO	✓				
EasyLiving	✓		✓		
X-Window	✓		✓	✓	✓
Multi utilisateur					
VNC Col. VNC	✓		✓		
Wincuts	✓		✓		
Augmented surfaces	✓	✓			
DistView COAST VIRTUS	✓	✓	✓		
Multi utilisateur – SDG					
MMM		✓			
Pebbles	(✓)	✓			
MID		✓			
MightyMouse	✓				
Dynamo		✓			
DiamondSpin		✓			
Multi utilisateur – systèmes « complets »					
Ubit (+UMS)	✓	✓	✓	✓	✓
I-AM	✓	✓	✓	✓	✓

Tab. I.1: *Possibilités de partage des entrées et des sorties des différents systèmes présentés.*

I.5.2 Niveaux d'abstraction

La figure I.16 montre les différences de granularité des communications pour la redirection et le multiplexage des entrées. Certains systèmes se contentent de dupliquer des évènements en leur appliquant un minimum de traitements tandis que d'autres utilisent des évènements abstraits permettant des échanges entre des systèmes incompatibles. Enfin, certains systèmes maximisent les traitements sur les évènements et échangent ainsi des commandes plutôt que des évènements.

La figure I.17 montre quant à elle les différences de granularité des informations échangées pour la redirection, l'intégration et la séparation des sorties. Du niveau le plus basique, et aussi le plus générique : l'interface est un ensemble de pixels jusqu'au niveau le plus évolué, et aussi le moins réutilisable : l'interface représente un ensemble de données (un modèle).

↑ Commandes (évènements interprétés) :
INDIGO, Extension de Whizz, DistView, COAST, VIRTUS

Evènements abstraits :
I-AM, VNC, MightyMouse, PointRight, Synergy, Pebbles, Diamondspin

Evènements systèmes augmentés :
MID, UBIT (UMS), MMM, MID, Dynamo, Augmented surfaces

Evènements systèmes :
Easy living (Terminal services), X-Window, RemoteJFC

Fig. I.16: *Granularité des informations échangées pour le partage des entrées.*

↑ Synchronisation des modèles :
I-AM, INDIGO, DistView, COAST, VIRTUS

Commandes d'affichage :
X-Window, UBIT, RemoteJFC

Champs de pixels :
VNC, Wincuts, EasyLiving (Terminal services), Maxivista

Fig. I.17: *Granularité des informations échangées pour le partage des sorties.*

I.5.3 Nos critères de caractérisation

Les critères de caractérisation que nous avons présenté (redirection et multiplexages des entrées, redirection, intégration et séparation des sorties) nous ont permis de grouper les systèmes étudiés. Les systèmes appartenant à un même groupe ayant des caractéristiques similaires au niveau de leurs fonctionnements mais également au niveau de leurs objectifs et de leurs champs d'application.

I.5.4 Bilan

Dans ce chapitre, nous avons analysé un ensemble de systèmes permettant d'utiliser et de partager des dispositifs d'entrée et de sortie multiples. Rappelons que l'objectif du travail présenté dans ce mémoire est l'étude des interactions lors de l'utilisation du mur-écran et plus généralement dans un contexte multi surface.

Dans la suite de ce mémoire, nous tenterons d'aider à l'adaptation des interfaces actuelles en proposant un modèle de *drag-and-drop* (manipulation directe) supportant les dispositifs d'entrée multiples mais aussi les dispositifs d'affichages multiples. En effet, dans les systèmes présentés ici, seul un nombre limité supportent à la fois la redirection et le multiplexage des entrées mais un seul – Augmented surface [Rekimoto and Saitoh, 1999] – permet des interactions poussées entre les différentes surfaces.

Dans le chapitre suivant, nous allons donc nous intéresser plus particulièrement au *drag-and-drop*. Nous présenterons différents travaux qui ont pour objectif d'étendre le *drag-and-drop* dans différents contexte où son utilisation est délicate, et en particulier dans les environnements d'affichages distribués.

Chapitre II

NOUVELLES TECHNIQUES D'INTERACTION DE MANIPULATION DIRECTE

Sommaire

Après nous être intéressés aux dispositifs d'affichage et de pointage multiples, nous allons nous attarder sur les techniques d'interaction adaptées à ces environnement et plus particulièrement à l'une des plus emblématiques : le *drag-and-drop*. Depuis quelques années, un nombre important de techniques dérivées du *drag-and-drop* est apparu. L'objectif de ce chapitre est de présenter l'ensemble de ces techniques.

II.1 Drag-and-drop

II.1.1 Historique

Depuis l'apparition du paradigme WIMP (Windows, Menu, Icon, Pointer), le *drag-and-drop* peut être considéré comme l'une des techniques d'interaction les plus abouties du modèle de manipulation directe. Alors que la plupart des interactions disponibles dans les interfaces sont de type *indirect* (utilisation de menus et de dialogues), le *drag-and-drop* apporte aux interfaces une plus grande facilité d'utilisation.

Avec le temps, le modèle de manipulation directe a considérablement évolué. Depuis le *click-and-drag* précurseur que l'on trouve dans *Smalltalk* (méthode de sélection d'un élément dans un menu), en passant par le *drag-and-drop* apparu initialement chez Apple [Horn], jusqu'aux extensions récentes telles que le *pick-and-drop*, ce modèle a vu ses capacités décuplées.

II.1.2 Définition

Définition II.1 *Le drag-and-drop est une technique d'interaction consistant à déplacer un ou plusieurs objets provenant d'un composant graphique (source) présent dans une interface utilisateur vers un autre composant (cible) de cette interface en utilisant un dispositif de pointage.*

Selon l'acception du domaine, un composant graphique peut-être défini comme une fenêtre dotée d'un comportement interactif. Le composant source et le composant cible sont uniques, même si le nombre d'objets déplacés lors du drag-and-drop peut être variable.

Il en découle qu'une opération de *drag-and-drop* est proche d'une tâche de pointage. Du fait de cette similitude, les propriétés d'une tâche de type *drag-and-drop* et d'une tâche de pointage sont les mêmes. Nous pouvons ainsi appliquer la loi de Fitts aux tâches de type *drag-and-drop* et utiliser des notions telles que l'indice de difficulté, l'indice de performance ou le *throughput* [Douglas et al., 1999].

II.1.3 Quand le drag-and-drop atteint ses limites

Malgré ses qualités, le *drag-and-drop* est parfois limité. On peut distinguer deux sortes de limites : d'une part, les limites physiques et d'autre part, les limites dues aux implémentations faites du *drag-and-drop*.

Les limites physiques

Les limites physiques du *drag-and-drop* sont apparues avec l'évolution du matériel. L'apparition de nouveaux matériels a limité les possibilités d'action du *drag-and-drop*. Un premier exemple est l'apparition des surfaces augmentées (grandes surfaces d'affichage disposant d'un système de pointage direct). En effet, il peut s'avérer difficile de réaliser un *drag-and-drop* sur

un surface qui peut atteindre $1,5m$ de diagonale. Garder le système de pointage en contact avec la surface est délicat lorsque de grands mouvements doivent être accomplis. Une autre limitation est apparue avec les systèmes à affichages multiples : par exemple, une configuration comprenant une machine disposant d'une surface tactile – tablet PC – et d'un affichage supplémentaire (donnant lieu à un bureau étendu) ne permet pas à son utilisateur de jouir de toutes les possibilités offertes par le *drag-and-drop*. En effet, le système de pointage direct ne couvre qu'une partie de l'espace de travail et il est donc impossible de déplacer un objet d'une surface à l'autre en utilisant le système de pointage direct.

Les limites des implémentations

Les limites dues aux différentes implémentations du *drag-and-drop* se résument essentiellement à l'absence de support des manipulations entre différentes machines. En effet, transférer des objets d'une machine à une autre est une opération qui est souvent coûteuse : certains utilisateurs optent pour l'envoi de courriels, d'autres utilisent le partage de fichiers de leur système d'exploitation, etc. Quelque soit la solution choisie, la simplicité de mise en œuvre n'a rien à voir avec celle d'un *drag-and-drop*.

Dans la section suivante, nous allons dresser un état de l'art des techniques proposées pour dépasser les limites du *drag-and-drop* évoquées ci-dessus.

II.2 Evolutions du drag-and-drop

Cette section présente les techniques dérivées du *drag-and-drop* de manière chronologique.

II.2.1 Pick-and-drop

Le *pick-and-drop* [Rekimoto, 1997] a été introduit pour permettre aux utilisateurs de déplacer des objets dans un environnement distribué. Alors qu'un *drag-and-drop* ne peut être réalisé que sur une machine (le point de départ et le point d'arrivé doivent se trouver sur la même machine), le *pick-and-drop* autorise un utilisateur à prélever un objet sur une machine et à le déposer sur une autre machine (figure II.1). Ceci est fait en donnant l'impression à l'utilisateur de prendre physiquement un objet sur une surface et de le déposer sur une autre surface.

En réalité, le *pick-and-drop* est plus proche du *copier-coller* que du *drag-and-drop*. En effet, on retrouve dans le *pick-and-drop* les deux mêmes étapes que dans le *copier-coller* : d'abord la sélection de l'objet à déplacer puis le dépôt de celui-ci. Cependant, le *pick-and-drop* et le *drag-and-drop* partagent un point fort par rapport à la technique du *copier-coller* : ils ne nécessitent pas d'avoir conscience d'un presse papier caché.

Le *pick-and-drop* a lui aussi des limitations : du fait de son implémentation, il ne peut être utilisé qu'entre deux surfaces supportant le même type de stylet et étant reliées au même réseau. En effet, à chaque stylet est attribué un identifiant et les données déplacées par *pick-and-drop* sont associées à l'identifiant du stylet utilisé pour l'opération et sont stockées sur un serveur *pick-and-drop*. De ce fait, le dépôt d'un objet se fait nécessairement avec le stylet qui a été utilisé pour le prélever.

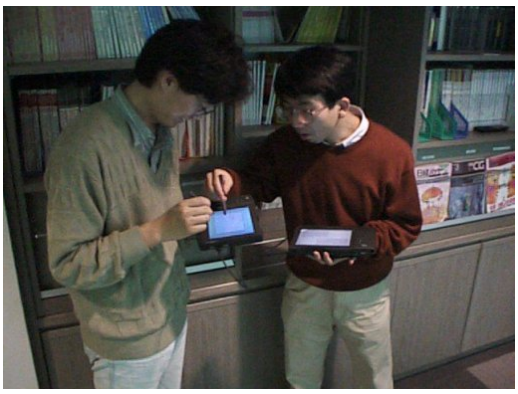

Fig. II.1: *Echange de données entre 2 PDAs [Rekimoto, 1997].*

II.2.2 Shuffle, Throw or take it

En 1998, Geißler [Geißler, 1998] proposait trois techniques pour travailler plus efficacement avec les murs interactifs. L'objectif était de proposer de nouvelles techniques d'interaction pour le DynaWall [institute] (combinaison de trois SmartBoard offrant une surface totale de 3072×768 pixels). En effet, effectuer des drag-and-drop sur une surface de 4,5 mètres de large par 1,1 mètre de hauteur pose des problèmes de déplacement physique de l'utilisateur.

La première technique proposée, le *shuffling*, permet de déplacer des objets d'une distance égale à leurs dimensions. Par exemple, un effectuant un petit *drag-and-drop* horizontal vers la droite sur une image de 200×300 pixels, celle-ci va se déplacer de 200 pixels vers la droite.

L'auteur propose ensuite une méthode de lancer – *throwing*. Pour lancer un objet, l'utilisateur doit réaliser deux mouvements : un premier dans le sens opposé au lancer et un second de plus grande amplitude dans le sens du lancer. La différence de longueur entre les deux mouvements détermine la distance à laquelle sera projeté l'objet. De l'aveu même de l'auteur, cette technique demande de l'entraînement avant d'être « maîtrisée ».

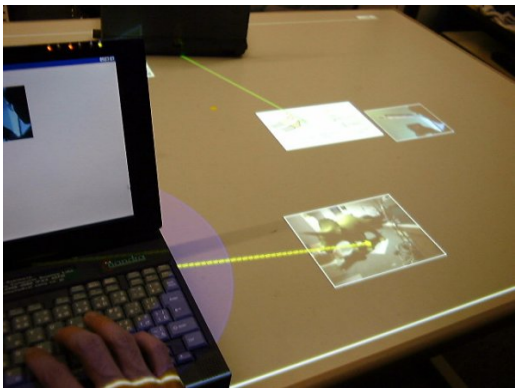

Fig. II.2: *Un utilisateur déplaçant une image de son ordinateur portable vers la table augmentée en utilisant un hyperdragging [Rekimoto and Saitoh, 1999].*

La troisième technique – *taking* – est une application du *pick-and-drop* décrit précédemment au DynaWall. Pour discriminer cette technique des précédentes, l'utilisateur doit rester environ une demi-seconde sur l'objet avec son stylo avant que l'objet ne disparaisse et que l'utilisateur ne puisse le déposer ailleurs sur le DynaWall.

II.2.3 Hyperdragging

L'*hyperdragging* [Rekimoto and Saitoh, 1999] est une extension permettant de réaliser des opérations de *drag-and-drop* entre, par exemple, un ordinateur portable et une surface partagée. Cette technique fait partie d'un projet de surface augmentée proposant un espace de travail continu. L'objectif est de permettre aux utilisateurs de partager facilement des données entre leurs ordinateurs portables, les surfaces augmentées murales et horizontales (tables) et des objets réels.

Pour cela, l'espace de travail est constitué de caméras permettant la reconnaissance des objets et de leurs positions. La position des écrans est donc connue et la continuité des mouvements entre les différents affichages peut être assurée.

L'utilisation de l'*hyperdragging* est transparente : l'utilisateur commence par effectuer un *drag-and-drop* classique et lorsque le curseur atteint le bord de son écran, il rejoint la surface partagée la plus proche et l'objet peut être déposé. Afin de limiter la confusion dans le cas où plusieurs utilisateurs effectueraient un *hyperdragging* en même temps, le pointeur qui est déporté sur la surface partagée est lié visuellement à l'ordinateur commandant le pointage (par une simple ligne dessinée au-dessus de l'espace de travail, voir figure II.2). Cette aide visuelle est appelée « ancrage du curseur ».

II.2.4 Drag-and-pop & Cie

Drag-and-pop

L'idée principale à l'origine du *drag-and-pop* [Baudisch et al., 2003] est « si tu ne peux pas aller à la cible, la cible viendra à toi ». Le principe du *drag-and-pop* est de détecter le début d'un *drag-and-drop* et de rapprocher de manière temporaire les cibles probables de l'opération (figure II.3). Celles-ci sont déterminées en fonction de la direction initiale du *drag-and-drop*.

Rapprocher les cibles probables d'une opération répond à certaines limitations du *drag-and-drop*. En particulier, si l'utilisateur dispose de plusieurs surfaces d'affichage et qu'une ou plusieurs de ces surfaces utilisent un système de pointage direct, déplacer un objet d'une surface à l'autre peut être évité grâce au rapprochement de la cible sur la même surface que l'objet déplacé.

Le prototype du *drag-and-pop* a été implémenté sur un bureau simulé et permet le déplacement des icônes de ce bureau. Dans le cas du déplacement d'un fichier à la corbeille par exemple, l'utilisation du *drag-and-pop* est simple : l'utilisateur initie le déplacement comme s'il s'apprêtait à réaliser un *drag-and-drop* classique. Après avoir déplacé l'icône de quelques pixels (suffisamment pour déterminer la direction du mouvement), les cibles potentielles du déplacement (celles-ci sont déterminées en fonction de leur compatibilité avec l'objet déplacé) sont virtuellement rapprochées du pointeur de la souris. Les icônes « réelles » sont grisées et une bande élastique les unit aux icônes fantômes qui se trouvent à proximité de l'objet en cours de déplacement. L'utilisateur peut ensuite déposer son objet sur l'une des cibles qui a été rapprochée. Une fois l'objet déposé, les icônes fantômes et les bandes élastiques disparaissent.

Cette technique se limite donc aux cas où le déplacement d'un objet se termine sur un autre objet. Dans l'exemple ci-dessus, déplacer une icône sur une autre icône peut se faire en utilisant le *drag-and-pop* mais le déplacement d'une icône sur le bureau (réarrangement du bureau) ne peut se faire q'en utilisant le *drag-and-drop* classique. Pour ce faire, le *drag-and-pop* peut être désactivé : l'utilisateur peut faire disparaître les icônes fantômes en s'éloignant d'elles et continuer son opération comme un *drag-and-drop* classique.

Drag-and-pick

Parallèlement au *drag-and-pop*, Baudisch a proposé le *drag-and-pick* [Baudisch et al., 2003] qui est une technique similaire permettant d'activer un objet à distance. Alors que le *drag-and-pop* se déclenche en faisant un *drag-and-drop* sur un objet du bureau, le *drag-and-pick* se déclenche en faisant un *drag-and-drop* sur une partie vide du bureau. Dans ce cas, de la même manière que pour le *drag-and-pop*, un groupe d'icône est sélectionné dans la direction du mouvement de l'utilisateur et est dupliqué à proximité du pointeur de l'utilisateur. Les icônes dupliquées sont reliées à leurs « originales »avec les mêmes bandes élastiques qui sont utilisées pour le *drag-and-pop* (voir la figure II.3). L'utilisateur termine son mouvement sur l'icône qu'il souhaite activer. L'opération complète est équivalente à un double-clic sur une icône.

L'utilisation du *drag-and-pick* permet donc de cliquer sur une icône qui soit n'est pas accessible – si le pointage direct n'est disponible que sur une seule surface d'affichage d'un bureau étendu – soit est difficilement accessible – un coin d'une surface augmentée de grande taille.

Vacuum

Bezerianos et al. [Bezerianos and Balakrishnan, 2005] ont poussé le principe plus loin : le *vacuum* permet, comme son nom l'indique, d'aspirer les objets de l'espace de travail. La sélection des objets aspirés est contrôlée par l'utilisateur grâce à un *widget* circulaire qui apparaît lorsque le *vacuum* est utilisé. Ce *widget* permet de définir facilement l'arc d'influence du *vacuum* mais surtout de le corriger si celui-ci venait à être erroné. En un seul mouvement, l'utilisateur active le *vacuum* et corrige son arc d'influence, le cas échéant.

Les objets « aspirés »sont représentés sous la forme de d'objets temporaires qui peuvent ensuite être manipulés en lieu et place de leurs originaux.

Le point commun de ces techniques d'interaction est qu'elles se proposent de rapprocher les objets de l'utilisateur. Nous allons voir que d'autres techniques ont une approche opposée : plutôt que de déplacer l'objet à proximité du pointeur, elles permettent de déplacer le pointeur à proximité de l'objet.

II.2.5 Drag-and-throw & push-and-throw

Le *drag-and-throw* et le *push-and-throw* [Hascoët, 2003] sont des méthodes de lancer destinées à limiter le nombre d'erreurs en offrant plus de contrôle à l'utilisateur. L'objectif est de permettre à l'utilisateur de connaître exactement le résultat de son lancer avant de le valider.

Bien que l'idée du lancer ne soit pas nouvelle pour déplacer des objets sur un espace de travail, ces deux techniques d'interaction ont été proposées pour pallier aux défauts des techniques de lancer [Geißler, 1998]. Les techniques proposées jusque là souffraient d'une mauvaise précision et, par conséquent, d'un taux d'erreur important.

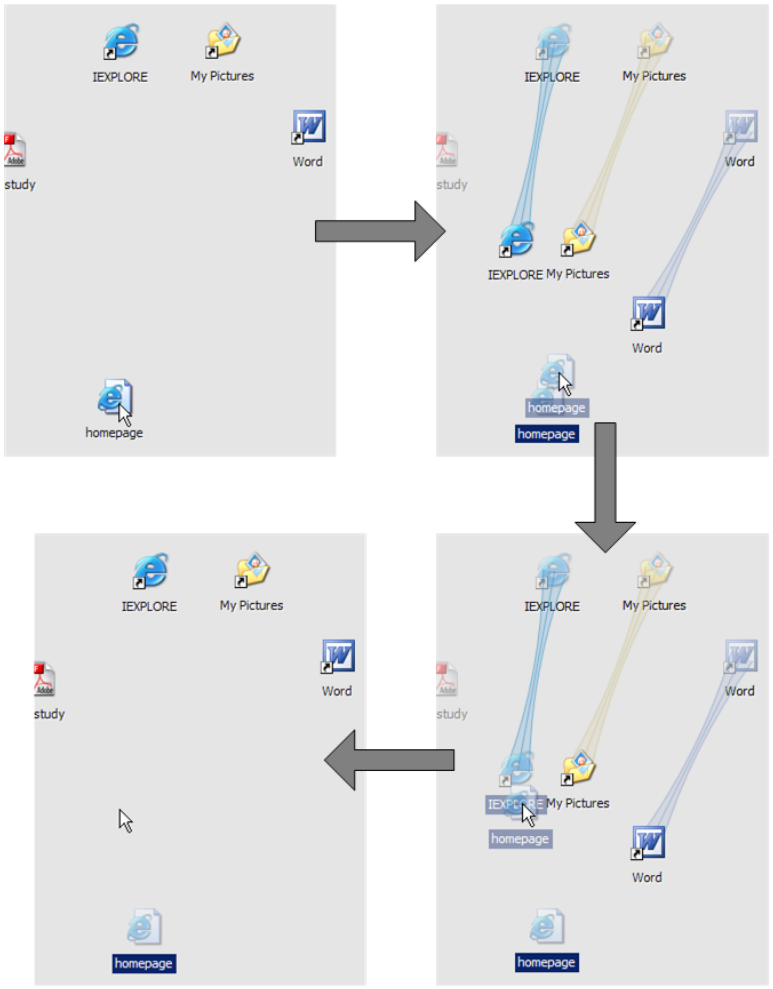

Fig. II.3: *Exemple d'utilisation du drag-and-pop.*

L'objectif du *drag-and-throw* et du *push-and-throw* est donc d'améliorer la précision des techniques de lancer tout en réduisant le taux d'erreurs sur des dispositifs à double-écran. Pour atteindre ce résultat, différents *feedbacks* visuels et métaphores sont utilisés. Dans leur conception initiale, ces techniques permettent de déplacer un objet d'une surface d'affichage vers une autre mais ne permettent pas de déplacer un objet à l'intérieur d'une surface.

Les métaphores

L'objectif étant la précision, une métaphore particulièrement adaptée est celle du tir à l'arc qui a été utilisée pour le *drag-and-throw*. En utilisant cette technique, l'utilisateur arme son « arc »et ajuste son mouvement. Ainsi, lorsqu'un utilisateur veut déplacer une icone vers un dossier, par exemple, il déplace son pointeur à partir de l'icone dans la direction opposée au dossier (figure II.4 (a) et (b)).

Le *push-and-throw*, quant à lui, repose sur la métaphore du pantographe. Le pantographe est un instrument de dessin qui permet de faire des agrandissements ou des réductions en utilisant les propriétés de l'homothétie pour conserver les proportions entre le dessin original et la copie [Wikimedia]. L'utilisateur du *push-and-throw* dispose donc d'une vue miniature de

l'espace cible. Ses mouvements dans la zone miniature sont amplifiés et retranscrits sur l'écran cible.

Les feedbacks

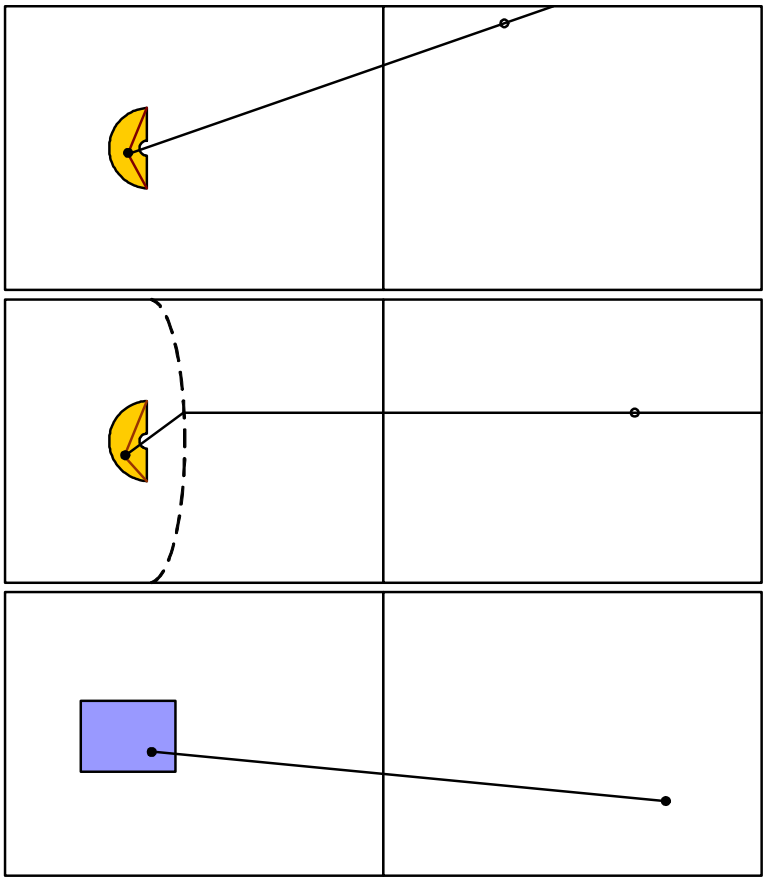

Fig. II.4: *De Haut en bas : (a) drag-and-throw avec une trajectoire naïve, (b) drag-and-throw avec une trajectoire corrigée par une ellipse et (c) push-and-throw.*

Qu'il utilise le *drag-and-throw* ou le *push-and-throw*, l'utilisateur dispose d'un ensemble de *feedbacks* visuels (II.4) durant toute la période d'ajustement de son lancer. Ces *feedbacks* vont l'aider dans son opération :

- La trajectoire : c'est la trajectoire que va emprunter l'objet lorsque le lancer sera validé. La trajectoire dispose également d'une aimantation (*snapping*) : les objets de la zone cible peuvent aimanter la trajectoire, ce qui permet de faciliter l'ajustement de la trajectoire par l'utilisateur. La trajectoire est représentée visuellement par une ligne au dessus de l'espace de travail.
- La mire : c'est le point de la trajectoire sur lequel l'objet lancé va être déplacé.
- La zone de décollage : c'est la zone dans laquelle l'utilisateur peut déplacer son pointeur durant l'opération de lancer. Cette zone de l'affichage source se projette sur l'ensemble de l'affichage destination. L'affichage de la zone de décollage aide l'utilisateur à comprendre où il peut déplacer son pointeur et lui donne des indications quant à la tension maximale

de son « arc »(*drag-and-throw*) ou quant au rapport entre la vue miniature et l'affichage destination (*push-and-throw*).

Evaluation préliminaire

Une évaluation préliminaire de ces deux techniques de lancer a donné des résultats encourageants [Hascoët, 2003]. En effet, le *drag-and-throw* et le *push-and-throw* se sont montrés plus rapides que le *drag-and-drop*. Cependant, les taux d'erreurs du *drag-and-drop* étaient sensiblement meilleurs. Le besoin de tests plus poussés s'est donc fait sentir. Nous présenterons de nouveaux tests dans le chapitre IV.

II.2.6 Stitching

Stitching [Hinckley et al., 2004] est une technique d'interaction qui permet de commencer un *drag-and-drop* sur une surface et de le terminer sur une autre surface. Cette technique se destine aux appareils mobiles disposant d'un système de pointage direct et pouvant communiquer par un réseau sans fil. Cependant, elle peut être utilisée avec un matériel plus classique.

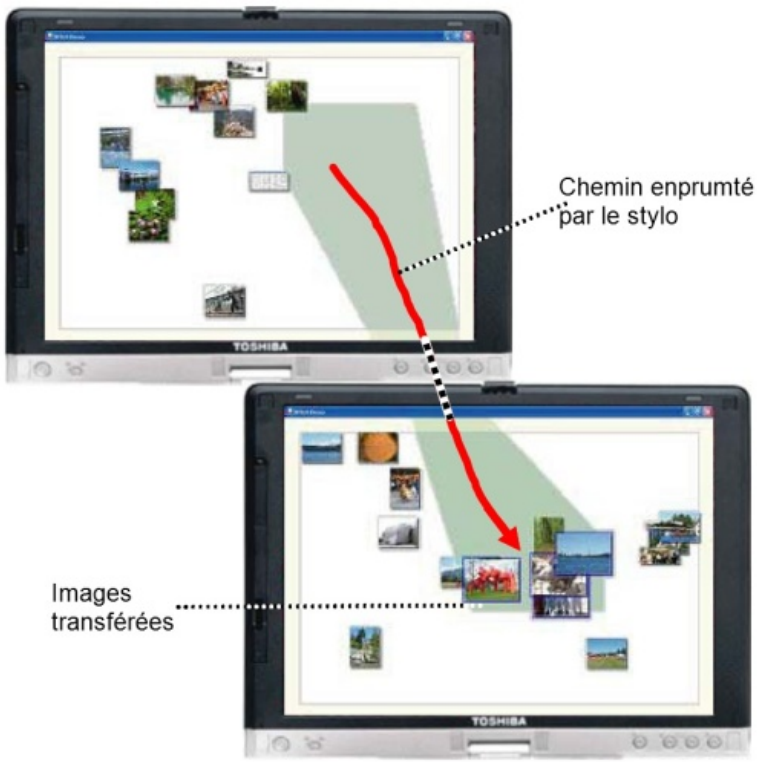

Fig. II.5: *Exemple d'utilisation de Stitching [Hinckley et al., 2004].*

Pour utiliser *Stitching*, un utilisateur commence un *drag-and-drop* sur un objet d'une surface, atteint un bord de la surface, interrompt son *drag-and-drop*, et le reprend sur le bord d'une autre surface (figure II.5).

Cette technique est assez proche du *pick-and-drop*. Cependant, elle est moins contraignante que le *pick-and-drop*. En effet, les deux *drag-and-drop* qui forment une opération complète sont synchronisés à la fin de l'opération en fonction de la direction des *drag-and-drop*. De ce fait, il est possible de réaliser avec deux systèmes de pointage différents. Ceci est utile si les 2 appareils ne supportent pas le même type de stylo par exemple.

II.3 Comparaison

Nous avons noté principalement trois points sur lesquels les techniques présentées dans ce chapitre divergent.

Le premier point est l'approche adoptée par ces techniques. D'un côté, nous avons le *drag-and-throw* et le *push-and-throw* qui se proposent de déplacer le pointeur vers la cible. Et d'un autre côté, nous avons le *drag-and-pop* qui, à l'inverse, déplace les cibles vers le pointeur de l'utilisateur. La conséquence est que l'attention de l'utilisateur se porte sur l'objet source de la manipulation directe pour les techniques « cible vers pointeur » et se porte sur l'objet cible pour les techniques « pointeur vers cible ».

Le second point de divergence est la nécessité pour l'utilisateur de se réorienter. En effet, avec le *drag-and-pop*, après un mouvement de quelques pixels, des icones temporaires apparaissent et l'utilisateur doit analyser ce groupe d'icones pour y retrouver la cible qu'il désire atteindre. C'est l'assimilation de ce nouvel environnement que nous appelons réorientation. D'autre part, le *drag-and-throw* et le *push-and-throw* nécessitent que l'utilisateur observe en permanence la mire afin qu'il adapte son mouvement jusqu'à atteindre la cible.

Enfin, le troisième point de divergence est visuel. En effet, les différentes techniques présentées utilisent des retours visuels (*feedback*) parfois très différents.

Le tableau IV.1 (p. 93) présente une comparaison des techniques d'interaction présentées dans ce chapitre ainsi que des nouvelles techniques que nous présentons dans le chapitre IV. Ce tableau comparatif s'appuie sur les trois points ci-dessus (approche, réorientation, *feedback*).

Dans le chapitre qui suit, nous allons étudier les différentes implémentations du *drag-and-drop* et proposer un modèle instrumental pour le *drag-and-drop* et ses extensions. Nous serons alors plus en mesure de comparer plus en profondeur les techniques décrites ici.

En étudiant les différentes implémentations du *drag-and-drop*, nous allons voir apparaître différentes étapes :

1. Initialisation
2. Détection du *drag*
3. *Drag*
4. *Drop*
5. Finalisation

Bien qu'on ne dispose pas des détails d'implémentation de la plupart des techniques présentées dans ce chapitre, ces étapes s'appliquent très bien à toutes les techniques de type *drag-and-drop* comme nous le verrons à l'issue du chapitre III. Il faudra cependant prévoir des *feedbacks* supplémentaires afin de proposer un modèle qui couvre l'ensemble des méthodes de type *drag-and-drop*.

Chapitre III

UN MODÈLE INSTRUMENTAL POUR LA MANIPULATION DIRECTE

Sommaire

Nous avons vu dans le chapitre précédent le grand nombre de techniques qui ont été proposées pour faire évoluer le *drag-and-drop*. L'objectif de ce chapitre est l'analyse des implémentations du *drag-and-drop* afin de voir dans quelle mesure ces techniques innovantes peuvent s'intégrer dans les mécanismes actuels.

III.1 Comparaison des implémentations du drag-and-drop

L'implémentation du *drag-and-drop* comporte deux aspects complémentaires : le modèle d'émission d'évènements de suivi du *drag-and-drop* que nous abordons en détails dans cette section et le mécanisme de transfert des données au moment du déposer que nous étudierons plus succinctement par la suite. Ces deux aspects sont présents dans les quatre systèmes présentés : Mac OS X/Carbon, X-Window/GTK+, Microsoft Windows, et Java/Swing.

Nous avons opté pour ces quatre systèmes afin d'obtenir une vue d'ensemble représentative des implémentations du *drag-and-drop*. Nous avons été aidé dans cette tâche par Renaud Blanch, qui a analysé le *drag-and-drop* tel qu'il est implémenté dans Carbon (Mac OS X). Son aide nous a été précieuse car nous ne disposions pas de machine Apple et que Renaud Blanch possède une très bonne connaissance des systèmes Mac.

III.1.1 Les étapes

Les modèles d'émission des évènements font généralement intervenir trois principaux acteurs : la *source* (composant d'où l'objet est déplacé), le *système* (*toolkit* et/ou système de fenêtrage sous-jacent), et la *cible* (composant sur lequel l'objet est déplacé). Par ailleurs, ce mécanisme peut se décomposer en 5 étapes :

- *l'initialisation* qui est effectuée une fois pour toute et qui permet de déclarer les sources et les cibles de *drag-and-drop* au système ;
- *le début* de l'interaction (activé en général par la détection d'un mouvement de la souris, un bouton particulier pouvant être maintenu enfoncé) durant lequel l'objet d'un *drag-and-drop* particulier est spécifié, alors que tous les mécanismes nécessaires à sa réalisation sont mis en place ;
- *le glisser* (*drag*) qui consiste en un déplacement de la souris, bouton toujours enfoncé, vers des cibles potentielles en les notifiant pour qu'elles puissent se déclarer intéressées ;
- *le dépôt* (*drop*) qui consiste à effectivement acheminer des données de la source à la cible ;
- la finalisation qui permet aux sources et aux cibles de redevenir passives pour le *drag-and-drop*.

Dans cette section la description des différents mécanismes s'appuient sur des diagrammes qui présentent, du point de vue de ces trois acteurs et de ces cinq phases, l'articulation fonctionnelle du drag-and-drop.

III.1.2 MacOS/Carbon

Le système Mac OS X d'Apple se caractérise par la présence de deux *toolkits* graphiques : Carbon, héritée des versions précédentes de Mac OS, et Cocoa, héritée de NextStep. Le fonctionnement du *drag-and-drop* de Cocoa ne présente pas d'originalité par rapport aux autres systèmes présentés ici. On en trouvera une description précise dans [Apple developer connection, b]. Nous considérons ici uniquement le fonctionnement du *drag-and-drop* pour la *toolkit*

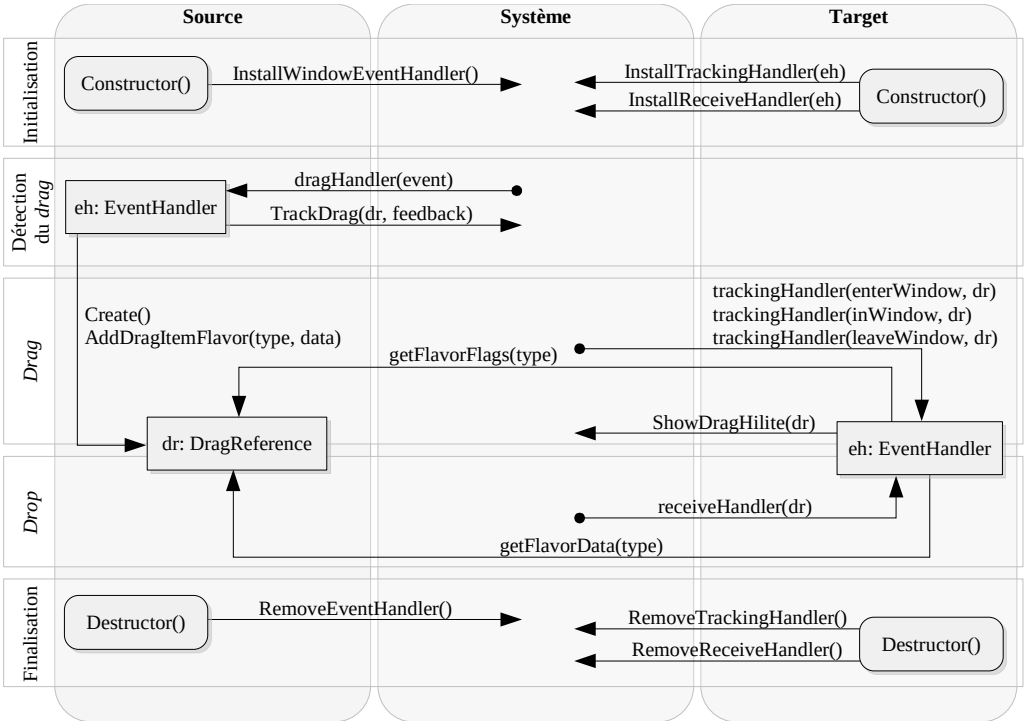

Fig. III.1: *Fonctionnement du drag-and-drop sous Carbon.*

Carbon [Apple developer connection, a] celle-ci étant la *toolkit* historique d'Apple qui reste très largement utilisée (iTunes, application phare de la suite iLife d'Apple est développée à l'aide de Carbon). Les fonctionnalités de *drag-and-drop* sont fournies par le *DragManager*, une partie de la *Human Interface Toolbox* implémentée par Carbon. La figure III.1, sur laquelle l'acteur système inclut le *DragManager*, donne un aperçu du processus.

C'est le *DragManager* qui permet initialement à une application d'enregistrer ses composants graphiques comme étant des cibles pour le *drag-and-drop* grâce aux fonctions *InstallTrackingHandler()* et *InstallReceiveHandler()*. La source doit, quant à elle, détecter dans sa routine habituelle de traitement des évènements le début d'un glisser. Elle peut utiliser à cette fin une fonction mise à disposition par le *DragManager*, *WaitMouseMoved()*, qui permet de discriminer un simple clic d'un début de glisser.

Lorsque le début d'un glisser est reconnu, il est de la responsabilité de la source de créer deux objets représentant les données déplacées :

- un objet matérialisant les données manipulées, l'accès à cet objet se faisant par une instance de *DragReference* ;
- un objet représentant graphiquement ce qui est déplacé à l'écran, par l'intermédiaire d'une région surlignée par le système qui sera attachée au pointeur (objet de type *RgnHandle*).

La source passe alors la main au *DragManager* en lui fournissant les deux objets précédents par l'intermédiaire de la fonction *TrackDrag()* qui ne retournera qu'à la fin du *drag-and-drop* en spécifiant si celui-ci a réussi ou échoué. Durant le glisser, le *DragManager* se charge d'afficher le *feedback* lié à la source (*drag-over effect*). Il notifie aussi les applications survolées qui se sont enregistrées comme cibles potentielles de trois principaux évènements : l'entrée du

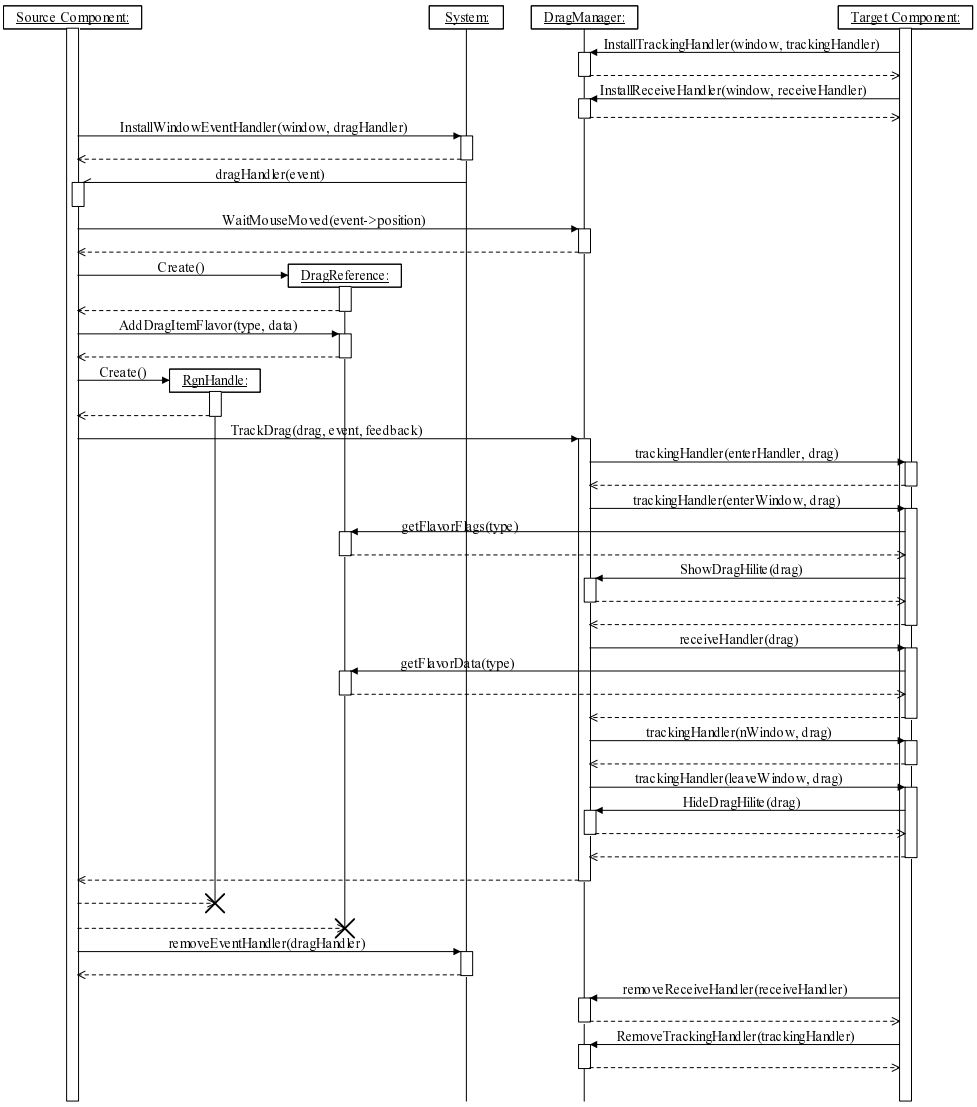

Fig. III.2: *Diagramme de séquence d'un drag-and-drop sous Carbon.*

pointeur dans la zone de la cible, le survol de la cible (évènement reproduit périodiquement) et la sortie du pointeur de la zone de la cible. La cible peut alors vérifier la compatibilité des données manipulées en testant les formats sous lesquels les données sont fournies grâce à *GetFlavorFlags()* et adapter son *feedback* (*drag-under effect*). Le *DragManager* permet un *feedback* par défaut (mise en surbrillance de la bordure d'une zone donnée par la cible). Il est géré simplement par la cible grâce à l'appel de *ShowDragHilite()* et *HideDragHilite()*.

Enfin, lors du dépôt, le *DragManager* notifie l'application concernée si elle est enregistrée. La cible peut alors tester le format des données par un appel à *GetFlavorFlags()*, récupérer la taille et le contenu de certaines données par un appel à *GetFlavorDataSize()* et *GetFlavorData()* et enfin, suivant la valeur qu'elle renvoie au *DragManager*, notifier de son acceptation ou non du dépôt. Si le dépôt est refusé, le *DragManager* produit un dernier *feedback* en ramenant le *drag-over* à l'endroit d'origine du *drag-and-drop* pour notifier l'utilisateur du refus. Dans tous les cas, la source reprend finalement la main et peut détruire les objets qu'elle a créés au début de l'interaction.

III.1.3 X-Window/GTK+

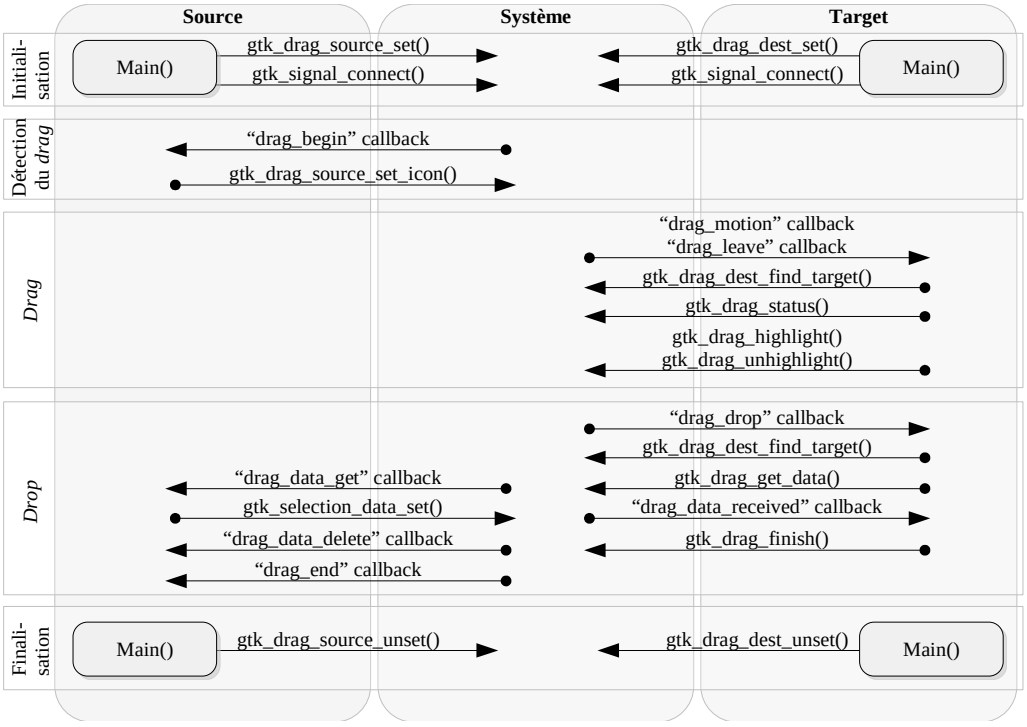

Fig. III.3: *Fonctionnement du drag-and-drop avec GTK+.*

Le modèle de *drag-and-drop* de la *toolkit* GTK+ [Homepage], s'appuie sur le mécanisme de communication inter-application fourni par *XToolkit Instrinsics* issu lui-même du *Inter-Client Communications Conventions Manuel* (ICCCM) de X-Window. Le mécanisme de communication inter-applications de base de X-Window repose sur les notions de propriété, d'atomes et de sélection. La notion de selection a un rôle essentiel dans le modèle de *drag-and-drop* de GTK+ du fait qu'elle est utilisée pour le transfert des données.

Dans le contexte de l'étude des différents modèles d'implémentation du *drag-and-drop*, la *toolkit* GTK+ présente la particularité de ne pas être orientée objet. De ce fait un système de *callbacks* est utilisé.

Dans un premier temps, les composants (*widgets*) doivent s'enregistrer comme source ou cible pour les opérations de *drag-and-drop* (figure III.3 et III.4). Une source potentielle s'enregistre par un appel à la méthode *gtk_drag_source_set()* en spécifiant en paramètre les boutons de la souris avec lequel le *drag-and-drop* peut être effectué, la liste des formats supportés ainsi que les actions supportées (copie, déplacement et/ou lien). De son côté, une cible potentielle s'enregistre par un appel à la méthode *gtk_drag_dest_set()* en spécifiant la liste des formats et les actions supportés. En complément de ces enregistrements, il est nécessaire de connecter les *callbacks* associés aux différents évènements intervenant durant une opération de *drag-and-drop*. Un composant source connecte les *callbacks drag_begin*, *drag_data_set*, *drag_data_get*, *drag_date_delete* et *drag_end* tandis qu'un composant cible connecte les *callbacks drag_motion*, *drag_leave*, *drag_drop* et *drag_data_received*.

C'est le système qui détecte le début d'un *drag-and-drop* et qui notifie le composant source grâce au *callback drag_begin*. Le cas échéant, il est alors possible de personnaliser l'icone se trouvant sous le pointeur de l'utilisateur (*drag-over feedback*). La forme du pointeur est quant à elle maintenue par le système. Celle-ci dépend du type de l'action effectuée (copie, déplacement ou lien). Après cet appel de la *callback drag_begin*, le composant source ne sera plus notifié d'aucun évènement avant la fin de l'opération, lors du transfert des données.

Pendant l'opération de *drag-and-drop*, les composants cibles au dessus desquels passe le pointeur sont notifiés via l'appel des *callbacks drag_motion* et *drag_leave*. Un composant cible peut alors vérifier la compatibilité des formats supportés par les composants source et cible en appelant la méthode *gtk_drag_dest_find_target()* qui recherche le premier format de données supporté par les deux composants en fonction des formats spécifiés lors de l'enregistrement de ces composants comme source et cible potentielles. L'acceptation ou non par le composant cible est spécifié grâce à la méthode *gtk_drag_status()* et, si un format commun est trouvé, le composant peut alors activer ou désactiver la surbrillance par un appel aux méthodes *gtk_drag_highlight()* et *gtk_drag_unhighlight()* (*drag-under feedback*).

Lorsque l'opération se termine (dépôt de l'objet), la cible est notifiée dans un premier temps par le *callback drag_drop*. A ce moment, le composant cible recherche un format commun avec la source par un appel à *gtk_drag_dest_find_target()*. Il de demande alors les données dans ce format en appelant *gtk_drag_get_data()*, ce qui a pour effet de déclancher l'appel au *callback drag_data_get* du composant source. Celui-ci peut alors transmettre les données sous la forme d'une selection (au sens de X-Window). Le *callback drag_data_received* du composant cible est alors appelé, permettant à celui-ci de récupérer les données. Le composant cible confirme alors la fin de l'opération en appelant *gtk_drag_finish()* en spécifiant si l'opération s'est correctement déroulée et si le composant source doit supprimer les données transférée (cas de la copie). Le cas échéant, le *callback drag_data_delete* du composant source est appelé et, pour terminer, le système supprime le *drag_icon* utilisé pour le *drag-over effect* et appel le *callback drag_end* du composant source.

Il faut noter qu'un certain nombre d'opérations peuvent être gérées automatiquement par le système, offrant ainsi un comportement par défaut pour le composant cible. Il est en effet possible, au moment de l'enregistrement du composant cible (*gtk_drag_dest_set()*), de demander une réponse automatique aux *callbacks drag_motion* et *drag_drop* ainsi que la mise en surbrillance automatique du composant cible.

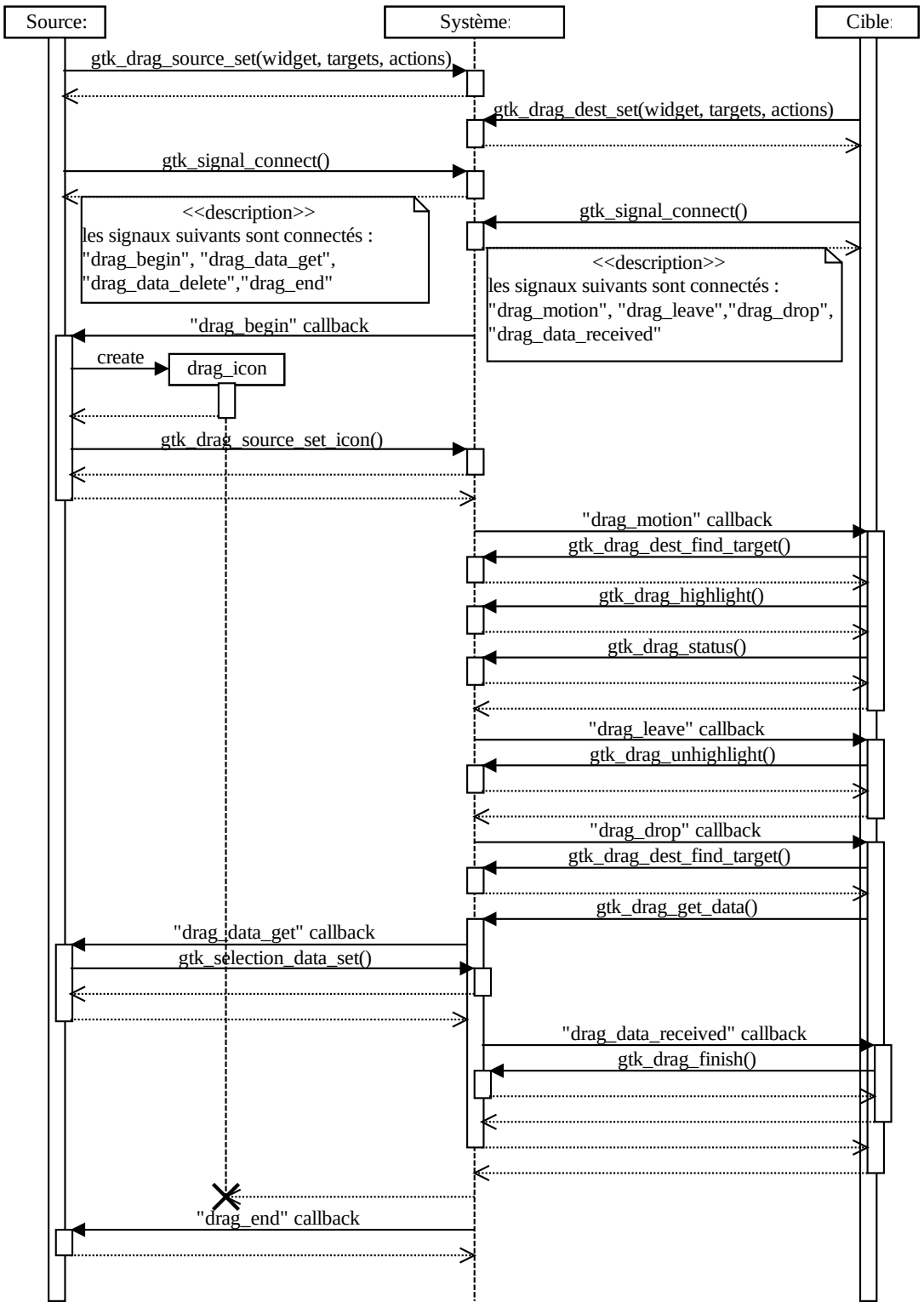

Fig. III.4: *Diagramme de séquence d'un drag-and-drop avec GTK+.*

III.1.4 Windows/OLE

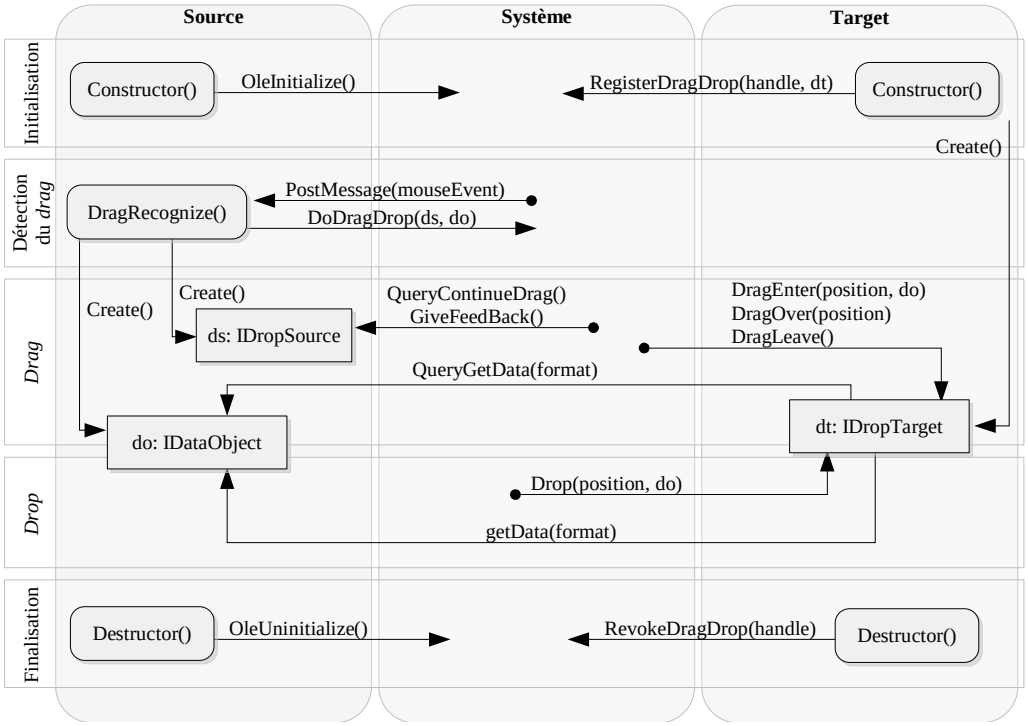

Fig. III.5: *Fonctionnement du drag-and-drop sous OLE.*

Nous ne parlerons pas ici de la technique originelle utilisée dans Microsoft Windows (envoi du message *WM_DROPFILE*). Elle a été supplantée par la technologie plus récente utilisée par MS Windows pour la gestion du copier-coller et du drag-and-drop : OLE (Object Linking and Embedding) [Brown] que nous présentons ici.

Chronologiquement, la première action permettant à l'utilisateur de réaliser un *drag-and-drop* est l'enregistrement du composant cible comme étant une cible potentielle : pour cela, il doit créer un objet instance d'une classe implémentant l'interface *IDropTarget* et appeler la fonction *RegisterDragDrop()*. De son côté, le composant source doit activer le support de l'OLE par un appel à la fonction *OleInitialize()* (figures III.5 et III.6).

Lorsqu'un mouvement de glisser est reconnu – après analyse des évènements souris reçus par le composant source –, le composant source crée deux objets *dropSource* et *dataObject*. L'objet *dropSource* est une instance d'une classe implémentant l'interface *IDropSource* et l'objet *dataObject* est une instance d'une classe implémentant l'interface *IDataObject*. Le composant source appelle ensuite la fonction *DoDragDrop()* en fournissant en paramètres ces deux objets. Cet appel de fonction est synchrone : le composant source ne reprendra véritablement la main que lorsque le *drag-and-drop* sera terminé ou annulé (l'objet *dropSource* obtiendra cependant la main ponctuellement pour les *feedbacks*).

Le système gère ensuite le déroulement de l'opération. Le composant source, au travers de l'objet *dropSource*, a la possibilité d'interrompre le *drag-and-drop* grâce à l'appel régulier par le système de la fonction *QueryContinueDrag()*. Il peut également gérer des aides visuelles grâce à l'appel régulier par le système de la fonction *GiveFeedBack()*. De son côté, le composant

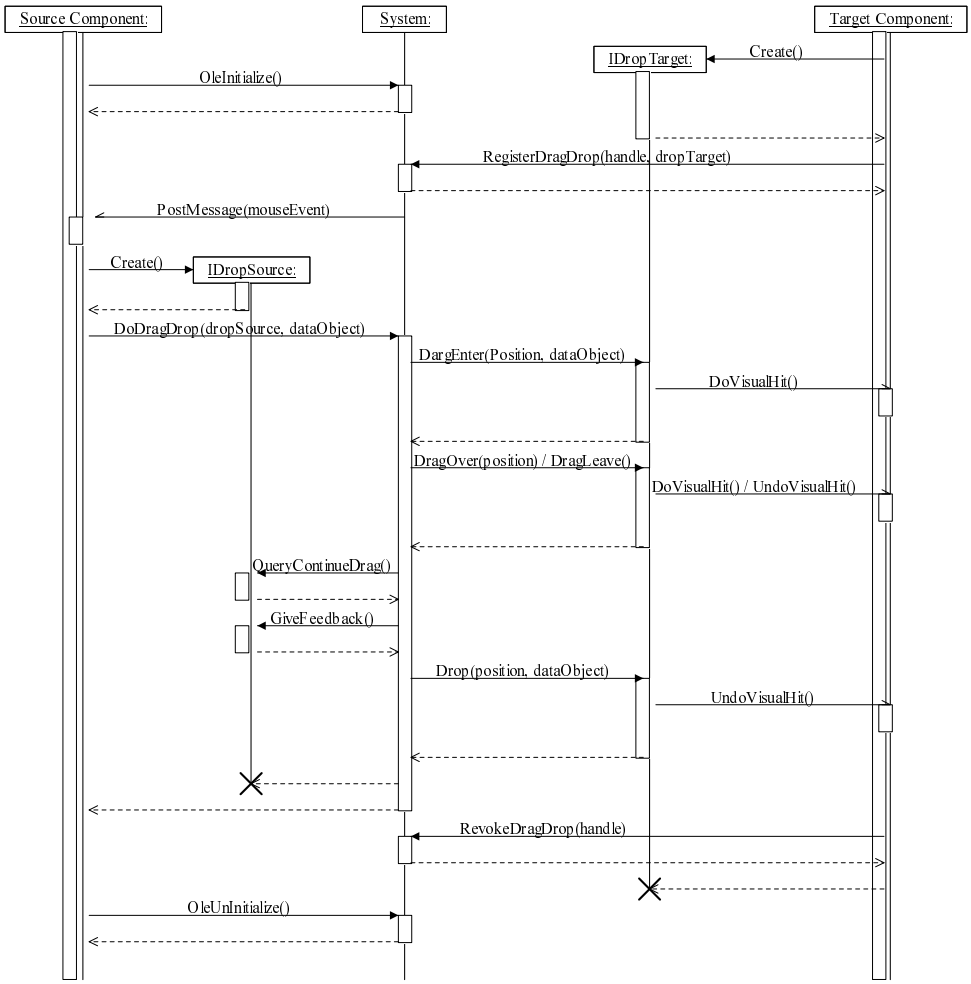

Fig. III.6: *Diagramme de séquence d'un drag-and-drop sous OLE.*

cible est notifié de l'avancement de l'opération : lorsque le pointeur entre dans le composant, le système appelle la fonction *DragEnter()* de l'objet *dropTarget* associé au composant cible. Il peut alors accepter ou refuser le glisser en fonction des données qui lui sont associées et de leurs formats. Le composant cible est également notifié lorsque le pointeur bouge ou qu'il quitte le composant par un appel du système respectivement aux fonctions *DragOver()* et *DragLeave()*.

Lorsque l'utilisateur termine le *drag-and-drop*, le système appelle la méthode *Drop()* de l'objet *dropTarget* associé au composant cible. Celui-ci peut alors, en accédant à l'objet *dataObject*, obtenir les données à transférer dans le format désiré. Il faut noter que le composant source est seulement notifié de la fin du *drag-and-drop* par le fait que les objets *dropSource* et *dataObject* sont détruits par le système.

Lors de la destruction des composants, le composant source doit désactiver le support de l'OLE par un appel à *OleUnitialize()* et le composant cible doit se retirer de la liste des cibles potentielles en appelant *RevokeDragDrop()*.

III.1.5 Java/Swing

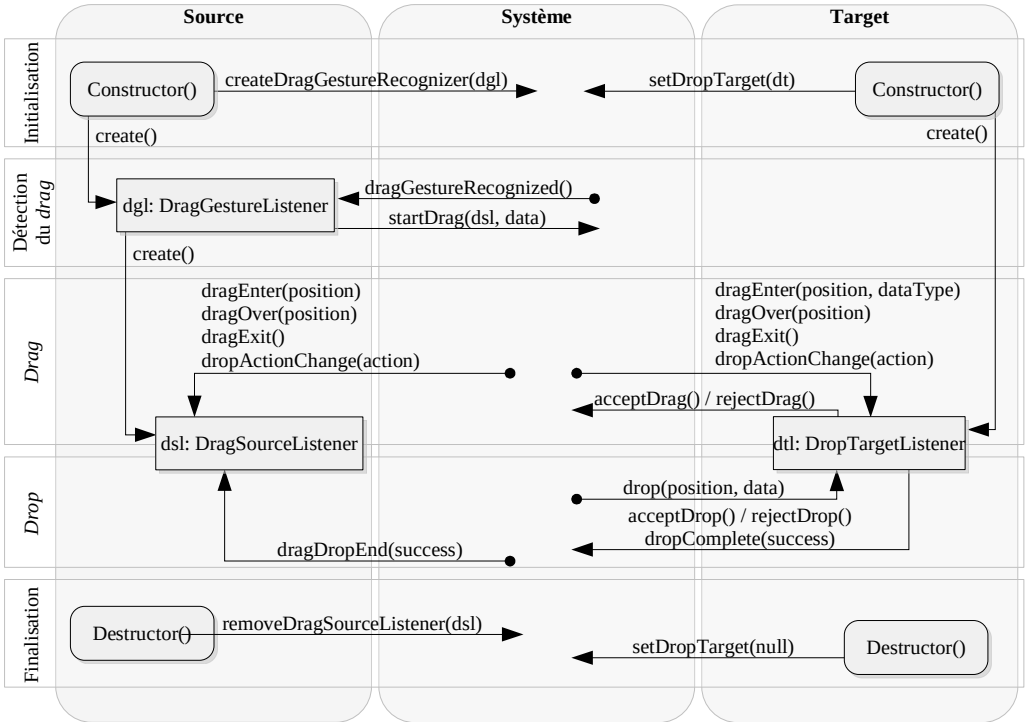

Fig. III.7: *Fonctionnement du drag-and-drop sous AWT/Swing.*

Le modèle d'évènements de Java étant basé sur le principe de délégation, la gestion des évènements est réalisée par des objets dédiés [De Lisa]. Lors de la construction du composant source, celui-ci doit construire une instance d'une classe implémentant l'interface *DragGestureListener* et s'y associer par un appel à la fonction *createGestureRecognizer()* de la classe *DragSource*. C'est le système qui a la responsabilité de détecter les évènements souris initialisant un *drag-and-drop* et c'est lui qui notifie ensuite le composant source au travers du *DragGestureListener* créé (figures III.7 et III.8).

De son côté, le composant cible doit s'enregistrer comme cible potentielle d'un *drag-and-drop*. Pour cela, il doit initialiser sa propriété *dropTarget* (via l'accesseur *setDropTarget()*) en créant une instance d'une classe implémentant l'interface *DropTargetListener*.

Lorsque le système reconnaît un glisser, il notifie le *DragGestureListener* du composant source via la méthode *dragGestureRecognized()*. Pour démarrer à proprement parler le *drag-and-drop*, le *DragGestureListener* doit appeler la méthode *startDrag()* de l'objet *DragGestureEvent* reçu en paramètre par la fonction *dragGestureRecognized()*. Une instance d'une classe implémentant l'interface *DragSourceListener* aura été créée et aura été passée en paramètre à la fonction *startDrag()*. Cet objet sera ensuite notifié des différents évènements ayant lieu au cours du *drag-and-drop*.

Au cours du *drag-and-drop*, les composants source et cible sont notifiés des mêmes évènements. Pour chaque évènement, le composant cible est systématiquement notifié le premier. À chaque cycle d'évènements, le composant cible doit accepter ou refuser le *drag-and-drop* : lors des messages *dragEnter()* et *dragOver()* en appelant les méthodes *acceptDrag()* ou *rejectDrag()*, et lors de l'évènement *drop()* en appelant *acceptDrop()* ou *rejectDrop()*. Le composant

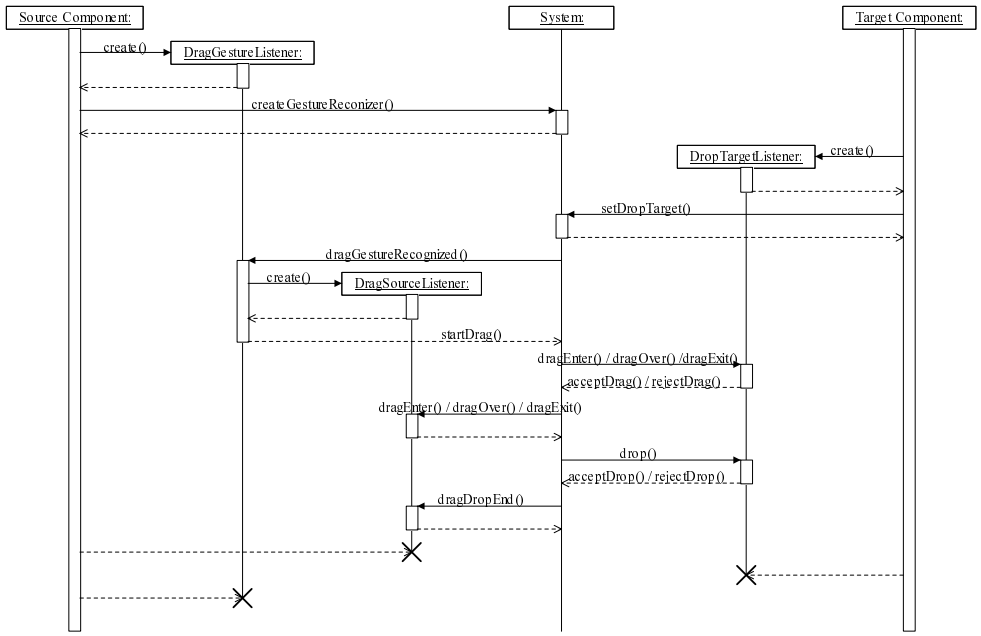

Fig. III.8: *Diagramme de séquence d'un drag-and-drop sous AWT/Swing.*

source est ensuite notifié de l'évènement et de l'acceptation ou du rejet de l'opération par le composant cible : il peut ainsi mettre à jour les indications visuelles (notamment changer le pointeur de la souris).

Pour pouvoir accepter ou refuser le glisser ou le déposer, le composant cible dispose d'un accès aux données à transférer dans les messages *dragEnter()* et *drop()*.

III.1.6 Structures de données et protocoles de transfert

Tous les systèmes décrits ici permettent de transférer des données sous différents formats simultanément. En effet, si on décide de glisser une image depuis un navigateur Web vers différentes applications, on s'attend à des comportements différents : si on dépose l'image dans un logiciel de retouche photographique, on s'attend à ce que l'image bitmap soit insérée dans l'espace de travail. Mais si on dépose l'image dans un éditeur html, alors il serait préférable que l'url de l'image soit insérée dans le code html.

Cet exemple illustre la nécessité de coder les données transférées par *drag-and-drop* sous plusieurs formats. On comprend ainsi pourquoi le déplacement d'un texte depuis un navigateur vers un éditeur de texte brut ou un traitement de texte aura un effet différent. L'éditeur de texte brut sélectionnera le format *plain text* tandis que le traitement de texte sélectionnera le format *rich text*. Certains logiciels offrent d'ailleurs la possibilité à l'utilisateur de forcer la sélection d'un format particulier au travers de la commande « collage spécial »(ceci s'applique au copier-coller et non au *drag-and-drop* mais les données entrant en jeu dans ces deux protocoles de transfert sont les mêmes).

Mac OS X

L'objet qui matérialise les données lors du *drag-and-drop* est un conteneur auquel la source peut ajouter des données associées à un identifiant de type (*TEXT* est par exemple l'un des types prédéfinis pour du texte brut). La source peut déclarer certains types sans pour autant incorporer les données à l'objet. Dans ce cas, elle doit enregistrer une routine qui sera appelée par le *DragManager* pour fournir sur demande les données à la cible lorsqu'elle le réclamera.

C'est donc la cible qui doit déterminer si l'objet qu'on tente de lui déposer contient des données susceptibles de l'intéresser et éventuellement convertir ce que lui fournit la source dans un format qui lui convient mieux.

X-Window

Sous X-Window/GTK+, le transfert des données peut se faire en s'appuyant sur le mécanisme de X-Selection (il passe alors par le serveur X).

L'échange de données entre deux clients X étant un problème qui se retrouve dans diverses situations (copier-coller, drag-and-drop, ou autre), un protocole particulier a été développé à partir de Motif 2.0 pour unifier le transfert des données quel que soit le contexte dans lequel il s'effectue. Ce protocole, appelé UTM (*Uniform Transfer Protocol*) vise donc aussi bien le *drag-and-drop* que le copier-coller ou d'autres contextes d'échange de données entre applications. Il est basé sur des *callbacks* dédiées placées à la source et à la cible et qui s'utilisent mutuellement pour déterminer le meilleur format pour transférer les données de la source à la cible, avant de lancer le transfert réel qui fait à son tour intervenir d'autres *callbacks* dédiées. C'est ce mécanisme qui est utilisé par GTK+ pour le *drag-and-drop*.

Microsoft Windows

Dans le protocole OLE, les données sont encapsulées dans un objet instance d'une classe implémentant l'interface *IDataObject*. Ces données peuvent être stockées dans autant de formats différents que nécessaire. L'objet est créé par le composant source avant l'appel à la fonction *DoDragDrop()*. Le système prend ensuite en charge l'objet et le transmet aux composants cibles lorsque le pointeur de la souris les survole. Ces derniers peuvent ainsi accepter ou refuser le déposer en fonction des formats de données supportés par l'objet *dataObject*.

Lorsque le *drag-and-drop* est terminé ou qu'il a été annulé, le système détruit l'objet *dataObject*.

Java Swing

Du point de vue du programmeur, il faut gérer les données sous la forme d'un objet implémentant l'interface *Transferable* (Swing propose entre autres une classe *StringSelection* permettant de gérer des données sous forme de chaînes de caractères). Il est possible de décrire ces données sous plusieurs formats simultanément. Au niveau système, les données sont transférées au format MIME (*Multipurpose Internet Mail Extensions*).

III.1.7 Discussion

Cette présentation des différents modèles permet de dégager les points communs et les différences qu'ils comportent. Elle permet également de constater ici ou là la présence de

mécanismes meilleurs que d'autres à différents titres. Le but de cette section est de faire le bilan de ces aspects, puis de mettre en évidence les évolutions qui sont nécessaires pour que les différents modèles de *drag-and-drop* puissent s'unifier et se généraliser aux environnements d'affichage distribués.

Pour faciliter la discussion, nous numérotons ici les cinq principales étapes mises en évidence dans figure III.1, III.3, III.5, III.7 de la manière suivante : (1) Initialisation, (2) Début de glisser, (3) Suivi du glisser, (4) Dépôt, (5) Finalisation.

Etapes 3 et 4 : des différences importantes

Malgré une architecture générale similaire illustrée par ces cinq phases, c'est au cœur du *drag-and-drop*, dans les étapes 3 et 4, que les différences sont les plus conséquentes. Ainsi, au cours de l'étape 3, les évènements ou les messages reçus sont de trois sortes : entrée au-dessus d'une cible, déplacement sur la cible, et sortie de la cible. Ils ne sont pas émis dans tous les systèmes. Ces évènements (ou messages) sont à l'origine des deux types de *feedbacks* mentionnés précédemment que sont le *drag-over effect* (*feedback* côté source) et le *drag-under effect* (*feedback* côté cible). Mais de grosses différences apparaissent sur leur mise en oeuvre. Avec MS Windows/OLE, par exemple, le composant source ne gère pas le *drag-over effect* et le composant cible ne gère pas non plus de *drag-under*. Au contraire, les *toolkits* Carbon ou GTK+ permettent au composant source de spécifier le *drag-over effect* et proposent un mécanisme simple de gestion du *drag-under effect* par la mise en surbrillance d'un composant cible.

Dernier point important durant cette étape de glisser, les actions que l'utilisateur peut faire sont pratiquement les mêmes. Elles ont pour but de modifier la sémantique associée au *drag-and-drop*. Ainsi l'utilisateur peut agir pendant l'étape 3 dans tous les systèmes pour faire en sorte que le résultat du *drag-and-drop* soit une copie de l'objet source sur l'objet cible, un lien, ou encore un réel déplacement. Ce n'est pas tout, les modes opératoires sont cohérents en s'appuyant sur une combinaison des touches *ctrl*, *alt* et *shift* (*option*, *command* et *shift* sur Mac OS X/Carbon) pendant le déplacement. Par contre, même si l'annulation en cours de déplacement ou la demande d'aide sont possibles dans tous les systèmes, elles ne disposent pas systématiquement de types d'évènements dédiés. Par exemple, la gestion de l'annulation n'est pas prévue dans les modèles événementiels de Java/Swing et Mac OS X/Carbon.

Lors de l'étape 4 (dépôt), on retrouve un point commun à tous les système : la cible est notifiée en premier, accepte ou refuse le dépôt, et c'est seulement ensuite que le composant source est notifié de la fin du *drag-and-drop*. Sur le plan des données à transférer, tous les systèmes étudiés permettent de gérer statiquement ou dynamiquement l'envoi des données. Les données créées dynamiquement sont générées au moment de la demande par le composant cible, et ceci en fonction du format demandé. Malgré ces points communs, des différences importantes se situent au niveau (a) des modalités de stockage et de transfert (protocole réseau ou zone de mémoire partagée) et (b) des formats d'échanges et des protocoles selon lesquels ils peuvent être reconnus. Le problème posé par les modalités de stockage n'est pas simple. Si les données passent par le réseau, il risque d'y avoir des latences indésirables, et dans le cas du partage d'une zone mémoire le mécanisme ne pourra pas être étendu au cas d'un *drag-and-drop* distribué tel que le *pick-and-drop* par exemple. Enfin, les différences de formats d'échange et de protocoles de négociation de ces formats constituent une entrave à l'ouverture vers les extensions du modèle. La discussion de ces différences dépasse le cadre de

ce chapitre. Elles sont en effet héritées d'un problème bien plus général, qui reste ouvert malgré les efforts qui ont été investis pour s'y attaquer : les protocoles d'échange de données entre les applications.

Etapes 1, 2 et 5 : de nombreux points communs

Au delà de ces différences, deux éléments quasiment identiques dans les diverses implémentations étudiées sont les phases d'initialisation (1) et de finalisation (5). Dans tous les cas, les sources et les cibles potentielles doivent se déclarer au système qui sera chargé de prendre en main les autres étapes du *drag-and-drop*. Pour ce qui est de l'étape 5 de finalisation, tous les systèmes l'utilisent pour libérer les ressources utilisées pendant le glisser et remettre les sources et les cibles impliquées dans leur état initial.

L'étape 2 de détection de début de glisser est elle aussi abstraitement identique. Dans tous les cas, c'est toujours le composant source qui a la responsabilité de démarrer le *drag-and-drop* après analyse des évènements souris (cette analyse étant plus ou moins répartie entre le composant et le système selon les cas). Certaines petites différences subsistent néanmoins. On peut remarquer par exemple que Mac OS X/Carbon présente une particularité intéressante pour cette détection : il s'agit de la possibilité d'utiliser un tiers dédié au *drag-and-drop* (le *DragManager*), pour discriminer entre un simple clic et un début de glisser. GTK+ permet de choisir quels boutons de la souris peuvent être utilisés pour une opération de *drag-and-drop*. Cette différence n'est certes pas majeure, mais elle pose néanmoins un problème de cohérence et montre que la question : « quel est le bon évènement pour initier un glisser ? », mériterait un peu plus d'attention. On constate donc que le *drag-and-drop* est régi par un protocole qui, du moins abstraitement, est très semblable d'un système à l'autre. Cette homogénéité est sans conteste due à un ensemble de contraintes imposées par la technique d'interaction elle-même. L'étude des nouvelles techniques d'interaction qui visent à palier les difficultés de la transposition du *drag-and-drop* de la station de travail individuelle classique à des nouveaux dispositifs d'affichage et à de nouvelles situations d'usage, nous a montré que ces contraintes ne sont plus suffisantes.

Positionnement du drag-and-drop

Une grande partie des différences observées ici sont dues au fait la gestion du *drag-and-drop* est répartie différemment entre le système de fenêtrage et les applications. Par exemple, le *drag-and-drop* n'est pas géré par le système de fenêtrage dans les environnements X-Window. C'est la *toolkit* graphique qui le gère. La limite de ce choix est que le *drag-and-drop* ne fonctionne pas nécessairement entre deux applications n'étant pas basées sur les mêmes *toolkit*. Il peut par ailleurs y avoir un problème de cohérence des comportements entre les différentes *toolkits*, celles-ci ne gérant pas le *drag-and-drop* de la même manière.

Nous pensons qu'il est préférable que la plus grande partie possible de la gestion du *drag-and-drop* soit effectuée par le système de fenêtrage. D'une part, ceci assure plus de cohérence entre les différentes applications et *toolkits*. D'autre part, ceci permet de simplifier le support du *drag-and-drop* dans les applications et les *toolkits*.

Intégration des techniques avancées

Le protocole de *drag-and-drop* doit évoluer pour s'adapter aux nouvelles exigences des différentes extensions décrites dans le chapitre précédent. Pour permettre l'intégration de ces techniques, il faut envisager différentes modifications.

La première des évolutions majeures que doit subir le *drag-and-drop*, se situe au niveau de l'étape 4 de transfert des données. En effet, le support des transferts inter-machines paraît primordial. Cela ne remet pas en cause de façon profonde la gestion événementielle du *drag-and-drop*, même si un nouveau problème apparaît : il faut d'abord sélectionner l'affichage cible avant de pouvoir sélectionner le composant cible.

Cependant, des solutions à ce problème topologique ont d'ores et déjà été proposées, notamment dans le système PointRight [Johanson et al., 2002b]. Après la sélection de l'affichage cible, il faut alors ouvrir une communication entre les deux machines concernées et cela nécessite une révision profonde des systèmes classiques présentés précédemment. Le problème est d'autant plus délicat que les performances sont critiques. Car un *drag-and-drop* doit être immédiat et le succès de la technique d'interaction en dépend.

Toujours pour cette même étape 4, un autre obstacle à l'intégration de ces extensions au sein des systèmes actuels, est l'incompatibilité des données échangées et des protocoles utilisés. Ce problème rejoint la discussion de la section précédente. Il ne faut donc pas s'étonner que les extensions décrites ici, qui ne sont que des prototypes, n'aient jamais été réalisées dans un objectif de compatibilité avec les systèmes existants.

Enfin, les autres évolutions importantes à envisager concernent les modèles d'évènements. Ces modèles paraissent assez ouverts pour accueillir des extensions mais pour augmenter leur pouvoir d'expression sans augmenter dramatiquement leur complexité, il est clair qu'une révision et une homogénéisation s'imposent. Car les contraintes augmentent. Pour le *drag-and-throw* par exemple, il faudra pouvoir représenter une trajectoire pour calculer les cibles de manière indirecte, disposer de « fantômes »spécifiques (zone de décollage, trajectoire, mire) pour assurer les *feedbacks* spécifiques à cette technique. Autre exemple : pour le *pick-and-drop* il faudra bien assurer les *feedbacks* homogènes habituels même si le *pick* et le *drop* n'ont pas lieu sur des systèmes identiques. Pour le *drag-and-pop*, il faut encore intégrer d'autres techniques pour calculer les cibles potentielles et les mécanismes de *drag-under* doivent être particulièrement enrichis.

La construction d'un nouveau modèle de *drag-and-drop* fait l'objet de la suite de ce chapitre. La construction de ce nouveau modèle s'appuie sur les principes de l'interaction instrumentale ainsi que sur les observations faites jusqu'ici sur les modèles existant. Ce nouveau modèle a en effet pour objectif minimum de faire aussi bien que les modèles existant.

III.2 Modèle d'interaction instrumentale pour le drag-and-drop et ses évolutions

Le modèle d'interaction instrumentale [Beaudouin-Lafon, 2000] se prête très bien à la description du *drag-and-drop* et de ses évolutions. Le premier avantage de ce modèle est que son utilisation permet de clarifier les concepts d'interaction qui entrent en jeu. De plus, alors que l'interaction instrumentale a initialement été introduite comme un modèle d'interaction, elle peut également, selon l'auteur lui-même, être utilisée comme base pour un modèle d'implémentation.

Ce qui nous amène au second avantage de ce modèle. En effet, le modèle d'implémentation que nous proposerons dans la section III.4 est directement dérivé du modèle d'interaction décrit dans cette section. Nous y voyons l'assurance d'une implémentation cohérente et complète répondant aux besoins de l'interaction.

III.2.1 Principes de l'interaction instrumentale

Le modèle d'interaction instrumentale repose sur le concept d'*instrument d'interaction* [Beaudouin-Lafon, 2000]. Un instrument d'interaction peut être considéré comme un médiateur entre l'utilisateur et les objets du domaine. L'utilisateur agit sur l'instrument, qui transforme les *actions* de l'utilisateur en *commandes* agissant sur les objets correspondant. Les instruments ont des *réactions* qui permettent à l'utilisateur de contrôler ses *actions* sur l'instrument, et fournit des *feedbacks* lorsque les *commandes* sont transmises aux objets.

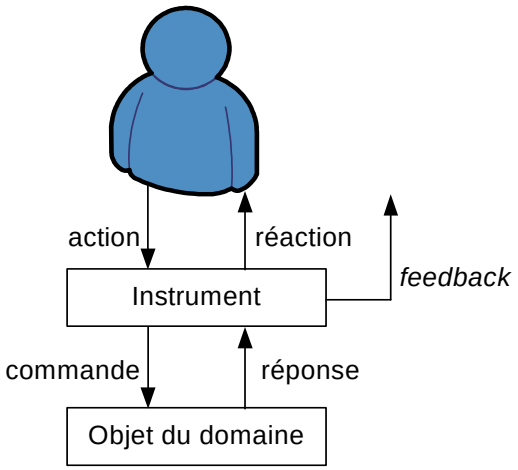

Fig. III.9: *L'instrument d'interaction joue le rôle de médiateur entre l'utilisateur et les objets du domaine [Beaudouin-Lafon, 2000].*

Un instrument décompose l'interaction en deux couches (figure III.9) :

- La couche d'interaction entre l'utilisateur et l'instrument : les *actions* de l'utilisateur sur l'instrument et les *réactions* de l'instrument.
- La couche d'interaction entre l'instrument et les objets du domaine : les *commandes* envoyées à l'objet et ses *réponses*, que l'instrument peut transformer en *feedbacks*.

III.2.2 Modèle d'interaction

Notre objectif est de définir un ensemble de *commandes* et de *réponses* pouvant intervenir entre un instrument et un objet lors d'une opération de type *drag-and-drop*. Ainsi, en spécifiant les échanges entre les objets du domaine et l'instrument, nous projetons de rendre l'instrument modulaire : un instrument de *drag-and-drop* pourrait être remplacé par un instrument de *pick-and-drop*, modifiant l'ensemble d'*actions* et de *réactions* intervenant entre l'utilisateur et l'instrument mais conservant le même ensemble de *commandes* et de *réponses* entre l'instrument et les objets du domaine. De cette manière, l'objet n'est pas affecté par le changement d'instrument.

III.2.3 Instruments et objets du domaine

Pour un *drag-and-drop* classique, l'instrument reste assez basique : un pointeur qui permet à l'utilisateur de déplacer des objets. Cependant, dès que l'on s'intéresse à d'autres techniques telles que le *drag-and-pop* ou le *drag-and-throw*, exhiber un instrument est très utile.

Pour les méthodes de lancer, les métaphores du tir à l'arc et du pantographe qui sont utilisées suggèrent clairement l'utilisation d'instruments. Pour le *pick-and-drop*, c'est un instrument du monde réel, un stylo, qui est utilisé pour effectuer une opération d'une surface d'affichage à une autre. Dans les autres cas, l'instrument est moins visible mais les *réactions* et les *feedbacks* peuvent toujours être inclus dans un instrument.

Les objets du domaine peuvent être de différentes sortes : des icones, du texte, des images, etc. mais ils doivent être différenciés en fonction de leurs rôles dans l'opération de type *drag-and-drop*. En effet, les objets du domaine peuvent avoir un rôle de *source* ou de *cible* : une source est déplacée au dessus de cibles potentielles jusqu'à ce que l'utilisateur mette fin à l'opération en déposant la source sur la cible choisie.

III.2.4 Actions, réactions et feedbacks

Il y a principalement quatre actions qui surviennent lorsqu'un utilisateur effectue une opération de type *drag-and-drop* :

- La sélection de la source et de la cible,
- La spécification du type d'action,
- Le transfert des données et
- L'annulation.

Lorsque les évolutions du *drag-and-drop* deviendront disponibles, une cinquième action sera également à prendre à compte :

- Le choix/l'activation d'un type d'instrument.

Cette dernière action peut être effectuée de différentes manières. Dans la plupart des cas, l'utilisateur voudra simplement spécifier l'utilisation d'un outil par défaut et ce choix apparaîtra donc dans son profil. Cependant, l'utilisateur peut avoir besoin de changer temporairement d'instrument. Par exemple, Baudisch a proposé le *drag-and-pop* et ses tests ont montré qu'il était plus performant que le *drag-and-drop* [Baudisch et al., 2003]. Cependant, le *drag-and-pop* ne fonctionne que si la cible peut être approchée du pointeur. En clair, il est possible de déplacer une icone sur une autre icone mais le *drag-and-pop* ne permet de déplacer une icone sur le bureau pour le réorganiser. C'est pourquoi Baudisch a également prévu que l'utilisateur puisse désactiver le *drag-and-pop* afin d'effectuer un *drag-and-drop* plus classique mais qui n'a pas la limitation évoquée précédemment. Une autre méthode de changement d'instrument sera par ailleurs exposée dans le chapitre IV.

Dans la majorité des systèmes de fenêtrage (voir la section III.1), les quatre actions principales listées ci-dessus se font très facilement. Pour sélectionner une ou plusieurs sources, l'utilisateur déplace simplement son pointeur sur la ou les sources et clique avec sa souris (ou tout autre système de pointage) et garde le bouton enfoncé. Un fantôme est ainsi obtenu (appelé aussi *drag icon*). Ce fantôme peut ensuite être déplacé au dessus des cibles. Pendant le déplacement, l'utilisateur a la possibilité de spécifier le type d'action qu'il veut effectuer grâce aux touches spéciales (*ctrl*, *alt* et *shift* sous Windows/Linux, *option*, *command* et *shift* sous MacOS). De manière générale, on dénombre trois types d'actions possibles : copier, déplacer,

créer un raccourci (ou un lien ou un alias). Enfin, l'utilisateur peut annuler une opération en cours en appuyant sur la touche *escape*.

Comme nous l'avons vu au début de ce chapitre, pour le *drag-and-drop* classique, les systèmes de fenêtrage fournissent des *feedbacks* appelés habituellement effets de *drag over* et de *drag under*.

Les effets de type *drag over* sont essentiellement des *feedbacks* qui apparaissent sur l'objet source. Il y a deux exemples typiques : la forme du pointeur et le fantôme. En effet, la forme du pointeur change en fonction de l'objet suivant qu'il est possible ou impossible de déposer l'objet source à la position du pointeur. D'autre part, le fantôme change également en fonction du type d'action qu'effectue l'utilisateur (copie, déplacement, lien). Certains systèmes de fenêtrage vont plus loin en fournissant des animations. Par exemple, pour indiquer qu'une opération a été annulée, une animation ramène le fantôme à sa position initiale. Il est intéressant de noter que même si le modèle du *drag-and-drop* est maintenant mature, tous les systèmes de fenêtrage ne proposent pas cette fonctionnalité. En l'absence d'animation, il est nettement moins évident que l'utilisateur comprenne bien qu'il vient d'annuler l'opération.

Les effets de *drag under* représentent les *feedbacks* fournis du côté des cibles. Ils transmettent des informations lorsqu'un fantôme se déplace au dessus d'une cible potentielle. La cible peut répondre de différentes manières, en modifiant sa forme ou sa couleur ou même, de manière plus sophistiquée en effectuant des actions. Par exemple, en déplaçant un fichier au dessus d'une arborescence de dossiers, lorsque le fichier reste quelques instants au dessus d'un certain dossier, le dossier se développe dans la vue de l'arborescence afin de permettre à l'utilisateur d'explorer récursivement la hiérarchie jusqu'à trouver le dossier désiré.

Si les effets de type *drag over* et *drag under* sont suffisants pour décrire les *feedbacks* dans le cas d'un *drag-and-drop* classique, ils ne sont pas suffisants pour la plupart de ses évolutions récentes. En effet, certaines évolutions introduisent de nouveaux *feedbacks* qui ne peuvent être considérés ni comme effet de *drag over* ni comme effet de *drag-under*. Ces *feedbacks* supplémentaires seront appelés des *feedbacks d'instrument*. Par exemple, dans le cas d'un *drag-and-pop*, des bandes élastiques sont utilisées pour aider l'utilisateur à identifier les cibles. Dans le cas du *push-and-throw* ou du *drag-and-throw*, une zone de décollage ainsi qu'une trajectoire sont affichées pour aider l'utilisateur à ajuster son mouvement. Des *feedbacks* similaires sont utilisés par les autres techniques.

Pour résumer, en observant les techniques de type *drag-and-drop* d'un point de vue instrumental, on note cinq actions principales et trois types de *feedback* : effet *drag under*, effet *drag over* et *feedback* d'instrument. Lorsqu'un utilisateur veut passer d'un instrument à un autre, les effets *drag under* et *drag over* devraient être préservés tandis que les *feedbacks* d'instrument devraient changer. Les effets *drag under* et *drag over* ne doivent pas dépendre de l'instrument utilisé.

III.3 Comparaison des instruments

Dans cette section, nous considérons que toutes les techniques de type *drag-and-drop* peuvent être implémentées dans un instrument. Ces instruments permettent à l'utilisateur de réaliser une partie ou la totalité des actions listées précédemment et fournissent différents types de *feedback*. Ces instruments diffèrent sur plusieurs points. Notre objectif est de déterminer les points importants pour la comparaison des instruments.

Un instrument joue le rôle de médiateur dans l'interaction entre l'utilisateur et les objets

du domaine [Beaudouin-Lafon, 2000] (figure III.9). Les interactions entre l'instrument et les objets du domaine (*commandes* et *réponses*) sont les mêmes pour tous les instruments implémentant des techniques de type *drag-and-drop*. A l'inverse, les interactions entre l'utilisateur et l'instrument (*actions*, *réactions*, *feedbacks*) sont spécifiques à chaque instrument.

Nous allons maintenant comparer les techniques décrites dans le chapitre II, à savoir le *drag-and-drop*, le *pick-and-drop* [Rekimoto, 1997], le *throwing* [Geißler, 1998], l'*hyperdragging* [Rekimoto and Saitoh, 1999], le *stitching* [Hinckley et al., 2004], le *drag-and-pop* [Baudisch et al., 2003], le *drag-and-throw* [Hascoët, 2003] et le *push-and-throw* [Hascoët, 2003].

III.3.1 Actions

Les actions de l'utilisateur différent de manière importante d'un instrument à l'autre. Pour le *drag-and-throw* et le *push-and-throw*, une mire est déplacée en direction de la cible. L'utilisateur doit en permanence observer les *feedbacks* de l'instrument – la mire en particulier – afin d'ajuster son mouvement. Il est en effet pratiquement impossible de déterminer à l'avance où le pointeur (stylo, doigt, pointeur de souris, etc.) doit être déplacé pour que la mire atteigne la cible.

Le *drag-and-pop* implique de la part de l'utilisateur un mouvement type *drag-and-drop* dont la direction est déterminante. Un ensemble de cibles dépendant de la direction initiale de ce mouvement est dupliqué à proximité du pointeur. Une fois que l'utilisateur a identifié la cible qu'il souhaite atteindre dans l'ensemble des cibles dupliquées, la position de celle-ci est stable et l'opération est simple à compléter. Cependant, étant donné que deux mouvements initiaux différents donnent des ensembles de cibles temporaires différents, l'utilisateur doit se réorienter une fois pour identifier la cible temporaire correspondant à la cible qu'il souhaite atteindre. Les bandes élastiques du *drag-and-pop* ont d'ailleurs été introduites pour aider l'utilisateur dans cette tâche.

Le *drag-and-throw* implique de la part de l'utilisateur un mouvement dans la direction opposée à la cible, ce qui est cohérent avec la métaphore du tir à l'arc qui est utilisée. A l'inverse, le *push-and-throw* ou le *drag-and-pop* implique un mouvement dans la direction de la cible. Nous remarquons ainsi que les actions peuvent également différer sur la direction du geste de l'utilisateur

Le *drag-and-drop* et l'*hyperdragging* mettent en œuvre des mouvements de *drag-and-drop* tout à fait classiques, la différence étant que l'*hyperdragging* permet au pointeur de quitter l'affichage auquel il est normalement cantonné. Le *pick-and-drop* et le *stitching* ont des approches assez similaires en décomposant l'action de l'utilisateur en deux parties : pour le *pick-and-drop*, l'utilisateur doit effectuer deux clics tandis que pour le *stitching*, il doit effectuer deux *drag-and-drop*, le premier se terminant au bord d'un affichage et le second commençant au bord d'un autre affichage. Précisons toutefois que ces deux dernières techniques diffèrent du *drag-and-drop* et de l'*hyperdragging* du fait qu'elles se destinent à des dispositifs de pointage directs.

Enfin, pour le *throwing* [Geißler, 1998], l'utilisateur doit faire deux gestes dont la direction et la longueur déterminent la position où l'objet sera lancé.

III.3.2 Réactions

Les réactions sont triviales et ne nécessitent aucun développement pour le *drag-and-drop*, le *pick-and-drop*, l'*hyperdragging* et le *stitching*.

Cependant, il y a une différences majeure entre les instruments de *push-and-throw* (et *drag-and-throw*) et de *drag-and-pop* : alors que les réactions du *push-and-throw* sont visibles sur la totalité de l'espace de travail, les réactions du *drag-and-pop* apparaissent dans l'espace moteur de l'utilisateur, c'est-à-dire à proximité de son pointeur. De ce fait, l'attention de l'utilisateur ne se porte pas sur les mêmes zones de l'affichage pour ces techniques.

En effet, en utilisant le *push-and-throw* ou le *drag-and-throw*, le pointeur se déplace dans une petite zone (la zone de décollage) et la mire, quant à elle, se déplace sur tout l'affichage. L'attention de l'utilisateur se porte sur la mire et donc en direction de la cible.

A l'inverse, en utilisant le *drag-and-pop*, l'attention de l'utilisateur se porte sur son pointeur à proximité de l'objet source. En effet, les cibles potentielles sont répliquées à proximité de l'objet source dans une zone qui peut être atteinte par l'utilisateur, que nous nommons l'espace moteur.

Nous appellerons « pointeur vers cible » (Tableau III.1) les instruments du type du *drag-and-throw*. L'objet source est répliquée (la mire) et est déplacée en direction des cibles potentielles. Nous les opposons aux instruments « cible vers pointeur » – comme le *drag-and-pop* – qui répliquent les cibles à proximité de l'objet source et du pointeur de l'utilisateur.

L'instrument de *throwing* utilise une approche « pointeur vers cible » étant donné qu'il projette l'objet source en direction de la cible.

III.3.3 Feedbacks

Seuls les *feedbacks* de *drag under* et de *drag over* sont utilisés pour le *drag-and-drop* et le *pick-and-drop*.

L'*hyperdragging* ajoute un *feedback* de trajectoire qui lie un pointeur et son dispositif d'affichage originel.

Le *stitching* utilise, quant à lui, un feedback mettant en évidence les bordures de l'affichage – un cadre de quelques pixels de largeur sur le pourtour de l'écran.

Le *throwing* ne fournit aucun *feedback* d'instrument. C'est certainement la raison principale de son manque de précision et de ses taux d'erreur très importants. En effet dans le contexte des méthodes de lancer, les *feedbacks* d'instrument peuvent être considérés comme des *feedbacks* de reconnaissance. Comme le suggérait olsen [Dan R. Olsen and Nielsen, 2001] :

> « Recognition feedback allows users to adapt to the noise, error, and miss-recognition found in all recognizer-based interactions ».

C'est pourquoi le *drag-and-throw* et le *push-and-throw* introduisent trois *feedbacks* : la zone de décollage, la trajectoire et la mire.

Le *drag-and-pop* offre quant à lui deux *feedbacks* : les cibles temporaires et les bandes élastiques qui relient les cibles temporaires à leurs originales.

Le tableau III.1 récapitule les différences observées entre les instruments.

Technique	**Réorientation** (action)	**Approche** (réaction)	**Retours visuels** (*Feedback*)
drag-and-drop	jamais	-	-
pick-and-drop	jamais	-	-
hyperdragging	jamais	-	trajectoire
stitching	jamais	-	bordure d'affichage
throwing	jamais	vers cible	-
drag-and-throw / push-and-throw	constamment	vers cible	zone de décollage et trajectoire
drag-and-pop	une fois	vers pointeur	icones temporaires et bandes élastiques

Tab. III.1: *Tableau comparatif des instruments.*

III.4 Vers un modèle d'implémentation unifié

Jusqu'à présent, les évolutions du *drag-and-drop* décrites dans le chapitre II ont été développées en tant que prototype en utilisant des modèles d'évènements *ad hoc*. On peut cependant noter que l'*hyperdragging* a été implémenté dans un produit commercial de Sony sous le nom de *Flying pointer* [Sony Japan].

Comme nous avons pu le voir dans la première partie de ce chapitre, les implémentations du *drag-and-drop* varient de manière assez importante entre les différents systèmes de fenêtrage. L'absence d'un modèle d'implémentation unifié du *drag-and-drop* entre les différentes plates-formes entrave l'intégration des évolutions ainsi que le support des environnements distribués. Jusqu'a présent, le *pick-and-drop*, l'*hyperdragging* et le *stitching* ont été implémentés dans des environnements d'affichage distribué.

L'objectif de notre travail est ici de proposer un modèle unifié et ouvert pour toutes les méthodes de type *drag-and-drop*. Ce qui inclut bien évidemment le *drag-and-drop*, mais aussi toutes les évolutions qui ont été récemment proposées pour étendre ses possibilités. Ces évolutions du *drag-and-drop* incluent des réactions supplémentaires ainsi que des *feedbacks* plus riches. Elles nécessitent également le support des environnements distribués. Notre volonté est de construire un modèle qui facilite l'intégration des évolutions récentes et futures tout en prenant en compte les implémentations existantes du *drag-and-drop*.

III.4.1 Principes

La partie la plus importante de notre modèle est le modèle d'évènements. Un aspect important de notre modèle est la **modularité** : les composants source et cible d'une opération doivent être notifiés des évènements pertinents et leur comportement ne doit pas être remis en cause par l'introduction d'une nouvelle technique.

Afin d'obtenir cette modularité, les éléments clés sont le *DmManager* et le *DmAbstractInstrument*, qui sont responsables de la gestion de toutes les opérations : notifier les composants des différents évènements qui ont lieu durant l'opération, et éventuellement afficher des retours visuels – variant en fonction de la technique utilisée. Le *DmManager* est visible pour les composants qui supportent les opérations de type *drag-and-drop*. Il gère l'enregistrement des

composants en tant que source et/ou cible pour les opérations de type *drag-and-drop*. D'un autre côté, le *DmAbstractInstrument* gère le comportement de l'instrument à proprement parler.

Comme nous avons pu le constater dans la première section de ce chapitre, parmi les systèmes de fenêtrage existant, MacOS est le seul à fournir une telle entité, qui est nommée *DragManager* dans la *toolkit* Carbon.

Les entités *DmManager* et *DmAbstractInstrument* assurent un comportement cohérent entre les différentes applications qui utilisent les techniques de type *drag-and-drop* : par exemple, il est très perturbant d'avoir deux applications qui réalisent des actions différentes (copie, déplacement, lien) lorsque la même touche spéciale est activée (*ctrl*, *alt*, *shift*). Un autre exemple de comportement incohérent est l'utilisation des raccourcis clavier différents pour une même opération – les opérations de couper / copier / coller sont généralement associées aux raccourcis *ctrl+x*, *ctrl+c* et *ctrl+v* mais on trouve encore des applications qui utilisent les raccourcis *shift+delete*, *ctrl+insert* et *shift+insert*.

Le comportement complet du *drag-and-drop* repose donc sur le *DmAbstractInstrument*, ce qui permet une grande modularité. Pour passer d'une technique (par exemple le *drag-and-throw*) à une autre (le *push-and-throw*), seul l'instrument doit être changé. Les composants – sources ou cibles – n'ont pas besoin d'être conscient de ce changement. Quelque soit la technique utilisée, les composants sources et cibles agissent de la même manière et sont notifiés des mêmes évènements.

III.4.2 Modèle d'architecture

Le modèle que nous proposons a été travaillé pour être aussi simple que possible. Comme on peut le constater sur le diagramme de classes de la figure III.10, deux interfaces ont été introduites. Ces deux interfaces doivent être implémentées par les composants sources (*DmSourceListener*) et cibles (*DmTargetListener*). Nous avons également introduits deux types d'évènements et une classe *DmManager* qui gère le processus d'enregistrement des composants sources et cibles. Nous avons finalement défini une classe abstraite *DmAbstractInstrument* dont doivent hériter tous les instruments. Les classes héritant de *DmAbstractInstrument* contiennent le comportement d'un instrument. Le préfixe *dm* des noms utilisés fait référence au modèle de **m**anipulation **d**irecte dont le *drag-and-drop* est certainement la réalisation la plus emblématique.

La figure III.11 présente un diagramme de séquence UML d'une opération de type *drag-and-drop* réussie. Ce diagramme met en jeu un composant source et un composant cible qui peuvent être n'importe quel composant visible capable de recevoir l'ensemble des évènements issus d'un dispositif de pointage.

DmSourceListener

L'interface *DmSourceListener* inclut toutes les méthodes nécessaires à un composant source pour gérer une opération de type *drag-and-drop*.

Le geste d'activation – qui marque le début d'une opération de type *drag-and-drop* – doit nécessairement être détecté par l'instrument. En effet, ce geste peut ne pas être le même pour toutes les techniques (e.g. un déplacement de quelques pixels au début d'un *drag-and-drop*, un clic pour le *pick-and-drop*, etc.). Le fait que l'instrument détecte le geste d'activation

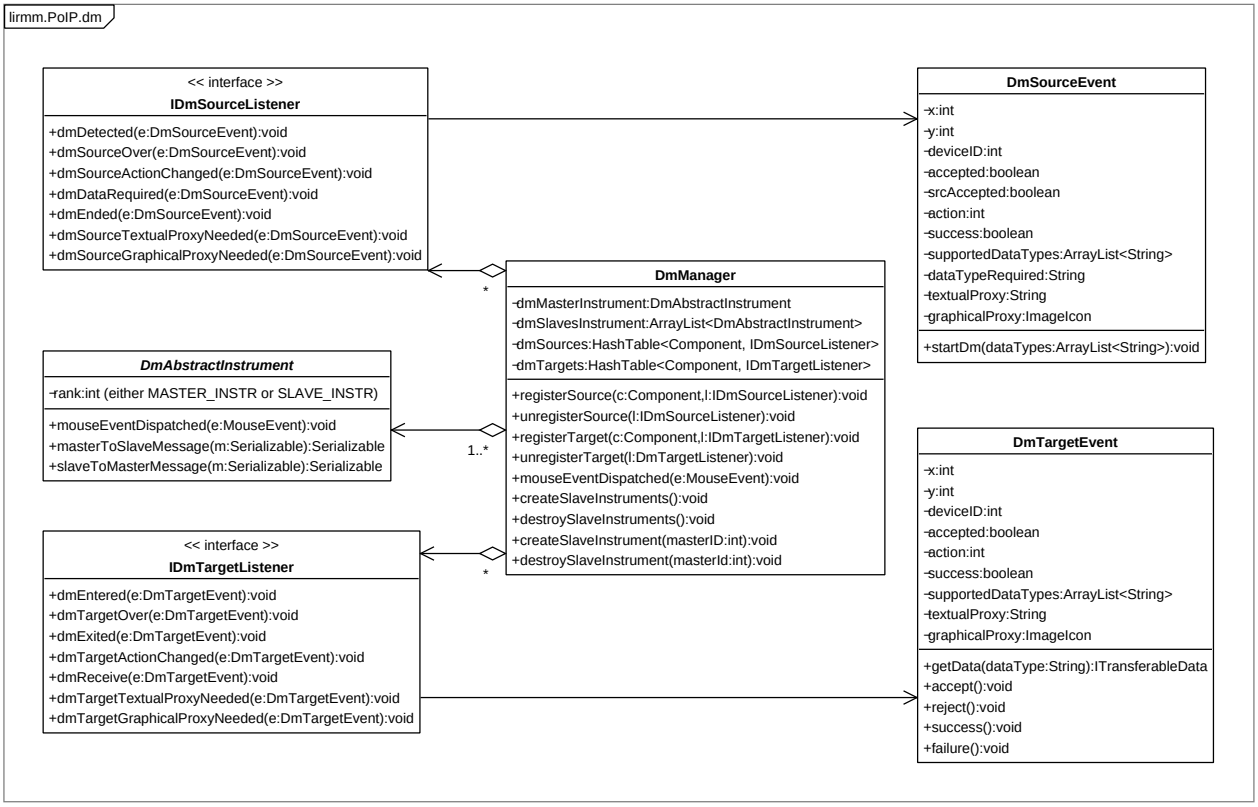

Fig. III.10: *Diagramme de classe UML.*

est également une assurance de cohérence : cela évite qu'une application déclenche un *drag-and-drop* après un déplacement de 4 pixels alors qu'une autre le fait après 8 pixels. Lorsque que ce geste d'activation est détecté, le composant source est notifié par l'appel de la méthode *dmDetected()* et doit appeler la méthode *startDm()* de l'évènement reçu en spécifiant les types de données supportés – c'est-à-dire les types de données que le composant source est capable de fournir.

Pendant le déroulement de l'opération, le composant source est notifié par les méthodes *dmSourceOver()* et *dmSourceActionChanged()* lorsque le pointeur se déplace ou que les touches spéciales (*ctrl*, *alt*, *shift*) sont activées ou désactivées.

Lorsque l'objet est déposé, les données à proprement parler sont demandées au composant source au travers de la méthode *dmDataRequired()*. Le type de données demandé par la cible est reçu en paramètre de la méthode (fait partie de l'évènement) et les données doivent être transmises en utilisant la méthode *setData()* de l'évènement reçu. Une fois que les données ont été transmises au composant cible, la source est notifiée de la fin de l'opération par la méthode *dmEnded()*. Ainsi, la source peut savoir si les données ont été transférées et traitées correctement et peut ainsi procéder aux mises à jour nécessaires (suppression de l'objet manipulé dans le cas d'un déplacement par exemple).

Il faut noter qu'il est très important que le méthode *dmDetected()* retourne un ou plusieurs types de données et que les données en elles-mêmes ne soient transférées que plus tard par la méthode *dmDataRequired()*. En effet, générer les données peut être un processus long et il

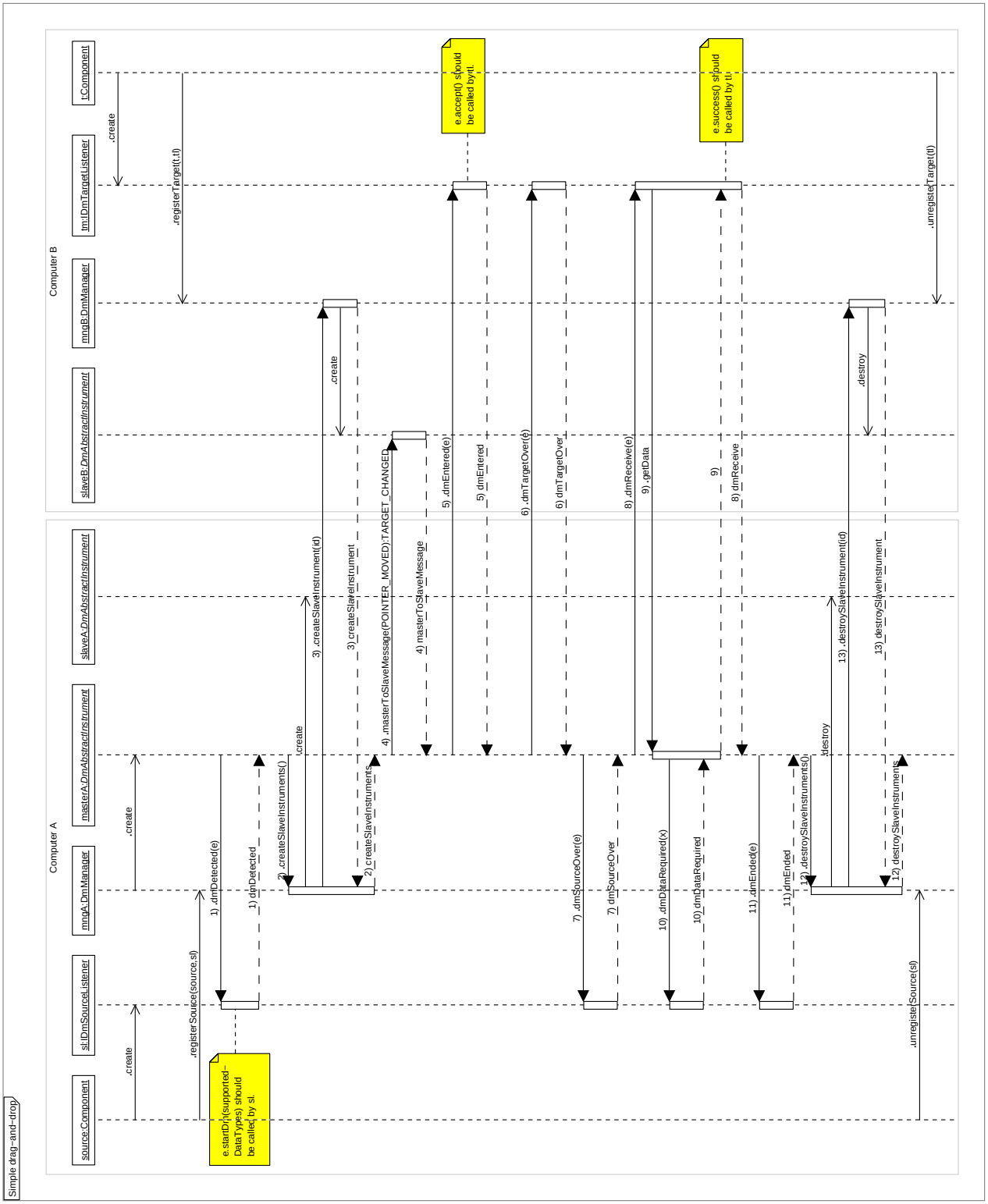

Fig. III.11: *Diagramme de séquence UML.*

peut s'avérer gênant d'avoir un délai au début de l'opération. De plus, pourquoi générer et transférer les données dans plusieurs formats alors que très probablement un seul sera utilisé par le composant cible ?

Finalement, un composant source pour une opération de type *drag-and-drop* doit être en mesure de fournir une représentation graphique et textuelle de l'objet manipulé. Ce type de représentation est par exemple utilisé pour le fantôme qui est affiché sous le pointeur dans un *drag-and-drop*.

DmTargetListener

L'interface *DmTargetListener* doit être implémentée par un composant qu'il puisse devenir une cible potentielle pour les opérations de type *drag-and-drop*.

Lorsqu'un objet est déplacé au dessus d'un composant cible, le composant est notifié par l'appel de la méthode *dmEntered()*. La cible doit alors appeler la méthode *accept()* ou *reject()* de l'évènement reçu en fonction de sa capacité à traiter les formats de données que peut produire le composant source (obtenus grâce à la méthode *getSupportedDataTypes()* de l'évènement). Pendant que l'objet est déplacé au dessus de la cible, le composant est notifié par la méthode *dmTargetOver()*, ce qui lui permet d'afficher des *feedbacks* de type *drag under*. Lorsque l'objet quitte le composant cible, la méthode *dmExited()* est appelée. Si l'utilisateur modifie le mode de transfert (en activant ou désactivant *ctrl*, *alt* ou *shift*), le composant cible est notifié par la méthode *dmTargetActionChanged()*. Il faut noter que le composant est notifié de cet évènement avant le composant source.

A la fin de l'opération, la méthode *dmReceive()* du composant cible est appelée. Celui-ci est en mesure d'obtenir la liste des formats que peut produire le composant source par un appel à la méthode *getSuporttedDataTypes()* de l'évènement et peut donc demander les données à proprement parler en appelant la méthode *getData()* de l'évènement en spécifiant le type de données requis. Une fois les données reçues, une des méthodes *success()* ou *failure()* de l'évènement doit être appelée. Ainsi, le système et le composant source sauront si les données ont été correctement reçues et traitées par le composant cible et agir en conséquence.

De la même manière que pour les composants sources, les cibles doivent être en mesure de fournir une représentation graphique et textuelle d'elles-mêmes. Ces représentations peuvent être utilisées par exemple pour l'affichage des cibles temporaires du *drag-and-pop*.

DmManager

Le *DmManager* a pour objectif la gestion commune à toutes les techniques de type *drag-and-drop*. Cela inclut les processus d'enregistrement des composants sources et cibles ainsi que la création des instruments (voir ci-dessous).

DmAbstractInstrument

La classe abstraite *DmAbstractInstrument* est la classe dont doivent hériter tous les instruments qui implémentent le comportement d'une technique de type *drag-and-drop*.

C'est l'interface entre l'utilisateur et les objets du domaine. Elle gère les *actions* de l'utilisateur auxquelles elle répond avec des *réactions* et des *feedbacks*. Pour ce faire, elle utilise un ensemble de *commandes* et de *réponses* pour communiquer avec les objets du domaine, les composants sources et les composants cibles. Cet ensemble de méthodes est défini par les

interfaces *DmSourceListener* et *DmTargetListener* et sont supposées rester inchangées. C'est la contrepartie de la modularité : les interfaces doivent être standards et donc fixées *a priori*.

Un instrument est toujours lié à un *DmManager* et connaît ainsi tous les composants sources et cibles enregistrés. On distingue deux types d'instruments : les instruments *maîtres* et les instruments *esclaves*. Un *DmManager* possède un et un seul instrument maître et un nombre variable d'instruments esclaves. Les instruments esclaves ont principalement été introduits pour le support des environnements distribués mais sont également utilisés dans les environnements mono machine par souci de cohérence. Les instruments esclaves sont habituellement utilisés pour trouver le composant se trouvant à une position donnée ou pour l'affichage de *feedbacks* (figure III.12).

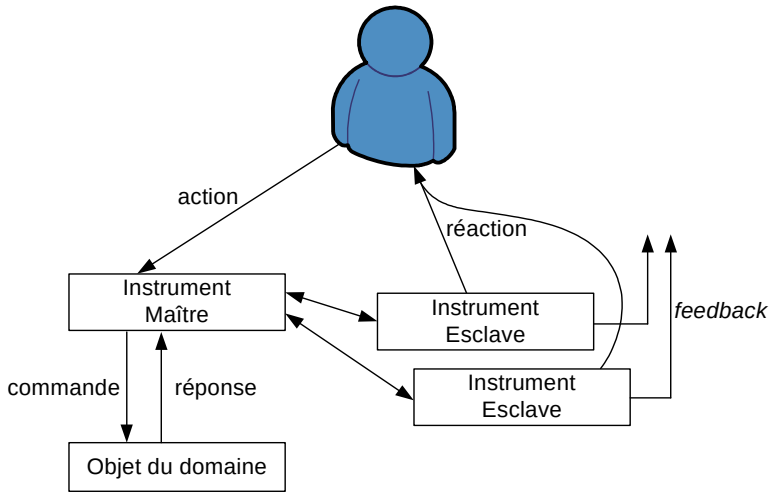

Fig. III.12: *Evolution de la figure III.9 avec un instrument fonctionnant de manière distribuée.*

Un instrument maître demande la création des instruments esclaves lorsqu'une opération de type *drag-and-drop* est détectée et demande leur destruction à la fin de l'opération. Un instrument maître est donc lié à n instruments esclaves pendant la durée d'une opération de type *drag-and-drop*, n étant le nombre de surfaces d'affichage accessibles – une surface d'affichage accessible possède un *DmManager*.

Supposons par exemple que deux machines (A et B) possèdent chacune une surface partagée et qu'un instrument de *drag-and-drop* est utilisé sur la machine A tandis qu'un instrument de *push-and-throw* est utilisé sur la machine B. Alors, si l'utilisateur de A commence un *drag-and-drop* à partir de A, le *DmManager* de A, qui possède déjà un instrument maître de *drag-and-drop*, va créer un instrument esclave de *drag-and-drop*. Le *DmManager* de B va également créer un instrument esclave de *drag-and-drop* pour A. Supposons maintenant qu'au même moment, l'utilisateur de B commence un *push-and-throw*, alors les *DmManager* de A et de B vont chacun créer un instrument esclave de *push-and-throw*, sachant que le *DmManager* de B possède déjà un instrument maître de *push-and-throw*. En d'autres termes, si les utilisateurs des machines A et B effectuent une opération de type *drag-and-drop* simultanément, alors chacun des *DmManager* possédera trois instruments : un maître et deux esclaves.

Plusieurs types d'instruments peuvent être utilisés simultanément, mais un seul type d'instrument est lié à une surface donnée (ou à un dispositif de pointage). A tout moment, un

DmManager possède exactement un instrument maître et autant d'instruments esclaves qu'il y a d'opérations de type *drag-and-drop* en cours.

Un dispositif de pointage, un instrument Comme le suggère l'exemple ci-dessus, un instrument est lié à un dispositif de pointage – et à la surface qui reçoit les évènements de ce dispositif.

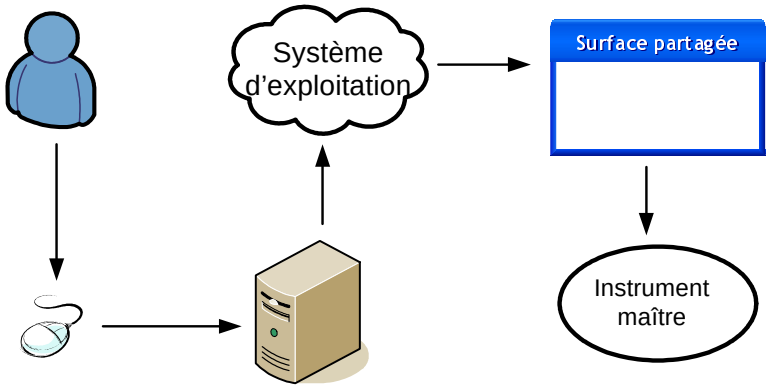

Fig. III.13: *Parcours des évènements d'entrée.*

Pour mieux comprendre la figure III.12, détaillons le cheminement des évènements d'entrée, c'est-à-dire les actions de l'utilisateur. La figure III.13 décrit le parcours de ces évènements d'entrée :

1. l'utilisateur interagit physiquement avec un dispositif de pointage,
2. la machine reçoit les informations du dispositif de pointage,
3. le système d'exploitation reçoit les informations du dispositif de pointage,
4. la surface partagée (*SharedSurface*) reçoit les évènements systèmes,
5. l'instrument maître reçoit les évènements augmentés (identification du dispositif de pointage).

On se rend compte qu'avec ce système, si plusieurs dispositifs de pointage sont utilisés sur la même machine, leurs flux d'évènements sont mélangés au niveau du système d'exploitation et la surface partagée ne reçoit qu'un seul flux d'évènements. Le support du multiplexage au niveau du système d'exploitation dépasse le cadre de notre travail. Nous avons implémenté le multiplexage des entrées à un niveau supérieur : une surface partagée ne peut gérer qu'un seul dispositif de pointage, celui-ci pouvant interagir avec l'ensemble des surfaces partagées. Un dispositif de pointage possède le même identifiant que la surface partagée qui contrôle – i.e. la surface qui reçoit, augmente et redirige les évènements issus ce dispositif de pointage.

La conséquence de ceci est qu'un dispositif de pointage est lié à un instrument. Ainsi, un utilisateur qui interagit avec un dispositif de pointage donné utilisera le même instrument – e.g. *push-and-pop* – quelle que soit la surface source de l'opération. L'instrument utilisé dépend de la surface contrôlant un pointeur et non de la surface sur laquelle se trouve ce pointeur.

III.4.3 Support des environnements distribués

L'implémentation d'une technique d'interaction comme le *drag-and-drop* dans un environnement distribué ne va pas sans poser un certain nombre de problèmes. Ces problèmes sont d'autant plus gênants lorsque l'interaction prend place de manière dynamique. Dans ce contexte, où un utilisateur déplace un objet depuis une surface vers une autre surface qu'il sélectionne « à la volée », plusieurs problèmes bien connus interviennent : (1) le problème de partage spontané de périphériques [Hinckley et al., 2004], (2) l'incompatibilité des implémentations des systèmes de fenêtrage et (3) le transfert des données.

Les implémentations actuelles du *drag-and-drop* ne permettent pas le support des environnements d'affichage distribués. Dans l'état de l'art, le *pick-and-drop*, l'*hyperdragging* et le *stitching* sont conçus et implémentés pour les interactions inter-machines. Cependant, leur objectif n'est pas de fournir une implémentation générale qui répondrait aux problèmes cités ci-dessus et qui pourrait être étendue pour le support de nouvelles techniques de type *drag-and-drop*.

Le *pick-and-drop* est certainement l'exemple le plus emblématique, étant la première technique d'interaction multi machine. Néanmoins, comme pour les autres techniques qui ont suivies, son implémentation est *ad hoc* et ne peut pas être facilement étendue pour supporter les autres extensions du *drag-and-drop* décrites dans ce mémoire.

Fournir une architecture qui répondrait de manière générale aux trois problèmes ci-dessus dépasse largement nos objectifs. En particulier, de nombreuses solutions ont été proposées ces dernières années pour améliorer les transferts de données aussi bien au niveau des protocoles que des formats. Plus récemment, le besoin de lier des périphériques de manière dynamique pour partager des informations [Barralon et al., 2004], collaborer ou communiquer a déjà conduit à un certain nombre de solutions au problème de partage spontané de périphérique [Hinckley et al., 2004]. L'objectif de notre travail a été de proposer une solution au second problème, c'est à dire l'incompatibilité des implémentations dans les différents systèmes de fenêtrage. Nous avons été également amenés à concevoir une base rudimentaire permettant à plusieurs utilisateurs de travailler de manière collaborative dans un environnement d'affichage distribué. C'est-à-dire que nous avons également abordé le premier problème évoqué ci-dessus.

III.5 Conclusion

Dans ce chapitre, nous avons présenté un nouveau modèle pour le *drag-and-drop* qui repose sur le modèle d'interaction instrumentale. Ce nouveau modèle possède le grand avantage de supporter toutes les techniques de type *drag-and-drop* connues à ce jour mais également des environnements distribués.

Ce modèle permet une simplification des concepts. En effet, avec le *drag-and-drop*, qui est souvent synonyme de facilité d'utilisation pour l'utilisateur final, la facilité n'est pas toujours au rendez-vous pour les développeurs.

Ce modèle nous a également permis d'éclaircir les différentes propriétés du *drag-and-drop* et de ses évolutions. Ceci va nous permettre de présenter dans le chapitre suivant les différentes études que nous avons réalisées pour tester les performances des différentes évolutions du *drag-and-drop*.

Chapitre IV

ETUDE DES PERFORMANCES DES NOUVELLES TECHNIQUES D'INTERACTION

Sommaire

Devant le nombre grandissant d'alternatives proposées pour étendre le *drag-and-drop* (voir le chapitre II), nous nous proposons de les comparer afin de déterminer quelle est ou quelles sont les meilleurs alternatives.

Cette comparaison va se dérouler en deux temps. Dans une première étude, nous nous intéressons exclusivement aux méthodes de lancer et au *drag-and-drop* classique. Puis nous effectuerons une étude plus complète avec la plupart des techniques présentées dans le chapitre II ainsi que deux nouvelles techniques.

Précisons que ces études ont été réalisées avant la conception du modèle décrit dans le chapitre III. Ce sont ces études qui nous ont amené à implémenter des techniques de manière *ad hoc* et qui nous ont poussé à concevoir un modèle unifié. Pour pouvoir présenter ces études le plus clairement possible, il était cependant nécessaire d'éclaircir tous les mécanismes du *drag-and-drop* au préalable.

IV.1 Etude des méthodes de lancer

L'étude réalisée lors de la conception initiale du *drag-and-throw* et du *push-and-throw* était une étude préliminaire et ne permettait pas de savoir précisément laquelle de ces deux techniques était la plus efficace [Hascoët, 2003]. Les utilisateurs préfèrent-ils faire un mouvement dans la direction de la cible ou dans la direction opposée à la cible ?

IV.1.1 Implémentation des méthodes de lancer

Pour le besoin de cette étude, les méthodes de lancer décrites dans la section II.2.5 ont été implémentées dans un prototype Java simulant un nombre quelconque de bureau. Chaque bureau contenant des icones. Trois techniques ont été implémentées :

- Le *drag-and-throw* avec trajectoire elliptique,
- Le *drag-and-throw* hybride,
- Et le *push-and-throw*.

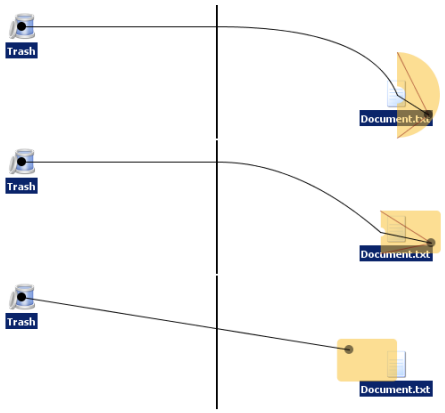

Fig. IV.1: *De Haut en bas : implémentation du (a) drag-and-throw avec trajectoire elliptique, (b) drag-and-throw hybride et (c) push-and-throw.*

drag-and-throw avec trajectoire elliptique

Il s'agit de la trajectoire décrite dans [Hascoët, 2003]. Cette trajectoire, en « cassant » la trajectoire naïve issue de la métaphore du tir à l'arc (figure IV.1 – a, voir aussi la figure II.4 – a et b, p. 44), permet de limiter l'effet de levier qui rend la trajectoire naïve pratiquement inutilisable.

En effet, en utilisant la trajectoire naïve, même si l'utilisateur conserve son pointeur dans la zone de décollage, il est possible que sa mire (emplacement où sera lancé l'objet si l'utilisateur valide l'opération) sorte de l'affichage cible. En d'autres termes, le demi-cercle représentant la zone de décollage ne projette pas parfaitement sur l'affichage cible.

drag-and-throw hybride

Il est difficile de résoudre le problème de la projection d'une zone semi-circulaire sur une zone rectangulaire évoqué précédemment. Ce problème a également des répercussions au niveau de la précision. Il est en effet important que, quelque soit la position du pointeur de l'utilisateur dans la zone de décollage, un mouvement de distance d implique toujours un mouvement de distance d' proportionnelle à d de la mire. Or le *drag-and-throw* à trajectoire elliptique n'a pas cette propriété. En effet, dans l'exemple de la figure IV.1-a, la moitié intérieure de la zone de décollage (moitié du rayon de la zone de décollage) se projette sur la moitié droite de l'affichage destination, et la moitié extérieure de la zone de décollage se projette sur la moitié gauche de l'affichage destination. Or ces les moitiés intérieure et extérieure de la zone de décollage n'ont pas des surfaces équivalentes alors que les deux moitiés de l'affichage destination ont la même surface.

Afin de résoudre ces problèmes, nous avons proposé le *drag-and-throw* hybride qui dispose d'une zone de décollage rectangulaire (figure IV.1-b). Cette zone de décollage peut être facilement projetée sur un affichage rectangulaire. De plus, le problème de différence de précision disparaît également. Les fondements du *drag-and-throw* sont tout de même conservés puisque l'utilisateur effectue un mouvement dans la direction inverse à la cible du déplacement en cours. Dans l'exemple de la figure IV.1-b, le coin inférieur droit de la zone de décollage est projeté sur le coin supérieur gauche de l'affichage cible.

push-and-throw

L'implémentation du *push-and-throw* est beaucoup moins problématique. La zone de décollage, qui est en fait une sorte de « vue radar » ou de « monde en miniature », est rectangulaire (figure IV.1-c) et se projette de manière triviale sur l'affichage cible. La trajectoire, quant à elle, est une simple ligne droite liant le pointeur de l'utilisateur et la mire.

IV.1.2 Etude

Une étude a donc été réalisée pour comparer les performances du *drag-and-drop*, du *drag-and-throw* et du *push-and-throw*. Etant donné que notre modèle d'interaction s'applique aux déplacements d'objets, nous avons suivi une méthodologie similaire à la méthodologie standard utilisée pour l'évaluation des performances utilisateur dans les tâches de pointage multidirectionnelles [Douglas et al., 1999].

Douze participants bénévoles ont été choisis pour participer à cette étude. Onze d'entre eux étaient droitiers et tous manipulaient régulièrement les outils informatiques et le *drag-and-drop* mais aucun n'avait d'expérience avec les techniques de lancer. Ceci constitue bien évidemment un avantage pour le *drag-and-drop* mais il est pratiquement impossible d'éviter ce genre de biais étant donné que les participants sont familiers avec les environnements informatiques.

Les tests pilotes de l'étude nous ont fait choisir la version hybride du *drag-and-throw* au détriment de la version avec trajectoire elliptique.

Nous avons utilisé un plan d'expérience avec répétition : les participants ont tous testé les trois techniques. Cependant, les techniques ont été présentées aux utilisateurs dans des ordres différents. Ces ordres étant équilibrés grâce à un carré latin. L'étude a été conçue pour que toutes les techniques soient présentées un même nombre de fois à une position donnée. Ceci assure également qu'une technique est toujours précédée un même nombre de fois d'une autre technique. De plus, pour réduire le bruit dû aux différences entre les modèles de lancer (le geste du *drag-and-throw* est inversé par rapport à celui du *push-and-throw*), une période d'au minimum 24 heures séparait chaque test.

Variables indépendantes

Les variables indépendantes suivantes ont été considérées :

- La technique d'interaction utilisée : *drag-and-drop* (dd), *drag-and-throw* hybride (hdt), et *push-and-throw* (pt)
- La taille de la cible : 32, 48 ou 64 pixels
- La position de la cible : 12 positions distribuées uniformément sur une grille (4×3)
- L'essai (1 à 36)
- Le bloc (1, 2, 3, 4)
- Le test (1, 2, 3)

un bloc est un ensemble de 36 essais qui correspondent à toutes les combinaisons possibles de tailles et de positions des cibles pour une technique d'interaction donnée. Un test est formé de 4 blocs pour une technique d'interaction. Chaque participant a réalisé 3 tests, un pour chaque technique d'interaction.

Une autre variable dépendante dépend de deux des variables citées ci-dessus : L'*indice de difficulté* [Douglas et al., 1999] est une fonction dépendant de la largeur et de la distance de la cible (donc de sa position). L'indice de difficulté est issu de la loi de Fitts et est défini par :

$$ID = \log_2 \left(\frac{D}{W} + 1 \right)$$

où D est la distance à la cible et W la largeur de la cible (direction du mouvement). La combinaison de trois largeurs et de 12 positions implique donc 36 indices de difficulté différents.

Variables dépendantes

Trois variables dépendantes définies par [Douglas et al., 1999] ont été utilisées pour évaluer les performances en terme de vitesse et de taux d'erreur des déplacements d'objets réalisés par les participants :

- Le temps de déplacement est le temps nécessaire pour déplacer un objet depuis l'affichage source vers une cible donnée et pour ramener le pointeur dans l'affichage source.

- Le taux d'erreur est le pourcentage d'erreurs (la cible n'est pas atteinte lorsque l'objet est déposé, c'est à dire lorsque le bouton de la souris est relâché).
- Le *throughput* est une mesure du standard ISO 9241 [Douglas et al., 1999] dans laquelle le temps et la précision (erreur par rapport au centre de la cible) entrent en compte.

Deux autres variables dépendantes ont été utilisées afin d'obtenir des indications quant à la progression des performances des participants :

- Le taux de progression des temps de déplacement (PRTD) est calculé de la manière suivante :

$$PRTD = \frac{(TDM_i - TDM_f)}{TDM_i}$$

où TDM_i est le temps moyen de déplacement dans le premier bloc et où TDM_f est le temps moyen de déplacement dans le dernier bloc.

- Le taux de progression du *throughput* (PRTP) est calculé de manière similaire :

$$PRTP = \frac{(TPM_f - TPM_i)}{TPM_i}$$

où TPM_i est le *throughput* moyen dans le premier bloc et où TPM_f est le *throughput* moyen dans le dernier bloc.

Procédure expérimentale

Un Pentium® IV 1,5GHz disposant de 512Mo de RAM et d'une carte graphique Nvidia® GeForce2 à double sortie vidéo a été utilisé en complément de deux écrans cathodiques de 17" réglés à une résolution de 1024×768 et d'une souris Microsoft® *basic optical mouse*.

Les tâches à réaliser par les participants étaient présentées par un programme Java ouvrant deux fenêtres, chaque fenêtre simulant un bureau contenant des icones. Dans un premier temps, les participants disposaient de deux bureaux contenant un certain nombre d'icones et pouvaient effectuer tous les mouvements qu'ils désiraient afin de découvrir la technique d'interaction. Une fois l'utilisateur prêt, les tests chronométrés impliquaient de déplacer une icone à partir du bureau de gauche vers le bureau de droite. La position de l'icone à déplacer était toujours centrée sur le bureau de gauche tandis que la position de la cible sur le bureau de gauche changeait à chaque essai.

Chaque participant a accompli quatre blocs de 36 essais (combinaisons de 12 positions et de 3 tailles de cibles) pour chaque technique d'interaction. Chaque sujet a donc réalisé 144 essais pour chaque technique soit un total de 432 essais. Les participants ont reçu pour consigne de déplacer les objets comme ils le feraient dans une utilisation quotidienne (un compromis entre rapidité et précision).

Un carré latin a été utilisé pour faire varier l'ordre dans lequel les différentes techniques d'interaction étaient testées par les utilisateurs.

Résultats

Temps de déplacement Comme le montre la figure IV.2, les participants ont été significativement plus rapides pour déplacer les objets avec le *push-and-throw* qu'avec le *drag-and-drop* ($p < 10^{-15}$). Le *push-and-throw* a présenté un temps moyen de déplacement inférieur de 16% à celui du *drag-and-drop*.

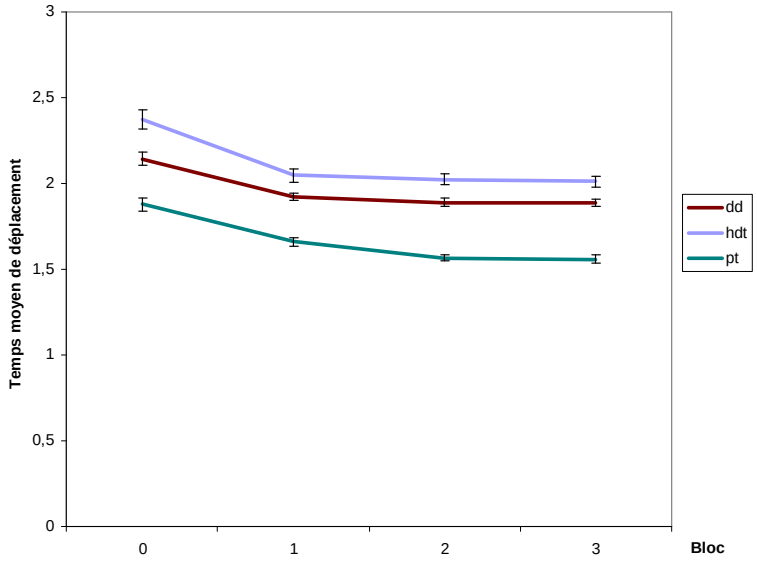

Fig. IV.2: *Temps de déplacement en fonction du bloc.*

A l'inverse, le *drag-and-throw* s'est révélé plus lent que le *drag-and-drop* mais avec une perte de seulement 7%. Il semble que le *drag-and-throw* nécessite plus de pratique pour obtenir de meilleures performances que le *push-and-throw*. Il semble également que les performances du *drag-and-throw* soient plus sujettes aux préférences de chacun. En effet, seulement trois des douze participants ont été à l'aise avec son style de gestes inversés et l'ont préféré au *push-and-throw*.

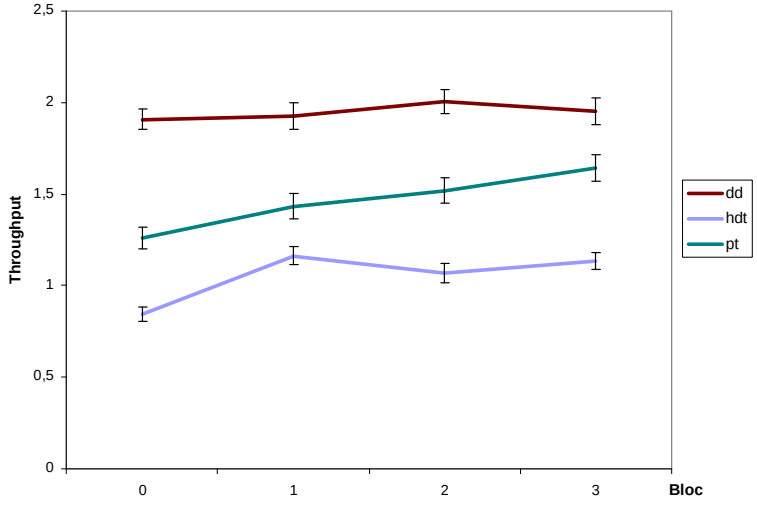

Fig. IV.3: *Throughtput en fonction du bloc.*

Throughput Rappelons que le *throughput* est un indicateur qui renseigne à la fois sur la vitesse et la précision d'une tâche de pointage [Douglas et al., 1999], qu'il se mesure en *bits* et

qu'une valeur de *throughput* plus élevée correspond à une vitesse et une précision plus grande.

Les résultats obtenus pour le *throughput* du *drag-and-drop*, du *drag-and-throw* et du *push-and-throw* sont significativement différents ($p < 10^{-15}$). Les meilleures mesures de *throughput* ont été obtenues pour le *drag-and-drop*. Cependant, comme le montre la figure IV.3, et comme le confirmera l'étude des taux de progression, le *drag-and-drop* obtient également le taux de progression le plus faible. Il est donc raisonnable de penser qu'avec plus de pratique, les utilisateurs obtiendront de meilleurs résultats au niveau du *throughput* avec le *drag-and-throw* et le *push-and-throw* qu'avec le *drag-and-drop*.

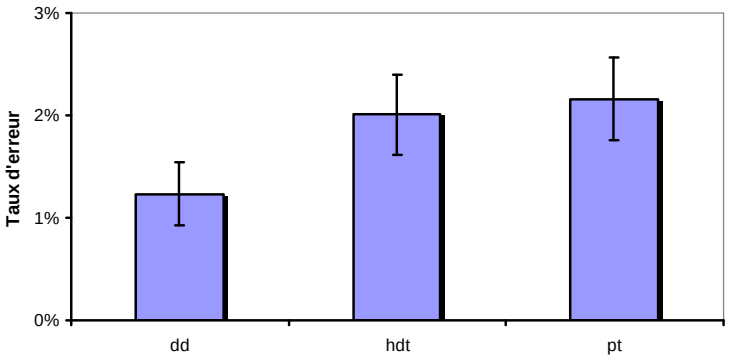

Fig. IV.4: *Taux d'erreur de chaque technique d'interaction.*

Taux d'erreur Les taux d'erreur moyens ont été légèrement plus importants pour les techniques de lancer que pour le *drag-and-drop*. Mais les résultats n'était pas significativement différents ($p = 0,16$). Cependant, il est très intéressant de remarquer que tous les taux d'erreur sont inférieurs à 3% (figure IV.4). Les taux d'erreurs obtenus sont largement meilleurs que ceux des autres techniques de lancer [Geißler, 1998].

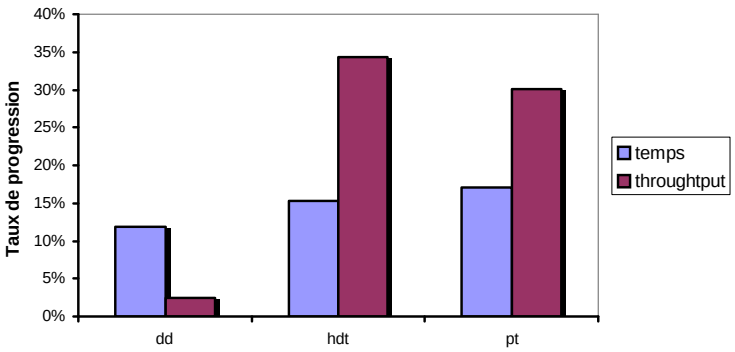

Fig. IV.5: *Taux de progression de chaque technique d'interaction.*

Taux de progression La comparaison des taux de progression au niveau des temps de déplacement et du *throughput* a montré des différences significatives ($p < 10^{-15}$). Comme on peut le constater sur la figure IV.5, les taux de progression sont faibles pour le *drag-and-drop*. Ces résultats reflètent le fait que les participants maîtrisaient déjà la technique du *drag-and-drop* et qu'ils se sont familiarisés avec l'exercice qui leur était demandé. Les meilleurs taux de progression ont été obtenus par le *drag-and-throw* pour le *throughput* et par le *push-and-throw* pour les temps de déplacement.

Il est intéressant de constater que, tandis que le *push-and-throw* enregistrait une progression au niveau des temps de déplacement, ce résultat était accompagné d'une augmentation du taux d'erreur. C'est pour cette raison que le *push-and-throw* n'a pas obtenu le meilleur taux de progression au niveau du *throughput*.

Regressions de Fitts La figure IV.6 présente les droites de régression de Fitts sur les résultats de cette expérience. La liaison entre les temps de déplacement et les indices de difficulté est très significative ($p < 10^{-7}$ dans tous les cas).

On retrouve des résultats proches de ceux observés jusqu'ici. En effet, L'influence de l'indice de difficulté d'une tâche sur le temps de déplacement est comparable pour le *drag-and-drop* et le *push-and-throw* tandis qu'elle est plus importante pour le *drag-and-throw* hybride.

Préférences des participants Un questionnaire a été confié aux participants à l'issue de l'étude. Les résultats de ce questionnaire ont montré que la majorité des participants (75%) ont préféré utiliser le *push-and-throw* que le *drag-and-throw* et le *drag-and-drop*. La plupart des participants ont été déconcerté par l'utilisation du *drag-and-throw* par rapport au *push-and-throw*. Finalement, la majorité des participants a considéré que les méthodes de lancer étaient plus adaptées à la tâche qui leur était demandée que le *drag-and-drop* et ils ont eu le sentiment qu'ils pouvaient obtenir suffisamment de précision avec les méthodes de lancer.

Conclusion de l'étude

Les méthodes de lancer demandent une certaine pratique. En particulier, pour le *drag-and-throw*, le fait que le mouvement de la main (du pointeur) et le mouvement de mire soient opposés semble introduire une surcharge cognitive qui ralentit l'opération mais qui diminue cependant avec la pratique. Dans l'ensemble, le *drag-and-throw* et le *push-and-throw* ont révélé de bonnes performances aussi bien au niveau des temps de mouvement qu'au niveau du *throughput* et des taux d'erreur. Les taux d'erreur étaient en effet bien meilleurs que ceux qui avaient pu être obtenus jusqu'ici par d'autres méthodes de lancer [Geißler, 1998]. La comparaison avec le *drag-and-drop* est également encourageante du fait des temps de déplacement du *push-and-throw* et des taux de progression.

De plus, il est important de préciser que le *drag-and-drop* a disposé d'au minimum deux avantages dans cette étude. Premièrement, tous les participants étaient familiers avec le *drag-and-drop*. Deuxièmement, il a bénéficié de l'accélération du pointeur mise en place par le système de fenêtrage et qui est une fonctionnalité très utile pour la tâche qui était demandée aux participants, à savoir parcourir de grandes distances en *drag-and-drop*. Or, cette accélération n'est pas possible avec les systèmes de pointage direct (surfaces augmentées) auxquels se destinent les méthodes de lancer présentées ici.

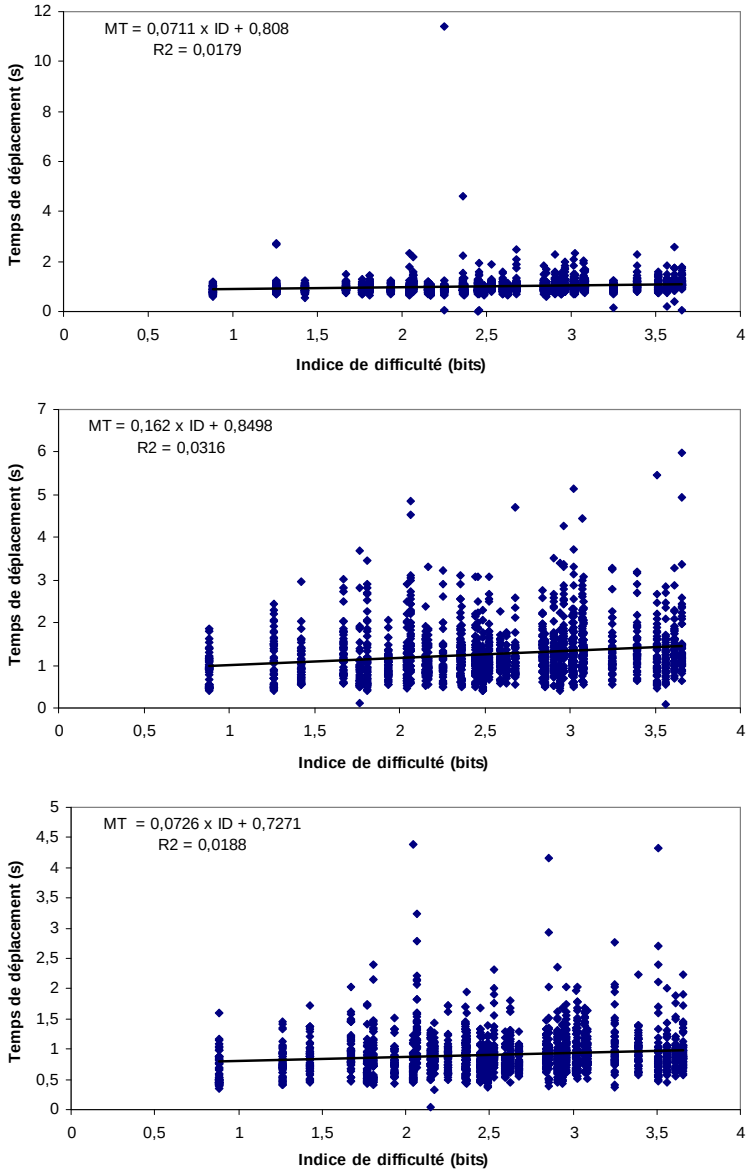

Fig. IV.6: *De haut en bas : Régression de Fitts pour (a) le drag-and-drop, (b) le drag-and-throw hybride et (c) le push-and-throw.*

Les conditions expérimentales étaient donc clairement en faveur du *drag-and-drop* et, par conséquent, il convient de préciser que les résultats obtenus ici ne sont pas seulement encourageants mais présentent les performances minimales pour nos méthodes de lancer.

IV.2 Etude complète

L'étude présentée précédemment, nous a permis de comparer les techniques de lancer au *drag-and-drop*. L'étude a été réalisée dans des conditions qui favorisaient clairement le *drag-and-drop*. En effet, les méthodes de lancer et les évolutions du *drag-and-drop* en général ne se destinent pas à un environnement matériel classique (utilisation de la souris en particulier) mais plutôt à des dispositifs de pointage direct (type stylo).

D'autre part, il semble intéressant de comparer les performances des méthodes de lancer avec les autres évolutions proposées pour le *drag-and-drop* [Baudisch et al., 2003; Rekimoto, 1997]

Le besoin s'est donc fait sentir de réaliser une étude dans des conditions plus réalistes, c'est-à-dire dans un environnement pour lequel les méthodes de lancer ont été imaginées : les surfaces augmentées.

Pour cette étude, nous avons bénéficié de la collaboration de Patrick Baudisch (Microsoft Research), qui est à l'origine du *drag-and-pop* [Baudisch et al., 2003] et de Brian Lee (Stanford University) du groupe *interactive workspace* qui a mis en place l'expérimentation dans le cadre du *iWall* [Johanson et al., 2002a].

IV.2.1 Candidats

Les techniques candidates pour l'étude étaient :

- Le *drag-and-drop*,
- Le *pick-and-drop*,
- Le *drag-and-pop*,
- Le *push-and-throw*,
- Et le *drag-and-throw*.

Toutes ces techniques ont été implémentées dans un prototype écrit en Delphi et reposant sur la bibliothèque graphique GDI+ [Network]. Ce prototype simule un bureau contenant un ensemble d'icones qui peuvent être manipulées de la même manière que sur un bureau réel. Les évènements utilisés sont des évènements souris basiques. Ce prototype peut être utilisé avec des systèmes de pointage du type Smart BOARD™ [SMART Technologies] ou Mimio™ [Virtual Ink] dans la mesure où ces systèmes génèrent des évènements souris classiques.

Certaines techniques évoquées dans l'état de l'art (voir le chapitre II) ne font pas partie de cette étude. Dans un premier temps, l'*hyperdragging* a été écarté car c'est une technique destinée à être utilisée avec des dispositifs de pointage indirect, ce qui est en contradiction avec nos conditions expérimentales : les tests vont se dérouler sur des surfaces augmentées et donc avec des dispositifs de pointages directs. D'autre part, le *stitching* n'a pas été inclus dans cette étude car cette technique n'était pas encore publiée au moment de la conception de cette étude.

Durant la préparation de cette expérience, nous avons été amenés à modifier/améliorer certaines techniques et à en introduire une nouvelle : le *push-and-pop*.

Pick-and-drop

L'implémentation du *pick-and-drop* dans notre prototype est légèrement différente de l'implémentation originale de Rekimoto [Rekimoto, 1997]. En effet, l'auteur explique comment différencier le *pick-and-drop* du *drag-and-drop* mais il n'explique pas comment différencier le *pick-and-drop* de la sélection.

Notre prototype est censé simuler un bureau. Il est donc nécessaire de pouvoir sélectionner une icone du bureau par un simple clic et il nous a donc fallu trouver une manière d'implémenter le *pick* du *pick-and-drop* autrement que par un simple clic.

Deux solutions se sont offertes à nous. La première, comme le *take* de Geißler, était de déclencher un *pick* si l'utilisateur laisse sont stylo en contact (ou le bouton de sa souris enfoncé) pendant un certain temps sans bouger. La seconde solution était de déclencher un *pick* si l'utilisateur effectue un léger *drag-and-drop* sur l'objet à déplacer.

Nous avons opté pour la seconde solution. Même si celle-ci empêche la cœxistence du *pick-and-drop* et du *drag-and-drop*, elle a l'avantage de permettre un *feedback* lors du *pick*. Ce *feedback* est le même que pour un *drag-and-drop*, c'est-à-dire un objet en semi-transparence sous le pointeur de l'utilisateur, et est donc facilement assimilé par l'utilisateur.

La figure IV.7 montre un exemple d'utilisation du *pick-and-drop* dans notre prototype. Dans un premier temps ($1 \rightarrow 2$), l'utilisateur effectue un petit *drag-and-drop* sur l'icone à déplacer. Son stylo n'est alors plus en contact avec la surface (ou le bouton de sa souris n'est plus enfoncé). (3) L'utilisateur clique alors sur l'icone cible, ce qui a pour effet de terminer l'opération (4) et d'ouvrir le navigateur dans notre exemple.

Drag-and-pop

Le *drag-and-pop* a été implémenté comme décrit dans la section II.2, cependant, nous avons résolu quelques problèmes identifiés par Baudisch et al.

La première modification a été apportée au niveau de la sélection des cibles probables d'une opération afin de limiter les erreurs dues à la sélection d'un mauvais ensemble de cibles. Dans la version originale du *drag-and-pop*, les cibles étaient sélectionnées dans la direction du mouvement avec un angle centré sur le mouvement initial (figure IV.8-gauche). Dans la version du *drag-and-pop* que nous avons implémentée, les angles de sélection des cibles ne sont pas les mêmes de chaque côté du mouvement initial.

Ceci est important, car, que ce soit en utilisant une souris ou un stylo sur une surface augmentée, le mouvement d'un utilisateur pour aller d'un point A à un point B est rarement la ligne droite. En effet, alors que la main et le poignet de l'utilisateur se déplacent, son coude et son épaule sont beaucoup moins mobiles. Son mouvement est donc souvent plus proche de l'arc de cercle que de la ligne droite.

Deuxièmement, dans sa version originale, le *drag-and-pop* positionnait parfois les icones fantômes à cheval sur deux surfaces d'affichage. Dans notre version, nous nous sommes donc assurés que les icones fantômes soient toujours positionnées sur le même affichage que le pointeur.

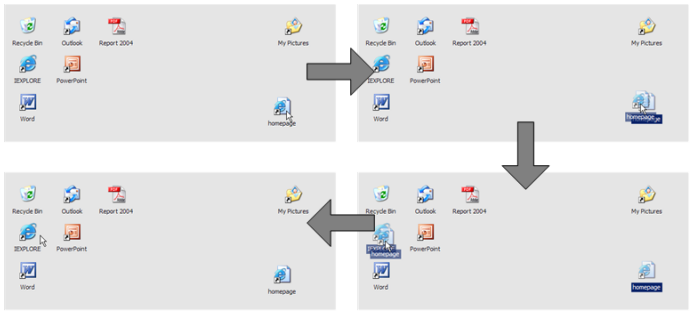

Fig. IV.7: *Exemple d'utilisation du pick-and-drop.*

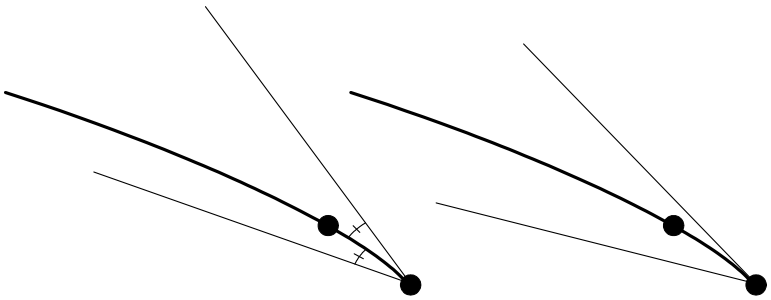

Fig. IV.8: *Angles de sélection des cibles probables pour le drag-and-pop. A gauche : dans la version originale du drag-and-pop. A droite : dans la version implémentée dans notre prototype.*

Push-and-throw et drag-and-throw

Le *drag-and-throw* et le *push-and-throw hybride* ont été implémentés d'une manière très similaire à ce qui a été fait pour l'étude présentée dans la section précédente. On peut cependant noter deux différences. La première se situe au niveau de la trajectoire. Alors qu'elle était représentée par une ligne dans l'implémentation précédente, elle est ici représentée par une bande élastique (figure IV.9).

L'utilisation d'une bande élastique en guise de trajectoire présente l'avantage de permettre à l'utilisateur de se faire une idée de la longueur de la trajectoire en ne voyant qu'une partie de celle-ci. Ce qui peut être intéressant sur une surface d'affichage de grande taille.

Notons également que la mire, qui était un disque noir dans la version précédente, est maintenant représentée par une copie semi-transparente de l'icone qui est en cours de manipulation. La couleur de la bande élastique dépend d'ailleurs de cette icone.

L'autre modification qu'ont dû subir le *drag-and-throw* et le *push-and-throw* est le support de déplacements pour lesquels la source et la cible se situent sur la même surface. Il est maintenant possible de déplacer un objet vers n'importe quelle position du bureau, quel que soit l'affichage concerné. Ainsi, la zone de décollage (figure IV.9) représente l'espace de travail complet et non plus seulement l'affichage cible. La conséquence de cette modification est de ne plus permettre l'utilisation des trajectoires elliptiques du *drag-and-throw*.

La figure IV.10 présente un exemple d'utilisation du *push-and-throw*. Dans un premier temps, (1 $\rightarrow$ 2) l'utilisateur pose son stylo (ou enfonce le bouton de sa souris) sur une

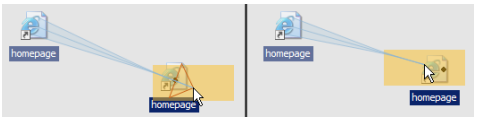

Fig. IV.9: *Les zones de décollage.*

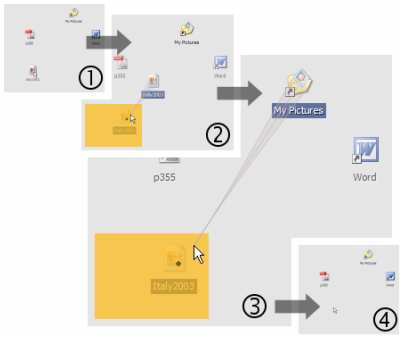

Fig. IV.10: *Un exemple d'utilisation du push-and-throw.*

icone, ce qui a pour effet de faire apparaître les *feedbacks* du *push-and-throw*. Une copie semi-transparente de l'icone (la mire) apparaît, qui est reliée au pointeur par une bande élastique. En déplaçant le stylo ou le pointeur de la souris à l'intérieur de la zone de décollage, la mire se déplace sur la position correspondante du bureau. (3) Lorsque l'utilisateur approche d'une cible potentielle, un dossier dans cet exemple, la cible adopte le comportement habituel (*drag-under feedback*), c'est à dire qu'elle apparaît en surbrillance. (4) lorsque le stylo ou le bouton de la souris est relâché, l'icone est déplacée.

Le *push-and-throw* et le *drag-and-throw* proposent finalement à l'utilisateur de faire un *drag-and-drop* dans un monde en miniature. Ceci peut introduire un manque de précision. Dans nos tests, la zone de décollage était paramétrée pour avoir une surface égale à $\frac{1}{64}$ de la surface du bureau ($\frac{1}{8}$ en hauteur, $\frac{1}{8}$ en largeur). Ce qui signifie qu'un mouvement qui correspond habituellement à 1 pixels provoque un mouvement de 8 pixels de la mire. C'est cette constatation qui nous a poussé à introduire deux nouvelles techniques : le *push-and-throw* accéléré et le *puh-and-pop*.

Push-and-throw accéléré

Le *push-and-throw* accéléré est le résultat de l'introduction de deux comportements dans le *push-and-throw* : (1) un facteur d'accélération non linéaire du pointeur et (2) un pointage sémantique [Blanch et al., 2004].

L'accélération non linéaire est comparable à ce qui se fait pour les systèmes de pointage indirect. Une fonction polynomiale de degré 2 est utilisée pour calculer le rapport entre les mouvements du pointeur et de la mire.

Le pointage sémantique a lui aussi un effet sur le facteur d'accélération en fonction des objets qui se trouvent sous la mire. C'est à dire que l'accélération de la mire est plus importante lorsque aucune cible probable ne se trouve sous la mire tandis qu'elle est diminuée lorsque la mire passe au dessus d'une cible probable.

L'accélération de la mire dépend au final de deux facteurs : d'une part de la vitesse de déplacement du pointeur de l'utilisateur et d'autre part, de la présence ou non d'une cible à proximité de la mire.

La conséquence de l'introduction de ce facteur d'accélération est qu'il n'est plus possible d'afficher la zone de décollage du *push-and-throw*. En effet, la position de la mire ne dépend plus uniquement de la position du pointeur mais aussi de la vitesse à laquelle s'est déplacé le pointeur pour atteindre sa position. D'autre part, il peut également arriver que le pointeur atteigne le bord d'une surface d'affichage et bloque ainsi le mouvement de l'utilisateur.

Pour remédier à ce problème, nous avons introduit une fonction de débrayage : dans le cas où l'utilisateur atteint les limites de son système de pointage (le bord d'un affichage disposant d'un système de pointage direct ou le bord du bureau), il peut interrompre son geste et le reprendre où il le souhaite. Pour ce faire, une zone en bordure de chaque affichage est réservée (visualisée par un *feedback* d'instrument), et si l'utilisateur lève son stylo (ou relâche le bouton de sa souris) dans cette zone, alors l'opération est mise en pause et reprendra lorsque l'utilisateur reposera son stylo (enfoncera à nouveau le bouton de sa souris). Lorsque l'opération reprend, le pointeur de l'utilisateur aura fait un bond tandis que la mire continuera son mouvement comme si de rien n'était.

Push-and-pop

La seconde solution imaginée pour remédier au manque de précision du *push-and-pop* a été un travail d'optimisation de la zone de décollage. La zone de décollage est une vue miniature de l'espace cible. L'idée initiale était de réaliser une technique qui combine les points forts du *drag-and-pop* et du *push-and-throw*. Le nom *push-and-pop* exprime le fait qu'elle est construite à partir du push-*and-throw* et du *drag*-and-pop. Le résultat est une technique disposant d'une zone de décollage, à la manière du *push-and-throw*. Cependant, cette zone de décollage n'est plus uniformément projetée sur le bureau mais dispose de copies temporaires des cibles probables dont la sélection et la disposition sont inspirées du *drag-and-pop*.

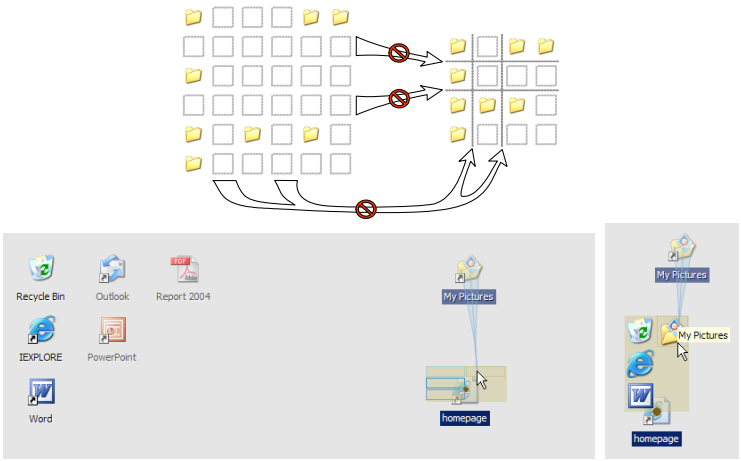

Fig. IV.11: *En haut : Arrangement des cibles [Baudisch et al., 2003]. En bas à gauche : version préliminaire du push-and-pop. En bas à droite : version finale du push-and-pop.*

La répartition de la zone de décollage entre les différentes cibles se fait de la même manière

que le positionnement des icones temporaires dans le *drag-and-pop* [Baudisch et al., 2003] (figure IV.11-haut). Cet arrangement des cibles permet de conserver les positions relatives des objets et ainsi de profiter de la mémoire spatiale de l'utilisateur. Pour cela, on place les objets cibles sur une grille, puis on supprime les colonnes et les lignes vides pour obtenir une grille compacte.

Dans un premier temps, nous avons conservé les dimensions de la zone de décollage que nous avons répartie entre les différentes cibles (figure IV.11-gauche). Cependant, en fonction de la composition du bureau, les zones réservées à chaque cible pouvaient avoir des formes très écrasées comme dans l'exemple de la figure IV.11. Nous avons donc fait en sorte que chaque cible dispose d'une zone rectangulaire dans la zone de décollage. De ce fait, la zone se décollage possède des dimensions variables qui dépendent du nombre de cibles. De plus, les cibles étant rectangulaires, nous avons reporté les icones des cibles dans la zone de décollage de la version finale du *push-and-pop* (figure IV.11-droite).

Au final, la technique du *push-and-pop* se rapproche du *pointage d'objet* [Guiard et al., 2004]. En effet, les espaces séparant les objets sur le bureau disparaissent lorsque ces objets sont représentés dans la zone de décollage ce qui permet une plus grande efficacité puisque le pointeur n'a plus à parcourir d'espace vides.

Mise à jour du tableau comparatif

Nous pouvons maintenant intégrer les deux nouvelles techniques présentées précédemment dans le tableau comparatif présenté dans la section II.3. Comme le montre le tableau IV.1, le *push-and-throw* accéléré possède des propriétés similaires au *push-and-throw* à la différence près qu'il ne dispose plus de zone de décollage mais d'une zone de débrayage en bordure de l'écran.

D'autre part, le *push-and-pop* est, quant à lui, plus proche du *drag-and-pop* à la différence qu'il ne nécessite qu'une seule réorientation de l'utilisateur. Cela signifie qu'il n'y a plus de risque qu'un mauvais ensemble de cibles soit sélectionné, comme cela pouvait être le cas avec le *drag-and-pop*.

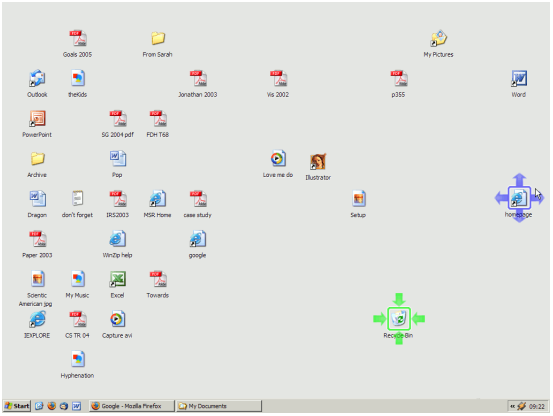

Fig. IV.12: *Disposition du bureau utilisé pour l'étude.*

Technique	**Réorientation** (action)	**Approche** (réaction)	**Retours visuels** (*Feedback*)
drag-and-drop	jamais	-	-
pick-and-drop	jamais	-	-
hyperdragging	jamais	-	trajectoire
stitching	jamais	-	bordure d'affichage
throwing	jamais	vers cible	-
drag-and-throw / push-and-throw	constamment	vers cible	zone de décollage et trajectoire
push-and-throw accéléré	constamment	vers cible	trajectoire et zone de débrayage
drag-and-pop	une fois	vers pointeur	icones temporaires et bandes élastiques
push-and-pop	une seule fois	vers pointeur	zone de décollage et icones

Tab. IV.1: *Révision du tableau comparatif des instruments.*

Hypothèses

Pour cette étude, nos hypothèses sont que toutes les techniques devraient s'avérer plus performantes que le *drag-and-drop* pour les grandes distances de déplacement. Nous nous attendons en particulier à ce que les techniques nécessitant peu ou pas de réorientation obtiennent de meilleurs résultats que les techniques nécessitant un réajustement constant.

D'autre part, nous voulons savoir si le *push-and-pop*, qui est censé combiner les avantages du *drag-and-pop* et du *push-and-throw*, obtiendra réellement de meilleurs résultats que ses deux « ancêtres ».

Les besoins logiciels

La notion de « cible probable » a été évoquée à plusieurs reprises : pour la sélection des cibles du *drag-and-pop* et du *push-and-pop* et pour l'accélération sémantique du *push-and-throw accéléré*.

Par exemple, on peut voir sur la figure IV.11-droite, que les cibles probables pour le déplacement d'une page web sont : la corbeille, le navigateur, le traitement de texte ainsi que tous les dossiers.

Notre prototype, qui simule un bureau, contient donc des informations permettant de déterminer quelles cibles sont susceptibles d'accepter l'objet que l'utilisateur souhaite déplacer. Ce genre d'information n'est pas totalement disponible dans le bureau Windows® par exemple. Le bureau de Windows ne fait aucune différence entre les applications tandis que notre prototype connaît les types de fichier que peut accepter chaque application.

Dans l'étude précédente, les participants n'avaient affaire qu'à deux icones pour chaque tâche : une icone source et une icone cible. Ceci permettait de n'avoir aucun bruit dû aux icones superflues. Cependant, pour cette nouvelle étude, les participants vont interagir avec un bureau contenant un certain nombre d'icones et cela pour plusieurs raisons : d'une part, nous

voulions tester toutes ces évolutions du *drag-and-drop* dans un environnement « réaliste » et, d'autre part, des techniques comme de *drag-and-pop* ou le *push-and-pop* n'ont que très peu d'intérêt s'il n'y a qu'une seule cible possible pour une tâche donnée. Nous avons donc utilisé l'arrangement de bureau présenté sur la figure IV.12 qui est inspiré d'un bureau réel.

Le dessin des bandes élastiques

Les bandes élastiques sont issues de la publication de Baudisch sur le *drag-and-pop* [Baudisch et al., 2003]. Nous avons eu besoin de les implémenter lorsque nous avons voulu comparer différentes techniques d'interaction dont le *drag-and-pop*.

Pourquoi les bandes élastiques L'utilisation des bandes élastiques n'est une simple question d'esthétique mais facilite la compréhension qu'a l'utilisateur de son environnement. La solution la plus simple serait de dessiner de simples lignes en lieu et place des bandes élastiques. C'est d'ailleurs ainsi que fonctionnaient les versions précédentes du *push-and-pop* et de ses variantes. Cependant, grâce à leurs formes incurvées, les bandes élastiques permettent à l'utilisateur d'avoir une appréciation de la longueur de la bande – et donc de la position de l'objet le plus lointain – en ne voyant qu'une partie de la bande.

Dans le cas du *drag-and-pop*, l'attention de l'utilisateur se porte principalement sur son espace moteur, c'est-à-dire autour de son pointeur. L'utilisation des bandes élastique permet à l'utilisateur dont l'attention reste portée sur son espace moteur de se faire une idée plus précise des copies temporaires qui sont faites que si les objets étaient liés par de simples lignes.

L'idée à l'origine des bandes élastiques est de visualiser sous une forme statique un mouvement qui a eu lieu. On se rapproche de ce que font les illustrateurs de bandes dessinées pour représenter des mouvements. Sous l'impulsion de Patrick Baudisch, qui est à l'origine du *drag-and-pop* et des bandes élastiques, nous avons poussé plus loin leur utilisation [Baudisch et al., 2006] au travers d'un ensemble de *feedbacks* visuels statiques permettant de visualiser une animation ou un changement d'état.

La figure montre un ensemble d'exemples d'utilisation des bandes élastiques, de haut en bas et de gauche à droite :

- un déplacement,
- une copie,
- un changement d'état (deux variantes),
- un déplacement temporaire,
- copie temporaire,
- une rotation,
- et deux déplacements simultanés (*switch*).

Algorithmes de dessin Tandis que les bandes élastiques du prototype original du *drag-and-pop*[1] reposaient sur les possibilités de déformation vectorielles du programme *Macromedia Flash*, nous avons utilisé une méthode plus classique permettant d'avoir plus de contrôle sur le résultat final.

Les méthodes de dessin mentionnées ont été implémentées en Delphi et reposent sur l'API graphique GDI+ dont elles dépendent fortement. En effet, ces méthodes se basent sur des

[1] le prototype original du *drag-and-pop* peut être testé à l'adresse http://patrickbaudisch.com/projects/dragandpop/demo/dragandpop.swf

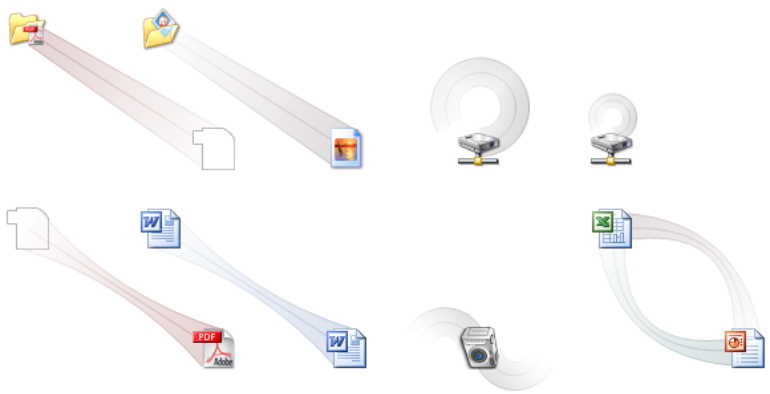

Fig. IV.13: *Exemple d'utilisation des bandes élastiques.*

fonctionnalités de GDI+ telles que les transparences, les dégradés, les chemin complexes, les transformations affines, etc.

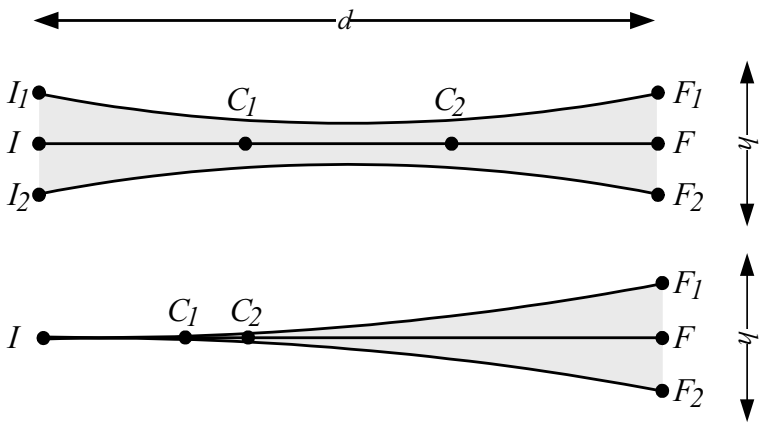

Fig. IV.14: *Méthodes de dessin des bandes élastiques. En haut : pour le drag-and-pop. En bas : pour le push-and-throw et le drag-and-throw.*

La figure IV.14 montre les différents points clés intervenant dans le dessin des bandes élastiques utilisées pour le *drag-and-pop* (voir la figure II.3 – page 43) et pour le *push-and-throw* et ses variantes (voir les figures IV.9 – page 90 –, IV.10 – page 90 – et IV.11 – page 91).

Cette figure n'est pas une simplification où les extrémités sont alignées. En effet, grâce aux transformations affines, le dessin est toujours effectué comme si les extrémités étaient alignées et GDI+ s'occupe d'appliquer les transformations affines – une rotation et une translation – pour que les extrémités de la bande élastique soient correctement placées.

Formes Soit une bande élastique à dessiner entre les points $I(x_1, y_1)$ et $F(x2, y2)$. On calcule la distance d entre I et J. Notons h la hauteur des objets reliés par cette bande élastique. Dans notre prototype Delphi, les objets étaient des icones et avaient donc une hauteur de 32 *pixels*.

Pour dessiner une bande élastique, nous dessinons une forme pleine et trois lignes dont une ligne droite et deux courbes de Bézier cubiques :

- La première courbe de Bézier cubique relie les points I_1 et F_1 en utilisant les points de contrôle C_1 et C_2.
- La seconde courbe de Bézier cubique relie les points I_2 et F_2 en utilisant les points de contrôle C_1 et C_2.
- La ligne droite lie les points I et F.
- La forme pleine est une forme constitué des deux courbes de Bézier décrites ci-dessus et fermée par les segments $[F_1F_2]$ et $[I_2I_1]$.

En utilisant les positions suivantes :

- $I(0,0)$
- $I_1(0,-\frac{h}{3})$
- $I_2(0,\frac{h}{3})$
- $F(d,0)$
- $F_1(d,-\frac{h}{3})$
- $F_2(d,\frac{h}{3})$
- $C_1(\frac{d}{3},0)$
- $C_1(\frac{2d}{3},0)$

La seconde version présentée sur la figure IV.14 montre une bande élastique asymétrique. Les objets reliés par cette bande élastique ne sont pas de la même taille. Nous avons utilisé cette forme de bande élastique pour lier le pointeur – ou plus exactement sont point chaud – à une icone – un fantôme de l'objet déplacé – pour les techniques de *drag-and-throw*, *push-and-throw* et *push-and-throw* accéléré. Le dessin de ce type de bande élastique est similaire au celui décrit précédemment à la différence près que les coordonnées des points I_1, I_2, C_1 et C_2 sont différentes :

- Les points I_1 et I_2 ont les mêmes coordonnées que le point I à savoir $(0,0)$.
- Le point C_1 est placé en $(\frac{d}{4},0)$
- Le point C_2 est placé en $(\frac{d}{3},0)$

Fig. IV.15: *Un utilisateur effectuant un drag-and-pop sur le iWall.*

Couleurs Alors que les couleurs des bandes élastiques étaient codées « en dur » dans le prototype original du *drag-and-pop* – à chaque icone étaient associées deux couleurs : une couleur pour les bords, une couleur de fond –, nous avons opté pour une solution plus générique.

Nous nous basons sur les représentations des objets se trouvant aux extrémités de la bande élastique pour déterminer la couleur de celle-ci. Dans les situations dans lesquelles nous avons travaillé, soit les objets se trouvant aux deux extrémités étaient les mêmes (bande symétrique), soit l'un d'eux était un simple point (le point chaud du pointeur). Nous n'avons donc pas eu de dilemme au niveau du choix de l'objet dont devait dépendre la couleur de la bande élastique.

A partir de la représentation graphique de l'objet manipulé, nous calculons une couleur moyenne en prenant en compte la composante *alpha* de chaque *pixel*. Nous travaillons sur des images codées en 32 bits (4 composantes codées sur 8 bits : rouge, vert, bleu, *alpha*). Lors du calcul de la couleur moyenne de l'image, la couleur de chaque *pixel* est donc pondérée par sa composante *alpha*.

La couleur obtenue est ensuite utilisée pour le dessin des bords et du fond, la différence entre le deux se trouvant au niveau de la composante *alpha*. Nous avons utilisé une valeur *alpha*[2] de 160 pour les bords et une valeur de 48 pour le fond.

IV.2.2 Etude - première partie

Une première étude a été réalisée sur le *iWall* [Johanson et al., 2002a] avec l'aide de Brian Lee.

Procédure expérimentale

Il était demandé aux participants de déplacer des icones vers la corbeille. L'icone à déplacer et l'icone cible étaient mises en évidence comme le montre la figure IV.12 : l'icone à déplacer encadrée de bleu et l'icone cible encadrée de vert.

La procédure expérimentale est très similaire à celle utilisée pour l'étude originale du *drag-and-pop* [Baudisch et al., 2003]. Cependant, les icones source et cible changeaient de position à chaque essai. Douze positions prédéterminées nous ont permis d'obtenir une répartition uniforme des indices de difficulté.

Cette première étude s'est déroulée sur le *iWall*, constitué de trois Smart BOARD™rétro-projetés affichant chacun une résolution de 1024×768 (figure IV.15). Pour interagir avec le *iwall*, les participants avaient le choix entre l'utilisation de leur doigt ou d'un stylo. La machine contrôlant les trois affichages du *iWall* était un Pentium® II 500MHz disposant de 256Mo de RAM.

Six volontaires (dont quatre femmes et deux gauchers) ont participé à cette étude. Cinq des six participants n'avaient que très peu ou pas du tout d'expérience avec l'utilisation des tableaux interactifs.

Chaque participant a testé les six techniques étudiées, à savoir *drag-and-drop*, *pick-and-drop*, *drag-and-pop*, *push-and-throw*, *push-and-throw* accéléré et *push-and-pop*. Les participants ont accompli 48 essais pour chaque technique, soit un total de 288 essais par participant. Les 48 essais étaient groupés en 4 blocs de 12 essais correspondant à 12 positions différentes des icones source et cible, c'est-à-dire à 12 indices de difficulté.

[2] une valeur *alpha* de 255 correspond à une couleur opaque tandis qu'une valeur de 0 correspond à une couleur transparente

Un carré latin a été utilisé pour équilibrer l'ordre de présentation des techniques entre les participants. Les indices de difficulté étaient, quant à eux, présentés dans un ordre aléatoire différent pour chaque bloc. Enfin, chaque participant a disposé de quelques minutes d'entraînement pour chaque technique avant le début des tests chronométrés.

Variables dépendantes

Pour cette étude, nous avons mesuré les temps de déplacement, c'est à dire le temps écoulé entre le moment où un participant sélectionne l'icone source jusqu'au moment où la cible est atteinte. Le temps de déplacement mesuré dans cette étude diffère de celui mesuré dans l'étude précédente sur deux points :

- Si le participant n'atteint pas la cible avec son premier mouvement (s'il « lâche » son objet en cours de route), alors, il lui faut reprendre l'objet et terminer l'opération.
- Dans cette étude, le temps de retour n'est pas mesuré. En effet, dans l'étude précédente, le temps de déplacement intégrait le temps de l'opération de déplacement ainsi que le temps de retour du curseur vers son point de départ.

Nous avons également mesuré les taux d'erreur. Nous distinguons deux types d'erreurs :

- Les erreurs qui peuvent être corrigées. C'est-à-dire, quand une icone est déposée temporairement une ou plusieurs fois sur le bureau. Il faut noter que les dépôts nécessaires au passage des jointures entre les différents affichages pour le *drag-and-drop* n'ont pas été comptabilisés dans ce taux. Pour le *drag-and-drop*, le nombre de dépôts temporaires est égal au nombre total de dépôts auquel on retranche le nombre de jointures entre les objets sources et cible.
- Les erreurs de cible. C'est-à-dire que l'icone à déplacer est déposée sur une mauvaise cible.

Enfin, a l'issue des tests, les participants on répondu à un questionnaire dans lequel il leur était notamment demandé de classer les six techniques qu'ils avaient utilisées par ordre de préférence.

Indice de difficulté Il faut noter que certaines techniques – plus particulièrement les techniques « cible vers pointeur » – modifient l'indice de difficulté de la tâche du fait que la distance entre les objets source et cible est modifiée alors que les dimensions de l'objet cible restent inchangées. En effet, l'indice de difficulté [Douglas et al., 1999] se définit par :

$$ID = \log_2\left(\frac{D}{W} + 1\right)$$

où D est la distance entre l'objet source et l'objet cible et W est la largeur de l'objet cible dans la direction du mouvement.

Si l'on compare l'indice de difficulté d'une tâche donnée en utilisant le *drag-and-drop* et le *push-and-throw*, nous allons obtenir des valeurs identiques. En effet le *push-and-throw* va réduire la distance à parcourir mais va également réduire la largeur de la cible avec le même ratio. Ainsi, le rapport $\frac{D}{W}$ est le même.

A l'inverse du *push-and-throw*, des techniques comme le *drag-and-pop* ou le *push-and-pop* vont réduire la distance D sans modifier la largeur W. Le rapport $\frac{D}{W}$ va donc être différent et induire une valeur différente pour l'indice de difficulté.

Du fait de cette variation de l'indice de difficulté en fonction de la technique utilisée, il nous a paru inadapté de mesurer des variables dépendantes de type indice de performance ou *throughput* dans cette étude.

Résultats

Temps de déplacement Une analyse de variance sur les valeurs moyennes des temps de déplacement a montré des différences significatives entre les techniques ($p < 0,001$). D'autre part, une comparaison pair-à-pair de Dunn a montré que toutes les comparaisons étaient significatives à l'exception de la comparaison entre le *push-and-throw* et le *push-and-throw* accéléré. Précisons que les résultats de la figure IV.16 sont présentés sans les intervalles de confiance pour ne pas en compliquer la lecture.

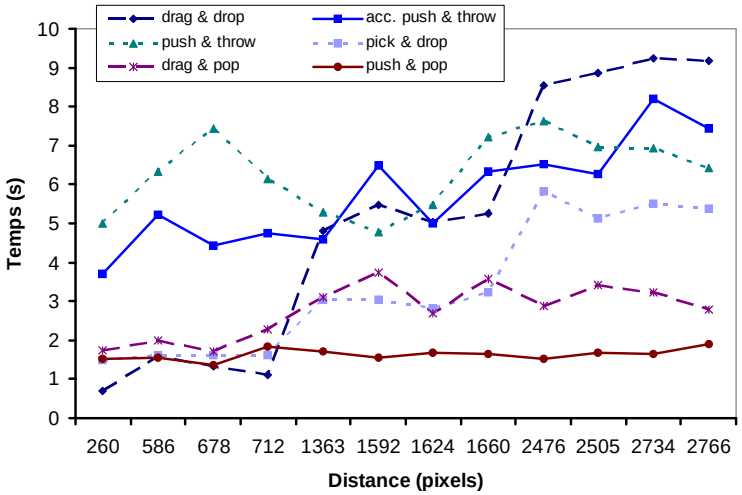

Fig. IV.16: *Temps de déplacement en fonction de la distance.*

Comme on peut le constater sur la figure IV.16, les participants ont, de manière générale, été plus rapides en utilisant le *push-and-pop*. Le *drag-and-pop* a été légèrement moins rapide.

Viennent ensuite le *pick-and-drop* et le *drag-and-drop* dont les performances varient clairement en fonction de la distance. Il est d'ailleurs intéressant de voir les « sauts » des courbes du *drag-and-drop* et du *pick-and-drop* lors du changement d'affichage cible (aux alentours de 1024 et de 2048 pixels).

Enfin, le *push-and-throw* et sa version accélérée ont obtenu de mauvaises performances dans cette étude et ne se sont révélés plus rapides que le *drag-and-drop* que sur les grandes distances.

Taux d'erreur La figure IV.17 présente les différents taux d'erreurs. Dans l'ensemble, il y a eu peu d'erreurs sur les cibles : en moyenne moins de deux cas par test de 36 essais. Par contre, les cas où l'icone à déplacer a été déposée temporairement sont nettement plus élevés. Le *drag-and-drop*, le *push-and-throw*, et le *push-and-throw* accéléré ont des taux supérieurs à 10%, tandis que le *drag-and-pop* à un taux de 7,5%, et que le *push-and-pop* et le *pick-and-drop* ont des taux de l'ordre de 0%.

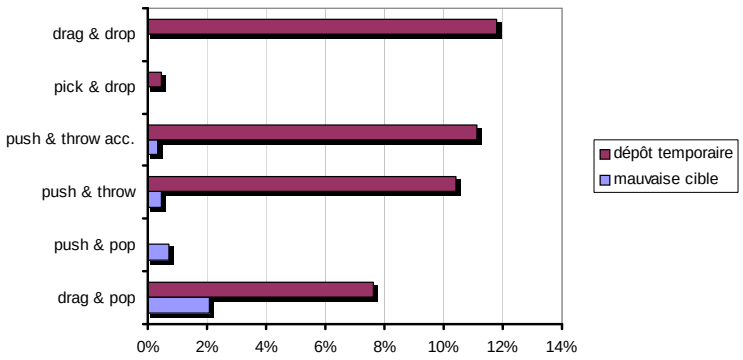

Fig. IV.17: *Taux d'erreurs pour chaque technique.*

Préférences des participants Pour évaluer les préférences des participants, 5 points ont été attribués à la technique préférée de chaque participant, 4 points pour la deuxième technique, etc. jusqu'à 0 point pour la technique la moins appréciée. Les scores sont présentés sur la figure IV.18.

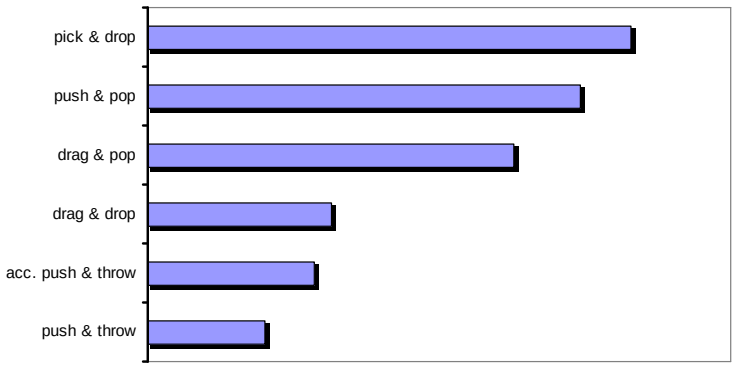

Fig. IV.18: *Préférences des participants.*

Deux groupes apparaissent clairement : le *pick-and-drop*, le *push-and-pop* et le *drag-and-pop* qui ont été appréciés des participants, tandis que le *drag-and-drop* et les *push-and-throw* ont été moins appréciés. Ces résultats correspondent aux temps de déplacement : les participants ont préféré les techniques qui leur offraient les meilleurs performances.

NB : Le lecteur pourra trouver le questionnaire soumis aux participant en annexe B (p. 145).

IV.2.3 Etude - seconde partie

Nous avons reproduit cette étude sur le mur-écran. L'objectif étant double : d'une part, nous voulions confirmer les résultats précédents et d'autre part, nous voulions différencier le *push-and-throw* et le *push-and-throw* accéléré, ce que la première étude n'avait pas permis.

Procédure expérimentale

Pour l'occasion, nous avons utilisé un Smart BOARD^{TM}en projection frontale et une surface augmentée rétro-projetée et équipée d'un Mimio^{TM}[Virtual Ink] en guise de dispositif de pointage. Chacun des deux affichages utilisait une résolution de 1024×768 pour une diagonale de 170 cm. La machine contrôlant ces affichages était un PC disposant d'un Pentium® IV 1,5GHz et de 512Mo de RAM.

C'est 12 participants qui ont participé à cette étude, tous étaient des hommes, un seul était gaucher. Les tâches à effectuer étaient les mêmes à la différence que, pour chaque technique, les participants ont réalisé 36 essais (3 blocs de 12) alors que les participants de la première expérience avaient réalisé 4 blocs soient 48 essais.

Résultats

Les résultats de cette seconde étude ont, dans l'ensemble, été conformes aux observations de la première étude. Cependant, le *push-and-throw* et sa version accélérée ont montré de meilleures performances.

Temps de déplacement Une analyse de variance sur les valeurs moyennes des temps de déplacement a montré des différences significatives entre les techniques ($p < 0,001$). D'autre part, une comparaison pair-à-pair de Dunn a montré que toutes les comparaisons étaient significatives à l'exception de la comparaison entre le *pick-and-drop* et le *push-and-throw* accéléré.

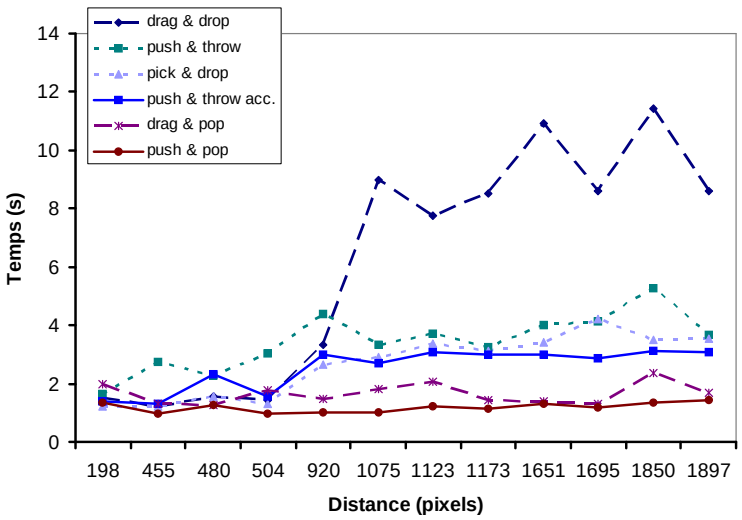

Fig. IV.19: *Temps de déplacement en fonction de la distance.*

De la même manière que dans la première étude, les participants ont été plus rapides lorsqu'ils ont utilisé le *push-and-pop* et légèrement moins rapide en utilisant le *drag-and-pop*. Arrivent ensuite le *push-and-throw* accéléré et le *pick-and-drop*. Les participants ont finalement été les plus lents en utilisant le *push-and-throw* et surtout le *drag-and-drop*. Le *drag-and-drop* offre des performances correctes lorsque les icones sources et cibles se situent sur la même

surface d'affichage. Ces performances chutent très nettement dès lors que le participant doit traverser la jointure entre les deux affichages.

taux d'erreur La figure IV.20 présente les différents taux d'erreur. Ils sont globalement identiques à ceux de la première étude à la différence que le *push-and-throw* accéléré présente ici un taux de dépôts accidentels plus faible.

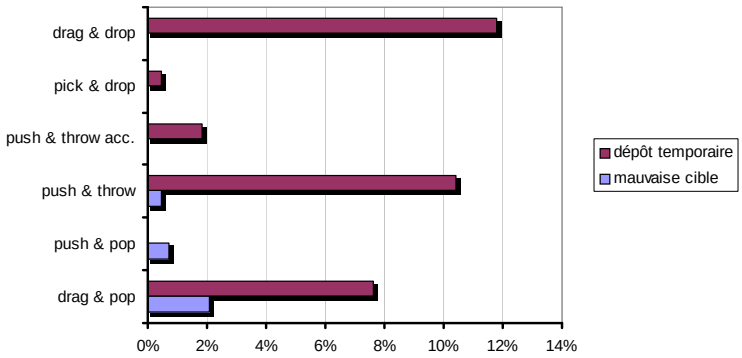

Fig. IV.20: *Taux d'erreurs pour chaque technique.*

préférences des participants Dans cette seconde étude, les préférences des utilisateurs ont été un peu différentes : les deux variantes du *push-and-throw* ont été mieux appréciées tandis que le *pick-and-drop* a été moins bien classé.

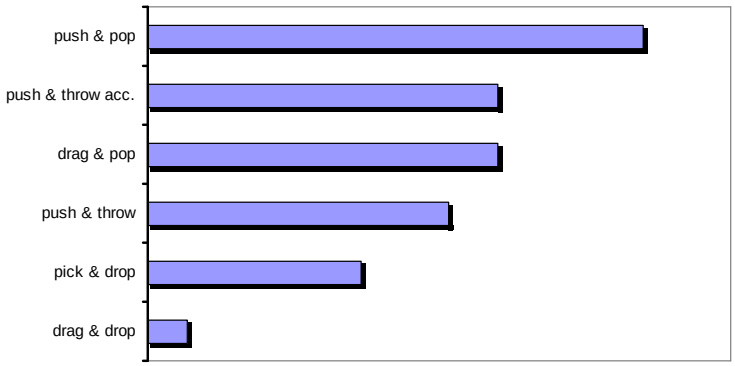

Fig. IV.21: *Préférences des participants.*

NB : Le lecteur pourra trouver le questionnaire soumis aux participant en annexe B (p. 145).

IV.2.4 Discussion

Les deux études présentées ici ont globalement confirmé nos hypothèses initiales.

Confirmant les résultats de Baudisch et al. [Baudisch et al., 2003], le *drag-and-drop* offre de bonnes performances lorsque les icones sources et cibles se situent sur la même surface d'affichage. Mais ces performances chutent très nettement dès lors que le participant doit traverser une jointure entre deux affichages. Nous avons également pu remarquer que le *pick-and-drop* était également affecté par la distance mais dans une moindre mesure.

Pour toutes les autres techniques, la distance entre la source et la cible n'a pas eu d'influence sensible sur les performances. Cependant, on peut remarquer que les techniques ne nécessitant qu'une réorientation ont montré de meilleurs performances que les techniques qui nécessitent de constamment observer les *feedbacks* pour ajuster son mouvement.

Au final, le *push-and-pop* apparaît comme une technique utile et efficace. Elle a en effet surpassé toutes les autres techniques, dont ses deux « ancêtres » : le *push-and-throw* et le *drag-and-pop*. Le *push-and-pop* a également obtenu des taux d'erreur extrèmement faibles, ce qui est l'explication principale de ses bons résultats comparés au *drag-and-pop*. En effet, tandis que le *drag-and-pop* nécessite de faire attention au mouvement initial dont dépend la sélection des cibles probables, le *push-and-pop* évite ces contraintes et ne présente donc plus de risque de sélectionner un mauvais ensemble de cibles.

Parmi les techniques dites « pointeur vers cible », il n'a pas été possible de départager le *push-and-throw* et le *push-and-throw* accéléré à l'issue de la première étude. Cependant, la seconde étude a montré la supériorité du *push-and-throw* accéléré à tous les niveaux : aussi bien au niveau des temps de déplacement et des taux d'erreur qu'au niveau des préférences des participants. Enfin, nous expliquons les mauvaises performances des deux variantes du *push-and-throw* lors de la première étude par la configuration matérielle utilisée : d'une part, la machine contrôlant les trois Smart BOARD^{TM} était vieillissante (PIII 500MHz) ce qui a limité le taux de rafraîchissement des *feedbacks*, la fluidité de ceux-ci ayant une importance primordiale dans le succès de ces techniques (voir tableau IV.1). D'autre part, le *iWall* est très large, et il a parfois été difficile pour les participants de voir correctement l'objet cible tandis qu'il manipulaient l'objet source.

IV.3 Conclusion

IV.3.1 Bilan

Nous avons, au travers de différentes études, comparé une large palette de techniques d'interaction destinées à remplacer le *drag-and-drop* dans certaines conditions particulières.

Dans un premier temps nous avons comparé nos méthodes de lancer avec le *drag-and-drop* lui même. Et, bien que les conditions expérimentales aient été favorables au *drag-and-drop*, les méthodes de lancer se sont très bien comportées.

Nous avons ensuite voulu comparer nos méthodes aux autres alternatives du *drag-and-drop*. Il s'en est suivi une étude complète sur les méthodes de style *drag-and-drop* sur les murs interactifs qui nous a elle même conduit à proposer deux nouvelles techniques : le *push-and-throw* accéléré et le *push-and-pop*.

Ce sont finalement ces deux techniques qui sortent vainqueurs de l'étude. En effet, le *push-and-pop* s'est montré le plus performant. On peut d'ailleurs dire que les techniques « cibles vers pointeur » ont été globalement les plus performantes pour les tâches demandées. Cependant, ces techniques ont une limitation importante : elles ne fonctionnent que s'il y a une cible identifiée à l'opération de type *drag-and-drop*. Quelle cible approcher du pointeur lorsqu'un uti-

lisateur déplace un mot dans la zone d'édition d'un traitement de texte ? Quelle cible approcher lorsque l'utilisateur qui déplace une icone souhaite simplement réorganiser son bureau ?

Alors, quelle technique utiliser dans de telles situations ?

Le *drag-and-drop* ? Il atteint clairement ses limites lorsque l'affichage est distribué.

Le *pick-and-drop* ? Il nécessite que pourvoir atteindre physiquement la cible.

Les techniques qui proposent une approche « pointeur vers cible »n'ont pas ces défauts. Et parmi elles, c'est le *push-and-throw* accéléré qui s'est montré le plus performant.

IV.3.2 La technique ultime

Une technique intéressante est donc la combinaison du *push-and-pop* et du *push-and-throw* accéléré. Nous avons donc implémenté un système permettant de combiner ces deux techniques de telle manière que le *push-and-pop* puisse se transformer en *push-and-throw* accéléré.

La méthode de désactivation consiste, pour l'utilisateur, à amener le pointeur au point initial de son mouvement. Dans le prototype utilisé pour la seconde étude, si un utilisateur souhaite réorganiser son bureau, il enfonce le bouton de sa souris sur l'icone à déplacer puis il déplace son pointeur. Après un mouvement de quelques pixels, les *feedbacks* du *push-and-pop* apparaissent, l'utilisateur revient alors à son point de départ (la tolérance est la même que le nombre de pixels dont il faut déplacer le pointeur pour que le *push-and-pop* se déclenche - 4 pixels dans notre implémentation). L'utilisateur peut alors déplacer son icone en *push-and-throw* accéléré.

Le diagramme états-transitions de la figure IV.22 présente le fonctionnement de l'instrument de *push-and-pop* pouvant se transformer en *push-and-throw* (accéléré).

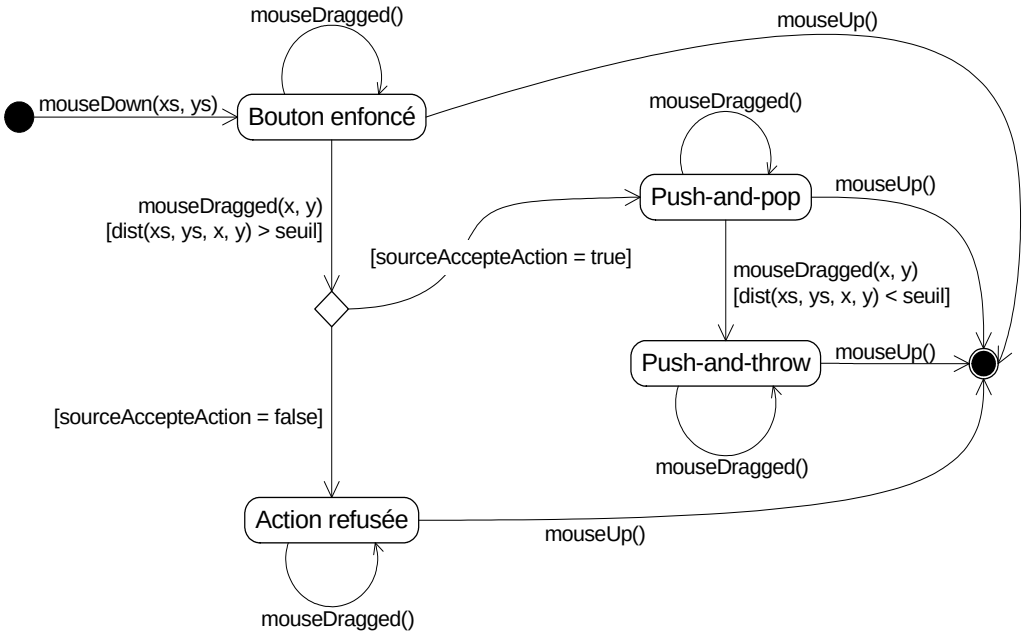

Fig. IV.22: *Diagramme états-transitions du push-and-pop.*

L'instrument analyse tous les évènements de pointage et en utilise trois plus particulièrement :

- *mouseDown()* lorsque le bouton de la souris est enfoncé – ou que le stylo entre en contact avec la surface interactive,
- *mouseDragged()*[3] lorsque le pointeur est déplacé et que le bouton est toujours enfoncé – ou que le stylo est déplacé sur la surface,
- *mouseUp()* lorsque le bouton de la souris est relâché – ou que le stylo quitte la surface interactive.

Etats Quatre états apparaissent sur le diagramme IV.22 :

- *Bouton enfoncé*
- *Action refusée*
- *Push-and-pop*
- *Push-and-throw*

Transitions Lorsque l'utilisateur effectue un *push-and-pop*, il enfonce le bouton de sa souris (état *bouton enfoncé*) et déplace le pointeur. Lorsque ce déplacement dépasse un certain seuil, l'opération de *push-and-pop* est proposée au composant source de l'opération.

Si celui-ci refuse l'opération, l'instrument se retrouve dans l'état *action refusée*. L'instrument sortira de cet état lorsque le bouton de la souris sera relâché et aucune action ne sera réalisée.

Si le composant source accepte l'opération, l'instrument passe à l'état *push-and-pop*. Si l'utilisateur déplace son pointeur vers le point initial de l'opération, l'instrument passe à l'état de *push-and-throw*. Que l'instrument soit à l'état *push-and-pop* ou *push-and-throw*, le passage de l'instrument à l'état final se produit lorsque l'utilisateur relâche le bouton de se souris.

IV.3.3 Vers une généralisation de ces techniques

L'annexe A (page 137) présente une réflexion sur la généralisation des techniques étudiées dans ce chapitre à un autre type d'interaction : l'activation qui peut se traduire par un clic sur un bouton, une icone, etc.

Nous sommes arrivés à la conclusion que les méthodes de type pointeur-vers-cible étaient plus adaptées à ce type d'interaction du fait qu'elles ne nécessitent pas d'informations particulières sur les composants cibles de l'interaction.

[3] Peu d'APIs différencient les évènements *mouseDragged()* et *mouseMoved()*. Mais ceci n'a pas d'importance dans le processus décrit ici.

Chapitre V

POIP : UNE API POUR LA MANIPULATION DIRECTE DANS DES ENVIRONNEMENTS D'AFFICHAGE DISTRIBUÉ

Sommaire

Le modèle d'interaction instrumental des techniques de type *drag-and-drop* décrit dans le chapitre III peut également être considéré comme un modèle d'implémentation. C'est en s'appuyant sur ce modèle que nous avons conçu une API permettant l'utilisation des techniques de type *drag-and-drop* dans les environnements d'affichages distribués.

V.1 Objectifs

Dans ce chapitre, nous présentons une API dont l'objectif est de permettre l'utilisation des techniques de type *drag-and-drop* dans les environnements d'affichage distribués. Une des propriétés primordiale de cette API est sa simplicité. Nous ne parlons pas ici de la complexité interne de l'API mais sa simplicité d'utilisation par un développeur qui désire que ses composants supportent les techniques de type *drag-and-drop*.

La simplicité d'utilisation de cette API doit être comparable à l'utilisation des modèles de *drag-and-drop* classiques. Le support des environnements distribués rend cette API plus complexe par certains aspects mais elle permet également de simplifier certains aspects du modèle du *drag-and-drop*, puisqu'elle implémente le modèle décrit dans le chapitre III.

Comme nous l'avons vu dans les chapitres I et II, il existe peu d'implémentations de techniques de type *drag-and-drop* supportant les environnements distribués. De plus, celles-ci ont été développées de manière *ad hoc*. Notre objectif est ici de proposer une API ouverte. Il devient alors possible de créer de nouvelles techniques de type *drag-and-drop* facilement. Il suffit pour cela de créer un nouvel instrument, le cœur de l'API s'occupant de la plus grosse partie du travail (connexion réseau, partage de surface, etc.).

V.2 Généralités sur l'implémentation

L'API *PoIP*[1] a été implémentée en *Java 5.0*. Pour décrire cette API, nous nous baserons sur certains concepts ou fonctionnalités de Java. En particulier, pour les communications réseau, la technologie *RMI* (*Remote Method Invocation*) a été utilisée en raison de sa simplicité d'utilisation ainsi que pour ses performances. Pour invoquer une méthode d'un objet distant en utilisant RMI, il est nécessaire de connaître les méthodes de l'objet distant. Pour cette raison, une classe dont les méthodes des instances peuvent être invoquées à distance implémente une interface connue de tous. Ainsi, l'objet distant n'a besoin de connaître que l'interface implémentée pour invoquer une méthode.

D'autre part, cette API repose sur la gestion des évènements par AWT. Le système d'évènements de AWT est utilisé pour capturer les évènements d'entrée provenant du système ainsi que pour générer de nouveaux évènements.

Comme le montre la figure V.1, nous avons découpé notre API en plusieurs paquetages. Nous avons tout d'abord eu besoin de permettre le partage de surfaces. C'est la base nécessaire à l'implémentation des techniques de types *drag-and-drop* dans les environnements distribués. Le partage des surfaces se matérialise dans les paquetages *PoIP.client* et *PoIP.server*. Les techniques de manipulation directe de type *drag-and-drop* sont, quant à elles, implémentées dans le paquetage *PoIP.dm*. Enfin, le paquetage *PoIP.topology* contient les classes qui permettent la gestion de la topologie des surfaces partagées, c'est-à-dire les positions relatives des surfaces les unes par rapport aux autres.

[1] **P**ointer **o**ver **IP**

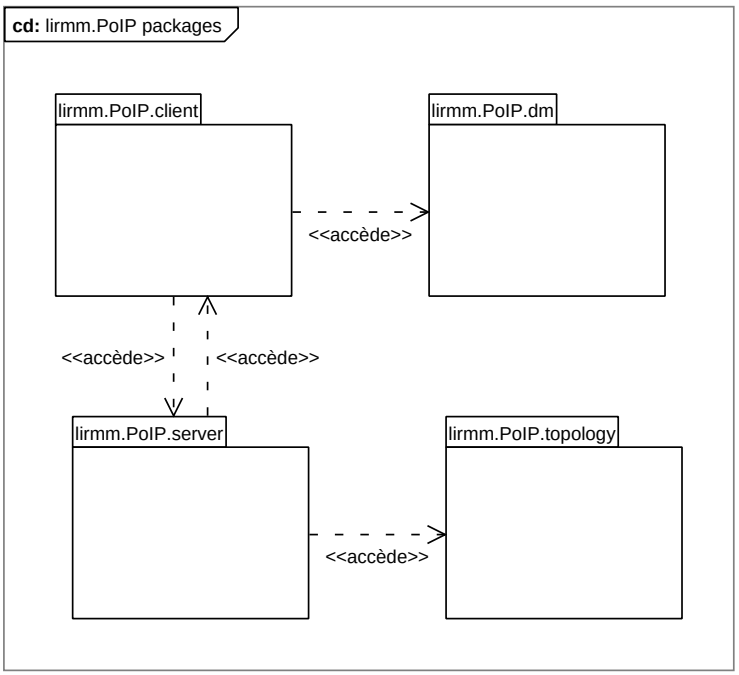

Fig. V.1: *Les paquetages de l'API PoIP.*

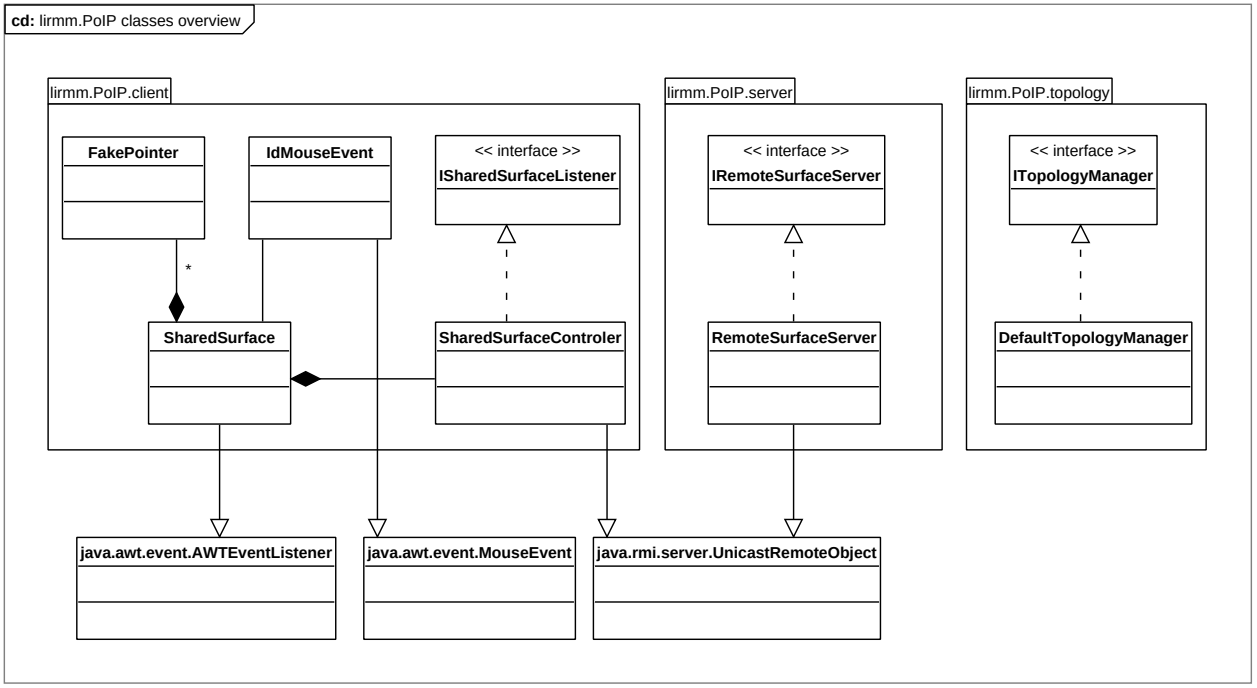

Fig. V.2: *Vue d'ensemble des paquetages et des classes mis en œuvre dans l'API.*

Nous allons décrire plus en profondeur les différentes composantes de l'API, en commençant par le partage des surfaces (client et serveur), puis en terminant par l'implémentation du modèle décrit dans le chapitre III.

V.3 Partage de surfaces

Les différentes entités permettant le partage de surfaces sont réparties dans les paquetages *lirmm.PoIP.client*, *lirmm.PoIP.server* et *lirmm.PoIP.topology* regroupant respectivement les classes nécessaires à une application cliente – possédant une surface partagée –, les classes nécessaires au serveur et les classes permettant la gestion de la topologie des surfaces partagées. On peut voir une vue d'ensemble des classes de ces différents paquetages sur la figure V.2 et le détail des paquetages *client* et *server* sur la figure V.3.

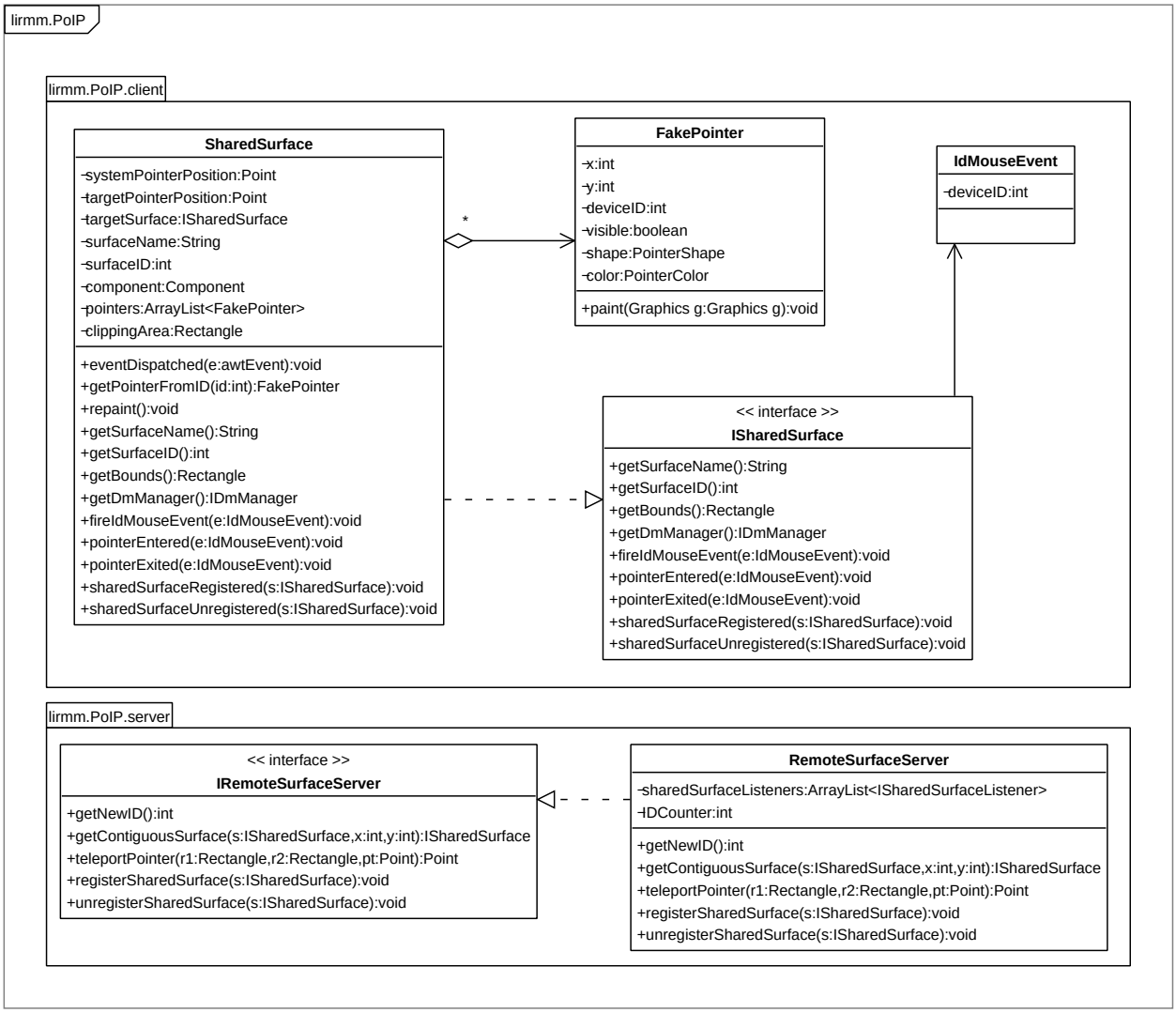

Fig. V.3: *Paquetages client et serveur.*

Dans un premier temps, nous allons voir le comportement de la partie serveur de l'API.

Puis nous verrons la partie cliente dans un second temps.

V.3.1 Serveur de surfaces partagées

La réalisation du serveur de surfaces partagées implique deux entités : une interface *IRemoteSurfaceServer* et une classe *RemoteSurfaceServer*. Ceci est dû à l'utilisation de *RMI* : l'interface décrit les méthodes qui peuvent être invoquées à distance tandis que la classe implémente cette interface.

Le serveur de surfaces partagées a un comportement simple :

- Il gère les identifiants des surfaces (et des dispositifs de pointage). Un identifiant est attribué à une surface partagée lorsque celle ci s'enregistre auprès du serveur.
- Il maintient une liste des surfaces partagées. Celles-ci s'enregistrent auprès du serveur par un appel à la méthode *registerSharedSurface()* et se dés-enregistrent par un appel à *unregisterSharedSurface()*. A chaque appel d'une de ces deux méthodes, les autres surfaces partagées sont notifiées par les méthodes *sharedSurfaceRegistered()* et *sharedSurfaceUnregistered()*.
- Il gère la topologie des surfaces. La méthode *getContigousSurface()* permet d'obtenir une surface contiguë à une autre et la méthode *teleportPointer()* permet de calculer la position du pointeur lors du passage d'une surface à une autre.

V.3.2 Topologie

La gestion de la topologie de notre API est relativement simple. Nous avons opté pour une topologie des surfaces en deux dimensions. La raison est que les dispositifs de pointage généralement utilisés fonctionnent sur deux dimensions.

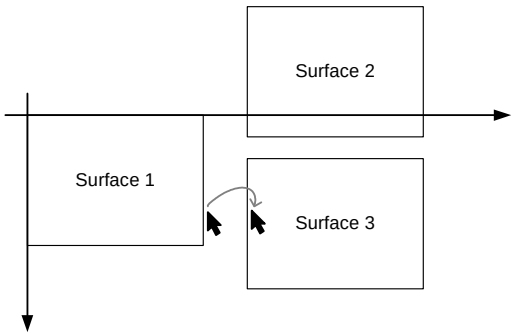

Fig. V.4: *Exemple de topologie des surfaces partagées.*

On peut voir sur la figure V.4 un exemple d'utilisation de la topologie des surfaces. Trois surfaces sont partagées, chacune sur un affichage différent. Dans cet exemple, un pointeur se déplaçant sur la surface 1 atteint le bord droit de la surface. Pour savoir où « téléporter »ce pointeur, les méthodes de gestion de la topologie entrent en jeu : dans un premier temps, la méthode *getContigousSurface(s : ISharedSurface, x : int, y : int)* est appelée. Le paramètre s représente la surface où se trouvait le pointeur (surface 1), x et y la position du pointeur par rapport à l'origine de cette surface – x et y sont nécessairement en dehors des limites de la surface. Dans l'exemple de la figure, le résultat de l'appel sera la surface 2. La méthode

teleportPointer(r1 : Rectangle, r2 : Rectangle, x : int, y : int) est ensuite invoquée. les rectangles *r1* et *r2* représentent respectivement les limites des surfaces 1 et 2 tandis que x et y représentent la position du pointeur par rapport à l'origine de la surface 1. Le résultat de l'appel sera la nouvelle position du pointeur sur la surface 2, c'est à dire par rapport à l'origine – coin supérieur gauche – de la surface 2.

V.3.3 Surface partagée

Une surface partagée se traduit par la création d'une instance de la classe *SharedSurface*. cet objet est associé à un composant graphique – instance d'une classe héritant de *java.awt.Component*. Elle possède un nom ainsi qu'un identifiant unique qui est fourni par le serveur de surfaces partagées lors de son enregistrement.

Elle agit sur ce composant à deux niveaux : au travers des évènements des dispositifs de saisie et également au niveau graphique.

Evènements des dispositifs de saisie

L'action d'un objet de type *SharedSurface* sur les évènements d'entrée du composant associé se fait en deux temps : tout d'abord, les évènements sont capturés puis ils sont reproduits, éventuellement sur une autre surface en les augmentant avec l'identifiant du dispositif de saisie qui a généré l'évènement.

La capture des évènements se fait en ajoutant un *AWTEventListener* à la *Toolkit* par défaut. Ces évènements sont appliqués à un faux pointeur et sont re-générés en y ajoutant un identifiant de périphérique de saisie.

L'utilisation d'un *AWTEventListener* a cependant une limite : tous les évènements de la machine virtuelle sont capturés et il n'est donc pas possible de partager plusieurs zones dont les composants associés sont gérés par la même machine virtuelle (figure V.5). Cette limitation n'a cependant rien de rédhibitoire.

Tous les évènements d'entrées sont augmentés pour identifier le dispositif à l'origine d'un évènement. Une telle identification des évènements est nécessaire pour la gestion d'un environnement multi utilisateur. Mais en réalité, l'augmentation des évènements se fait après qu'ils aient été transmis à la machine virtuelle Java. A ce niveau, le gestionnaire de surface partagée (*SharedSurface*) capture les évènements, les augmente puis les redirige vers la bonne surface.

Ce fonctionnement implique qu'il n'y ait qu'une seule zone partagée par JVM. Dans le cas contraire, plusieurs gestionnaires de surface partagées captureraient les mêmes évènements.

Par ailleurs, nous remarquons que l'identification des dispositifs de saisie est en fait une identification de la surface partagée sur laquelle un évènement a été capturé.

Pointeurs

De faux pointeurs sont utilisés afin de permettre l'utilisation simultanée de plusieurs pointeurs sur une surface. Ces faux pointeurs sont différenciés par leurs couleurs de fond comme on peut le constater sur la figure V.6.

Chaque surface contrôle un unique pointeur (qui possède le même identifiant que la surface). Cependant, les pointeurs pouvant se déplacer d'une surface à l'autre, il peut y avoir plusieurs pointeurs sur la même surface à un moment donné.

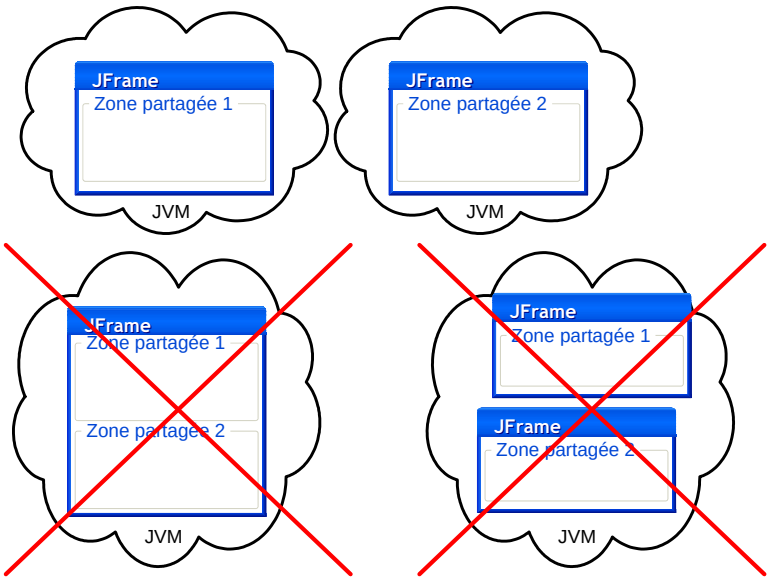

Fig. V.5: *Une seule zone peut être partagée dans une JVM.*

Fig. V.6: *Les six couleurs de fond disponibles pour les pointeurs.*

Le curseur système, quant à lui, est contrôlé grâce à un *Robot* – classe *java.awt.Robot* de l'API Java qui permet entre autre de déplacer le pointeur système. A chaque mouvement du pointeur système, celui-ci est recentré sur le composant graphique associé et son mouvement est reproduit sur le faux pointeur correspondant. Cependant, l'utilisation du *Robot* n'est pas systématique. En effet, son utilisation est incompatible avec les dispositifs de pointage direct (Mimio^{TM}, Smart BOARD^{TM}, etc.) qui contrôlent eux-mêmes déjà le pointeur système.

Lorsque le pointeur système se trouve au dessus du composant graphique lié à une *SharedSurface*, il devient invisible tandis que le faux pointeur est visible.

Il est possible de spécifier à une surface partagée si le dispositif de pointage utilisé est direct ou indirect. Un *Robot* est utilisé uniquement si le dispositif de pointage est indirect (souris, pavé tactile, etc.). Dans le cas d'un dispositif de pointage direct, un *Robot* n'est pas utilisé. L'inconvénient est alors que le faux pointeur ne peut pas se déplacer sur les autres surfaces partagées. Cependant, cette limitation est tout à fait cohérente : un système de pointage direct est limité à sa surface.

Feedbacks

Lorsqu'un composant est associé à une instance de *SharedSurface*, celle-ci créé un *LayeredPane* au dessus du composant graphique. Ainsi la *SharedSurface* dispose d'une zone sur laquelle elle peut afficher des *feedbacks*.

Ce *LayeredPane* est utilisé pour afficher les faux pointeurs. Il est également mis à disposition du *DmManager* qui peut y dessiner les *feedbacks* de type *drag under* et *drag over* ainsi que les *feedbacks* d'instrument.

Le *LayeredPane* utilisé est placé au dessus des autres *LayeredPane* pouvant exister afin que les faux pointeurs ne puissent jamais se retrouver affichés en dessous d'un autre composant. Pour la même raison, les différents *feedbacks* du *DmManager* sont dessinés avant les faux pointeurs. Ainsi, ceux-ci apparaissent toujours au dessus de tous les éléments de l'interface utilisateur.

V.4 Manipulation directe

Le paquetage *lirmm.PoIP.dm* contient les classes implémentant le modèle d'interaction instrumentale des techniques de type *drag-and-drop* décrit dans le chapitre III ainsi que certaines techniques décrites dans les chapitres II et IV.

V.4.1 Implémentation du modèle

DmManager

L'implémentation du *DmManager* a nécessité, de la même manière que *SharedSurface*, l'introduction d'une interface *IDmManager*. Cette interface squelette permet aux objets distants de connaître les méthodes d'un *DmManager* qui peuvent être invoquées à distance.

Un unique *DmManager* est créé lors de la création d'une *SharedSurface*. Les *DmManagers* des différentes *SharedSurfaces* peuvent communiquer entre eux grâce au fait que les *SharedSurfaces* se connaissent entre elles : un *DmManager* peut connaître la liste des *SharedSurfaces* distantes en les demandant à la *SharedSurface* à laquelle il appartient et il peut demander à chacune des *SharedSurfaces* distante l'accès à leur *DmManager*.

Instruments maîtres et esclaves

Le modèle du chapitre III distingue les instruments maître et esclave. La notion d'instrument est primordiale dans notre modèle. C'est grâce à elle que notre modèle et donc notre implémentation peuvent être étendus aisément. Tous les mécanismes de bases sont fournis et le développement d'une nouvelle technique de type *drag-and-drop* se traduit uniquement par le développement d'un nouvel instrument – une classe héritant de *DmAbstractInstrument*.

Il nous a paru important que la notion d'instruments maîtres et esclaves complique le moins possible le développement d'un nouvel instrument. Pour cette raison, les instruments maîtres et esclave sont dans la pratique des instances de la même classe. La différenciation des deux types d'instruments se fait grâce à une propriété d'instance *rank* qui peut avoir la valeur *INSTRUMENT_MASTER* ou *INSTRUMENT_SLAVE*.

Il existe un unique instrument maître pour chaque *DmManager*, et donc pour chaque *SharedSurface* et machine virtuelle. Lorsque la *SharedSurface* capture tous les évènements de la machine virtuelle, elle les fait suivre au *DmManager* qui lui même les fait suivre à son instrument maître. Ainsi, l'instrument maître peut analyser les évènements d'entrées et agir en conséquence (notification des composants graphiques).

Les communications entre l'instrument maître et les instruments esclaves se fait par l'intermédiaire des *DmManagers* et de deux méthodes *slaveToMasterMessage(m)* et *masterToSlaveMessage(m)*. Les messages *m* ainsi que les résultats de ces méthodes ont comme seule contrainte de pouvoir être transmis au travers du réseau, c'est-à-dire d'implémenter l'interface *java.io.Serializable*.

Les différents instruments sont identifiés grâce au même identifiant qui est utilisé pour les *SharedSurfaces*. Un instrument maître et tous ses instruments esclaves possèdent l'identifiant de la *SharedSurface* à laquelle appartient l'instrument maître. Un *DmManager* ne peut pas posséder plusieurs instruments esclaves ayant le même maître. Tous les instruments esclaves d'un *DmManager* ont donc des identifiants différents.

Evènements et écouteurs

Les interfaces *IDmSourceListener* et *IDmTargetListener* sont directement issues du modèle du chapitre III. Cependant, il est possible que les méthodes de ces interfaces soient invoquées par un instrument distant (via *RMI*) et pour que ces interfaces soient des squelettes *RMI*, il convient de gérer d'éventuelles exceptions *RemoteException*. Or, nous ne voulions pas que les développeurs de composants supportant les techniques de type *drag-and-drop* aient affaire à ces exceptions : nous voulions que l'aspect distribué de ces techniques soit transparent. C'est pourquoi nous avons introduit deux nouvelles interfaces *IDmSourceListenerProxy* et *IDmTargetListenerProxy* qui jouent le rôle de mandataires entre les *DmManagers* – et donc les instruments – et les interfaces *IDmSourceListener* et *IDmTargetListener* implémentées par le développeur final. Ces objets mandataires sont créés lors de l'enregistrement des composants comme sources ou cibles probables auprès du *DmManager*.

Le fait que les méthodes des interfaces *IDmSourceListener* et *IDmTargetListener* puissent être invoquées à distance présente une autre conséquence : les évènements – *DmSourceEvent* et *DmTargetEvent* – doivent pouvoir être transmis via le réseau et donc implémenter l'interface *java.io.Serializable*.

Les données

Comme cela a déjà été évoqué précédemment, nous ne nous sommes pas focalisés sur la gestion des formats de données. Dans notre implémentation des techniques de type *drag-and-drop*, la gestion des données est simple et ne présente que deux contraintes :

- Les formats de données doivent implémenter l'interface *Serializable* afin de pouvoir être transmise facilement par le réseau,
- Les formats de données doivent avoir un nom unique afin de différencier les différents formats qui peuvent être fournis.

Une interface *ITransferableData* minimale a été introduite : elle étend l'interface *Serializable* en y ajoutant une méthode *getDataType()* qui doit renvoyer le nom du format de données.

V.4.2 Implémentation des instruments

Toutes les techniques décrites dans les chapitres II et IV n'ont pas été développées. Seuls ont été développés les instruments de *drag-and-drop* et de *push-and-pop* (plus désactivation en *push-and-throw*), technique qui s'est avérée être la plus performante (voir chapitre IV).

drag-and-drop

L'instrument de *drag-and-drop* (figure V.7 – *a*) a été implémenté sans avoir recours au mécanisme de *drag-and-drop* interne de Java. Il analyse directement les évènements d'entrées sur les composants sources et cibles.

Un *fantôme* est fourni par le composant source et est affiché à la position du pointeur par un des instruments esclaves.

Les communications entre l'instrument maître et ses instruments esclaves ont lieu à plusieurs moments :

- pour retrouver l'écouteur du composant source (au début d'un *drag-and-drop*),
- à chaque déplacement du faux pointeur (qui implique un déplacement du fantôme),
- lorsque le composant cible change ,
- et lorsque le fantôme change (ce qui n'arrive qu'une seule fois).

Notons finalement, que si un composant cible refuse un objet lors de l'évènement *dmEntered()*, alors il ne sera pas notifié des autres évènements (*DmTrgOver()*, *dmExited()* or *dmReceive()*).

push-and-throw

L'instrument de *push-and-throw* est implémenté de la même manière que dans le chapitre IV. Il y a cependant deux différences notables. D'une part, l'affichage de la trajectoire est faite par une ligne et n'utilise pas les bandes élastiques. D'autre part, on voit apparaître dans la zone de décollage (figure V.7 – *c* et *d*) les différentes surfaces de l'espace de travail. Ces différentes surfaces sont placées dans la zone de décollage en fonction de la topologie des surfaces telle qu'elle est définie par le serveur de surfaces partagées.

Il faut noter que, dans l'exemple de la figure V.7 – *c* et *d*, l'affichage de la trajectoire est distribué entre les différents instruments esclaves. La correspondance des différentes parties de la trajectoire dans le champ visuel des utilisateurs dépend de l'exactitudes des informations topologiques dont dispose le serveur de surfaces partagées.

push-and-throw accéléré

Le *push-and-throw* accéléré a également été implémenté de la même manière que dans le chapitre IV. Nous pouvons faire les mêmes remarques concernant la trajectoire que pour le *push-and-throw*. Un système de débrayage est utilisé pour « recentrer » le pointeur au cour du mouvement (voir chapitre IV).

push-and-pop

L'instrument de *push-and-pop* (figure V.7 – *b* et *d*) créé un tableau de cibles autour du pointeur. Les cibles sélectionnées pour constituer ce tableau sont celles qui acceptent l'objet déplacé. En effet, lors de la création de l'ensemble de cibles, les méthodes *dmEntered()* et *dmExited()* de chaque cibles sont appelées. Si l'objet est accepté – appel de la méthode *accept()* de l'évènement reçu par *dmEntered()* –, la cible est incluse dans le tableau. Sinon – appel de la méthode *reject()* de l'évènement reçu par *dmEntered()* –, la cible est ignorée.

Les cibles sélectionnées sont organisées avec les méthodes décrites sur la figure IV.11. L'algorithme prend en compte la topologie des surfaces partagées pour le placement des cibles et a été amélioré pour prendre en compte des cibles dont les représentations ont des tailles différentes. Si une cible fourni une représentation graphique trop grande, celle-ci est réduite soit par une mise à l'échelle (*scaling*) soit par un découpage (*cropping*). La meilleure méthode doit encore être déterminée. Des exemples de composants dont les représentations graphiques sont trop grandes sont les composants d'édition (*text areas* ou *text fields*).

La zone de décollage (i.e. le tableau de cibles) contient un disque noir au point de départ du geste de l'utilisateur. Si l'utilisateur amène à nouveau son pointeur sur ce disque, alors l'instrument de *push-and-pop* va se transformer en un instrument de *push-and-throw* accéléré. Comme nous l'avons déjà mentionné, l'avantage du *push-and-throw* est de permettre d'atteindre toutes les parties de l'espace de travail.

V.5 Discussion sur l'implémentation

Il faut noter que l'implémentation des techniques de type *drag-and-drop* présentée dans ce chapitre a été réalisée au niveau applicatif.

On pourrait présenter les différentes couches logicielles comme suit :

Application
toolkit
Système de fenêtrage
Système d'exploitation

Il convient de noter qu'il n'est pas toujours aisé de différencier le système de fenêtrage du système d'exploitation – en particulier sous MS Windows.

Nous n'avions pas les moyens pour notre implémentation de modifier un système existant. Cependant, nous pensons que l'implémentation des techniques de type *drag-and-drop* doit être faite au niveau des systèmes de fenêtrage et non au niveau des *toolkits* comme c'est le cas actuellement (voir chapitre III).

L'implémentation du *drag-and-drop* au niveau des *toolkit* est à l'origine de comportements hétérogènes entre des applications n'utilisant pas les mêmes *toolkits*. D'autre part, nous proposons un modèle ouvert dans lequel il est possible de changer l'instrument utilisé. Or, il est nécessaire que le même instrument soit utilisé sur toutes les applications, ne serait-ce que parce qu'une opération de type *drag-and-drop* peut faire intervenir plusieurs applications (source et cible(s)).

Par ailleurs, on peut constater que certaines implémentations du *drag-and-drop*, bien que situées au niveau *toolkit*, utilisent des mécanismes externes afin d'offrir une meilleure homogénéité entre les *toolkits*. On peut par exemple citer le protocole de transfert *Uniform Transfer Protocol* utilisé sous X-Window ou OLE utilisé sous MS Windows.

Pour ces raisons, nous pensons que le modèle présenté dans ce chapitre devrait être idéalement implémenté au niveau du système de fenêtrage.

V.6 Conclusion

Nous avons présenté dans ce chapitre l'implémentation du modèle décrit dans le chapitre III dans l'API *PoIP*. Cette implémentation nous a amené au développement des bases nécessaires aux collecticiels supportant les environnements d'affichages distribués.

L'étape suivante est logiquement l'utilisation de cette API *PoIP* dans une application qui mette en œuvre plusieurs utilisateurs dans un environnement distribué. C'est l'objet du chapitre qui suit.

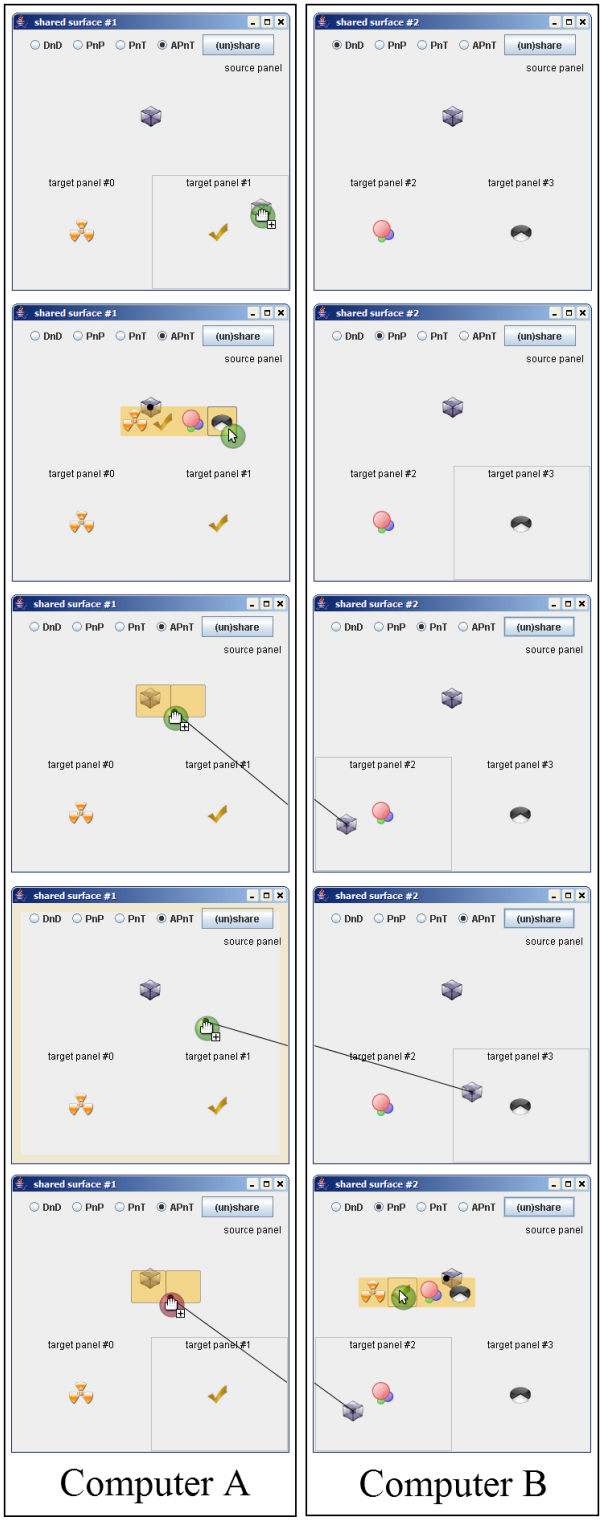

Fig. V.7: *(De haut en bas) (a) drag-and-drop (b) push-and-pop (c) push-and-throw (d) push-and-throw accéléré (e) deux manipulations simultanées : un push-and-throw de A vers B et un push-and-pop de B vers A.*

Chapitre VI

ORCHIS : UNE APPLICATION POUR LA MANIPULATION COLLABORATIVE DE SIGNETS

Sommaire

L'API *PoIP* décrite dans le chapitre V a été utilisée pour le développement d'une application de visualisation collaborative de signets – aussi appelés *bookmarks* ou favoris. Ce projet est en réalité une évolution de l'application *Edelweiss*.

Edelweiss [Hascoët and Sackx, 2002] est une application adaptée au mur-écran (voir chapitre d'introduction) et dédiée à la gestion de collections de signet : édition, consultation, organisation. *Edelweiss* utilise l'écran interactif comme une zone de contexte où est représentée la collection de signets et l'écran classique comme une zone focus pour la consultation des pages web. Il s'agit d'une application mono utilisateur dans la mesure où un seul utilisateur peut interagir avec le système à un moment donné. En effet, une seule machine contrôle les deux dispositifs d'affichage et, bien que le mur-écran puisse avoir deux dispositifs de pointage (un Mimio^{TM}et une souris), ces deux dispositifs ne contrôlent qu'un seul pointeur et ne peuvent par conséquent pas être utilisés simultanément.

Dans ce chapitre, notre objectif est, dans le même domaine d'application, de proposer une application multi utilisateur. Ainsi, l'écran interactif peut devenir une zone de partage permettant à plusieurs utilisateurs d'échanger et d'organiser leurs collections de signets.

Cependant, l'organisation collaborative des collections de signets est une activité ponctuelle. Il est nécessaire qu'un utilisateur puisse ajouter un signet aisément lorsqu'il est en pleine navigation. Il semble inapproprié de devoir exécuter une application dédiée uniquement pour ajouter un signet à sa collection.

C'est pour cette raison que nous avons mis en place un sous-projets pour atteindre nos objectifs : *Bookies* est un site internet permettant à un utilisateur de gérer une collection de signets et qui propose des fonctionnalités collaboratives basiques.

VI.1 Bookies : constitution d'une base de données

VI.1.1 Objectifs

Bookies[1] a été initialement développé par Joël Randrianandrasana que j'ai co-encadré entre mars et juin 2006 pour son stage de troisième année de licence professionnelle en informatique. Ce projet évolue encore aujourd'hui.

L'objectif principal de ce travail est d'offrir un outil facile d'accès permettant la constitution d'une base de données de signets importante qui servira de jeu de données à l'application *Orchis* qui fera l'objet de la deuxième partie de ce chapitre.

Ce travail a donc consisté en la réalisation d'un site internet permettant à un groupe d'utilisateurs de gérer leurs signets et ce, au travers de deux fonctionnalités principales :

- La gestion des signets personnels via une interface web. Les signets sont ainsi stockés sur un serveur. L'utilisateur peut ainsi y accéder depuis n'importe quelle machine.
- Le partage des signets avec les autres utilisateurs.

Bookies se décompose en deux entités complémentaires et indissociables : un serveur et une interface utilisateur. Le serveur héberge les signets de tous les utilisateurs dans une base de données et l'interface utilisateur permet l'accès à cette base de données de manière conviviale.

La gestion des signets repose sur l'utilisation d'étiquettes – ou *tags* – à la manière de ce qui se fait dans la beaucoup de sites issus du « web 2.0 ».

[1] *Bookies* est accessible à l'adresse http://www.lirmm.fr/edel/bookies/

VI.1.2 Utilisation d'une folksonomie

Il est difficile de donner une définition concise du web 2.0. De manière générale, une application web 2.0 met l'accent sur la collaboration, dispose d'une interface attrayante et utilise des technologies telles que AJAX ou RSS. Le lecteur peut se trouver une définition plus complète sur wikipédia[2].

Comme beaucoup de systèmes issus du web 2.0, nous avons opté pour un système basé sur une *folksonomie*[3]. A l'inverse des taxonomies développées de manière professionnelle à l'aide de vocabulaires contrôlés, les foksonomies ont une approche moins systématique et peuvent même paraître naïves. Cependant, pour les utilisateurs, elles permettent de réaliser une classification facilement et à moindre coût du fait qu'elles ne nécessitent pas d'apprendre une nomenclature hiérarchique compliquée. Des étiquettes sont simplement placées « à la volée » sur les objets manipulés – des signets dans notre application.

De plus, les folksonomies sont, par définition, ouvertes et peuvent s'adapter rapidement aux changements et aux innovations. Ceci est particulièrement important pour un système de signets, lesquels référencent le web qui est un domaine en perpétuelle évolution.

L'intérêt des folksonomies est lié à l'effet communautaire : pour une ressource donnée sa classification est l'union des classifications de cette ressource par les différents contributeurs. Ainsi, partant d'une ressource, et suivant de proche en proche les terminologies des autres contributeurs il est possible d'explorer et de découvrir des ressources connexes [Wikimedia].

Les inconvénients des folksonomies sont justement les problèmes contre lesquels luttent les systèmes de classification formelle. Ceci inclut les polysémies – mots ayant plusieurs sens différents, parfois même opposés (e.g. *hôte* désigne selon le contexte celui qui reçoit ou celui qui est reçu) – et les synonymes – mots différents ayant des sens identiques ou approchés (e.g. pomme de terre et patate).

VI.1.3 Système de recommandations

Bookies propose un système de recommandations qui permet aux utilisateurs du système de partager des signets.

Nous distinguons deux utilisations du système de recommandations. Il est nécessaire de préciser que *Bookies* offre deux vues à une utilisateur : la première est une vue des étiquettes et des signets de l'utilisateur. La seconde est une des étiquettes et des signets des autres utilisateurs, ceux-ci étant représentés dans une arborescence dont le premier niveau contient les noms des différents utilisateurs. Les deux utilisations sont donc :

- Si l'utilisateur navigue dans la vue de ses propres signets ou dans celle des signets des autres utilisateurs, il peut recommander un signet ou une étiquette à un autre utilisateur grâce à un menu contextuel.
- Si l'utilisateur navigue dans la vue des signets des autres utilisateurs, il peut trouver des signets ou des étiquettes qui l'intéresse et qu'il souhaiterait rapatrier dans sa propre collection. Pour cela, il peut recommander un signet ou une étiquette à lui-même.

Les signets rapatriés apparaissent alors dans la catégorie *recommandations/nom_utilisateur/*

[2] http://fr.wikipedia.org/wiki/Web_2.0

[3] Le terme folksonomie est une adaptation française de l'anglais *folksonomy*, combinaison des mots *folk* (le peuple, les gens) et de *taxonomy* (la taxinomie). Certains auteurs utilisent alternativement les termes *potonomie* ou *peuplonomie*.

dans l'arborescence des signets de l'utilisateur, *nom_utilisateur* étant le nom de l'utilisateur ayant recommandé les signets.

VI.1.4 Bilan

Pour ce projet, beaucoup d'attention a été portée à la définition de la base de données ainsi qu'à l'interface utilisateur.

Le schéma de la base de données proposée permet une souplesse maximum pour les utilisateurs (utilisation d'une folksonomie) tout en permettant une certaine hiérarchisation des concepts grâce à l'utilisation d'une arborescence d'étiquettes.

L'interface utilisateur, basée sur les technologies du Web 2.0 et plus particulièrement sur AJAX[4], ne nécessite aucun rafraîchissement de page. L'interface est chargée une seule fois lors de la connexion de l'utilisateur et les données visualisées sont chargées dynamiquement lorsque cela est nécessaire.

A l'heure actuelle, 7 utilisateurs ont adopté *Bookies* et se partagent plus de 800 signets et plus de 150 étiquettes. Cette base d'expertise peut ainsi être partagée lorsqu'un nouveau collaborateur arrive, lui facilitant ainsi son immersion dans un domaine donné.

A l'heure actuelle, le principal défaut de *Bookies* est la mise à disposition d'une unique interface. Il serait en effet bénéfique de proposer des interfaces utilisateur alternatives : une interface basée sur des technologies plus classiques et permettant une compatibilité avec plus de navigateurs. Il serait également intéressant de pouvoir intégrer l'accès à *Bookies* depuis l'interface du navigateur de l'utilisateur (*plugin* ou extension). Notons que *Bookies* est actuellement en phase de réécriture pour résoudre ce défaut.

Au final, le projet *Bookies* est utilisé et permet la constitution d'une base de données importante qui sera utilisée par l'application *Orchis* qui fait l'objet de la section suivante.

VI.2 Orchis : visualisation interactive

VI.2.1 Objectifs

Orchis a été développé pour permettre une manipulation des signets plus interactive que ce qu'offre *Bookies* qui est limité par le fait qu'il s'agit d'une application web. En effet, *Orchis* se base sur l'utilisation de l'API *PoIP* (voir chapitre V) et, par conséquent, autorise des manipulations distribuées.

Les manipulations de type *drag-and-drop* distribuées sont utilisées dans *Orchis* pour permettre à plusieurs utilisateurs de partager ou de s'échanger des signets.

Orchis[5] a été développé par Jérôme Cance, lors d'un stage de deuxième année de master professionnel en informatique. L'un des intérêts principaux de *Orchis*, est de permettre la mise en œuvre de l'API *PoIP* dans une application qui n'est pas seulement une application test. De plus, le fait qu'*Orchis* soit développé par une personne différente de l'auteur de *PoIP* permet de tester les qualités de l'API :

- Est-elle utilisable par une personne qui n'est pas spécialement expérimentée dans l'utilisation du *drag-and-drop* ?
- Est-elle bien documentée ?

[4] Asynchronous JavaScript And XML

[5] b**O**okma**r**k **c**ollaborative exc**h**ange and organ**i**sation **s**ystem

- Est-elle stable ?
- A-t-elle de bonnes performances ?
- A-t-elle une bonne compatibilité (réutilisation de l'existant) ?

Nous répondrons à ces questions après avoir décrit plus précisément l'application *Orchis*.

VI.2.2 Manipulation de graphes de signets

Orchis est une application Java de visualisation et de manipulation de graphes de signets.

Les graphes de signets manipulés avec *Orchis* sont des extraits de la base de données de *Bookies* (base de données MySQL). Comme nous l'avons vu dans la section précédente, *Bookies* gère des signets auxquels sont associés des étiquettes. Il est possible d'assigner un nombre quelconque d'étiquettes à un signet et une hiérarchie d'étiquettes est utilisée. Pour résumer :

- Une étiquette peut avoir une (et une seule) étiquette parent.
- Un signet peut avoir un nombre quelconque d'étiquettes associé

Dans les deux cas, il s'agit de relations orientées. En effet, la relation « *l'étiquette E1 a pour parent l'étiquette E2* » n'est pas équivalente à la relation « *l'étiquette E1 a pour parent l'étiquette E2* » . De même, la relation d'association d'un signet à une étiquette est orientée car les entités associées ne sont pas de la même nature.

Pour visualiser la base de données issue de l'utilisation de *Bookies*, *Orchis* construit un graphe ayant comme sommets les étiquettes et les signets d'un utilisateur et comme arêtes les liens de parenté des étiquettes et les associations étiquette-signet. Il y a donc des sommets de deux natures différentes mais également des arêtes de deux natures différentes.

Définition VI.1 *Un graphe orienté $G = (V, E)$ est défini par un ensemble de sommets V et un ensemble d'arêtes $E \subset V^2$. Tout arc $e_k \in E$ est associé à un couple (ensemble ordonné) de sommets $(v_i, v_j) \in V^2$ où v_i est le sommet de départ et v_j celui d'arrivée.*

Un graphe est dit simple s'il ne contient ni boucles – arête liant un sommet à lui-même –, ni arêtes parallèles – arêtes ayant les mêmes sommets de départ et d'arrivée.

Le graphe construit par *Orchis* est donc un **graphe simple** constitué des sommets $V = T \cup S$ et des arêtes $E = P \cup A$, T étant l'ensemble des étiquettes, S l'ensemble des signets, $P \subset T^2$ l'ensemble des relations de parenté entre les étiquettes et $A \subset S \times T$ l'ensemble des associations entre un signet et une étiquette. De plus, une étiquette ne peut avoir qu'une seule étiquette parent. Donc plus formellement $\forall (e_i, e_k) \in P, \forall (e_j, e_l) \in P, j = i \Rightarrow k = l$.

Du fait q'une étiquette ne peut avoir qu'une seule étiquette parent, le sous graphe partiel $G' = (T, P)$ ne contenant que les sommets de type étiquette et les arêtes de type parenté d'étiquettes est un arbre.

Visualisation Nous avons vu la forme théorique du graphe de signets manipulé par *Orchis*. Nous allons maintenant nous intéresser à la visualisation de ce graphe.

Il faut tout d'abord préciser que l'algorithme de placement des sommets du graphe est un algorithme de placement radial. Cet algorithme est en réalité un algorithme de placement d'arbre mais nous l'avons utilisé sur un arbre couvrant du graphe à placer.

Rappelons que le graphe à visualiser possède deux types de sommets – les étiquettes et les signets – et deux types d'arêtes – les liens de parenté entre étiquettes et les liens d'association signet-étiquette. Comme on peut le voir sur la figure VI.1, les deux types de sommets sont

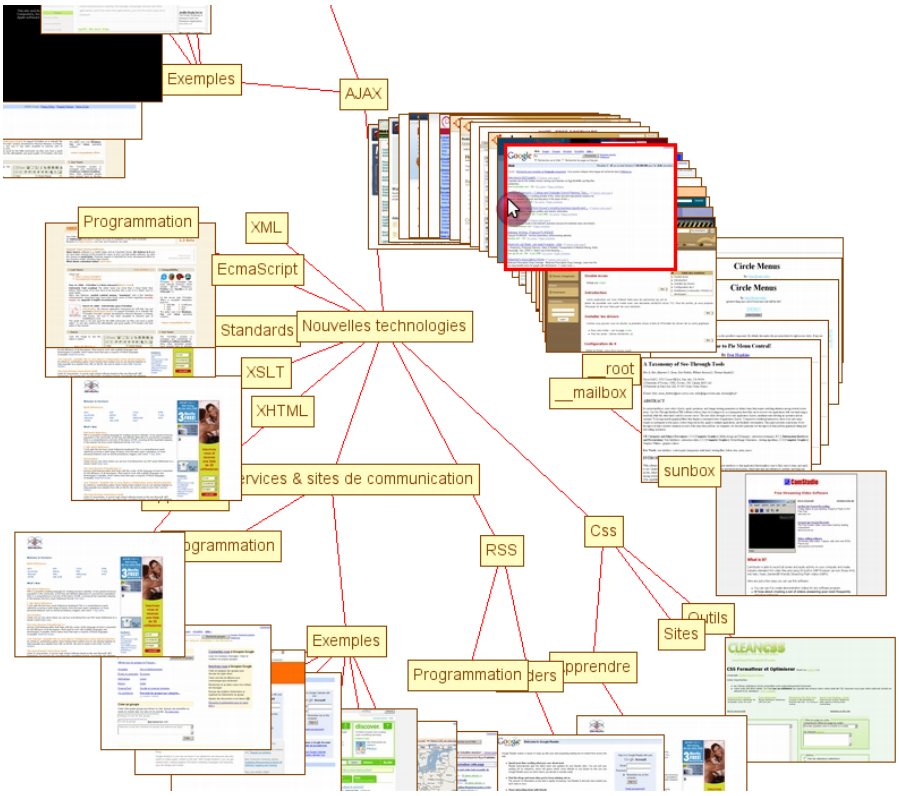

Fig. VI.1: *Capture d'écran d'Orchis.*

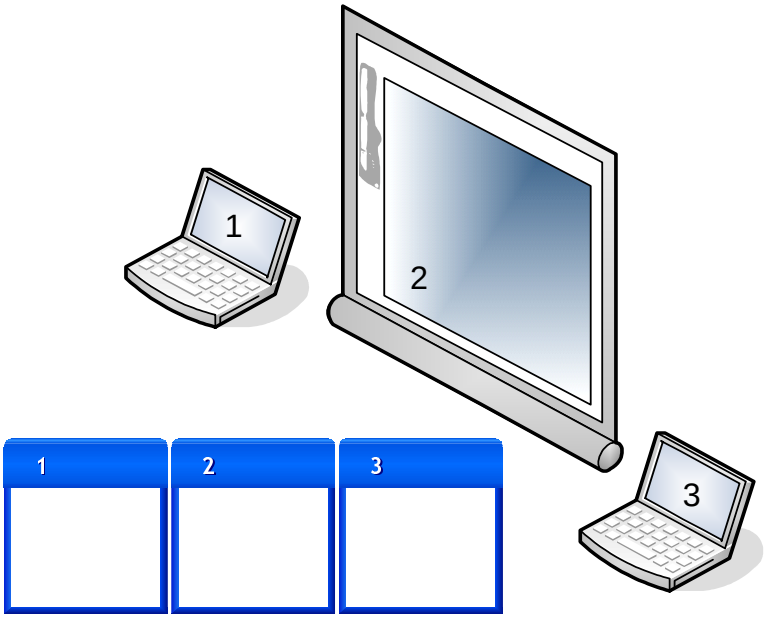

Fig. VI.2: *Agencement des affichages.*

visualisés de manière différentes : du texte pour une étiquette, une vue miniature de la page internet référencée pour un signet.

A l'inverse, les deux types d'arêtes ne disposent pas de représentations différentes. Cela n'est en effet pas nécessaire. D'une part, les arêtes de type association signet-étiquette ne posent aucun problème puisque les deux sommets liés à une telle arête sont de natures différentes et sont visualisés différemment. En ce qui concerne les arêtes de type parenté d'étiquettes, il est aisé pour l'utilisateur de voir leurs orientations grâce à l'algorithme de placement utilisé.

VI.2.3 Orchis, une application collaborative

Dans cette section, nous allons nous intéresser à l'aspect collaboratif d'*Orchis*. Cette section va également nous permettre de voir dans quelle mesure il est simple de mettre en œuvre l'API *PoIP*.

Les signets et les étiquettes des graphes manipulés par *Orchis* supportent les opérations de type *drag-and-drop* introduites par *PoIP*. Nous allons commencer par présenter deux scénarios d'utilisation d'*Orchis*.

Scénarios d'utilisation

Deux utilisateurs, deux machines Dans le premier scénario, deux utilisateurs veulent échanger des signets, chaque utilisateur disposant de son propre ordinateur portable. Les utilisateurs se placent côte à côte, de telle manière que chacun ait les deux écrans dans son champ de vision.

Du point de vue des utilisateurs, chacun lance l'application *Orchis* sur sa machine. Les systèmes de pointages utilisés sont des dispositifs indirects (pavé tactile ou souris) et les pointeurs associés peuvent être déplacés sur l'espace constitué par les deux applications *Orchis*. Les utilisateurs peuvent alors, comme ils le souhaitent, déplacer des signets et/ou des étiquettes d'une vue à l'autre.

On peut distinguer deux types de déplacements :

- L'utilisateur A « donne » un signet à l'utilisateur B en déplaçant un signet par *drag-and-drop* de la surface A (machine de l'utilisateur A) vers la surface B (machine de l'utilisateur B). Et symétriquement.
- L'utilisateur A « prend » un signet à l'utilisateur B en déplaçant un signet par *drag-and-drop* de la surface B vers la surface A. Et symétriquement.

Il faut noter qu'il est tout à fait possible que les deux utilisateurs déplacent des signets ou des étiquettes simultanément, quelque soit les types des déplacements.

Deux utilisateurs, deux machines, un écran interactif Ce deuxième scénario diffère du premier par le fait qu'entre en jeu, en plus des deux machines du premier scénario, un écran interactif de grande taille (type mur-écran, voir p. 3). Cet écran interactif est lié à sa propre unité centrale ; il ne fait pas office d'affichage secondaire pour un des ordinateurs portables. Il dispose par ailleurs d'un dispositif de pointage direct.

Dans ce scénario, il n'est pas nécessaire qu'un utilisateur voie l'écran de l'autre utilisateur. Chaque utilisateur voit son propre écran ainsi que l'écran interactif partagé.

Une instance de l'application *Orchis* est lancée sur chaque machine, soient trois instances : une sur chaque ordinateur portable et une sur l'écran interactif. Ces trois instances forment un espace de travail continu dont on peut voir l'agencement sur la figure VI.2. L'utilisateur A

(affichage 1) peut déplacer son pointeur vers l'écran interactif en se dirigeant vers la droite et l'utilisateur B (affichage 3) peut faire de même en dirigeant son pointeur vers la gauche.

Dans cette configuration, il est possible de faire les mêmes échanges que dans le scénario précédent. Cependant, cette fois, un utilisateur n'accède pas directement au signet de l'autre utilisateur. L'écran interactif sert de zone tampon et un transfert de signet (ou d'étiquette) se fait en deux temps :

- Dans un premier temps, l'utilisateur A déplace un signet de sa collection vers l'écran interactif.
- Dans un second temps, l'utilisateur B déplace ce même signet de l'écran interactif vers sa propre collection de signets.

L'avantage de l'utilisation d'une zone tampon est qu'un utilisateur n'a pas directement accès à la collection de signets et d'étiquettes d'un autre utilisateur. Ainsi, il n'est pas amené à voir les signets privés d'un autre utilisateur et il ne peut pas modifier sa collection (e.g. associé un signet à une étiquette).

Au final, ce scénario d'utilisation supporte beaucoup plus facilement des utilisateurs supplémentaires que le scénario précédent. En effet, dans le premier scénario, le fait que l'utilisateur A puisse voir l'écran de l'utilisateur B est réaliste mais ceci devient beaucoup plus difficile si quatre utilisateurs doivent chacun être en mesure de voir quatre écrans. Alors que dans le second scénario, un utilisateur n'a besoin de voir que son propre écran ainsi que l'écran interactif partagé, ce qui est nettement plus réaliste.

De plus, imaginons qu'un utilisateur veuille donner un signet à tous les autres utilisateurs. En adaptant le premier scénario, il lui est nécessaire de faire autant d'opérations qu'il y a d'utilisateurs (autre que lui-même). Alors qu'en adaptant le second scénario, il suffit à l'utilisateur voulant donner un signet de le déposer sur la surface partagée. Tous les autres utilisateurs peuvent alors rapatrier le signet en question dans leur propre collection.

VI.2.4 Implémentation

Nous allons maintenant voir comment est implémenté *Orchis* en nous focalisant sur ce qui nous intéresse principalement, à savoir l'utilisation de l'API *PoIP*.

Création de l'interface utilisateur

Lors de la création de la *JFrame* principale de l'interface utilisateur, un objet de type *SharedSurface* est crée :

```
sharedSurface = new SharedSurface(this,
                    SharedSurface.INDIRECT_POINTING);
```

Comme nous l'avons déjà expliqué dans le chapitre V, il est possible de spécifier si le dispositif de pointage utilisé est direct ou indirect. Dans l'exemple ci-dessus, un dispositif indirect est utilisé, ce qui signifie que le pointeur système sera contrôlé par un *Robot* est qu'il sera possible de déplacer le pointeur correspondant à cette surface sur les autres surfaces partagées.

Enregistrement des cibles et des sources potentielles

Lors de la création de la vue du graphe de signets, cet objet s'enregistre en tant que source et cible potentielle pour les opérations de type *drag-and-drop* :

```
frame.getSharedSurface().getDmManager().registerSource(
  this, listener);
frame.getSharedSurface().getDmManager().registerTarget(
  this, listener);
```

Pour ces enregistrements, il est nécessaire de préciser le composant source ou cible ainsi que l'écouteur associé. Dans l'exemple ci-dessus, l'objet *this* est la vue du graphe (instance d'une classe hgéritant de *JPanel*) et l'objet *listener* est instance d'une classe implémentant les interfaces *IDmTargetListener* et *IDmSourceListener*.

Nous allons maintenant nous intéresser au fonctionnement interne de cet objet *listener*.

Fonctionnement interne des écouteurs

Nous ne présentons ici qu'un exemple du fonctionnement interne des écouteurs : nous allons voir comment l'écouteur gère les données à transférer lors d'une opération de type *drag-and-drop*.

Dans un premier lieu, voyons la déclaration des objets nécessaires au stockage des données pour un composant source :

```
private Hashtable<Integer, Component> currentData =
  new Hashtable<Integer, Component>();
```

On utilise ici une collection de données, les données étant associées à l'identifiant du pointeur qui les manipule. Ainsi, plusieurs pointeurs peuvent manipuler des données différentes simultanément.

De la même manière, on maintient une collection des objets se trouvant sous les différents pointeurs :

```
private Hashtable<Integer, Component> hightlights =
  new Hashtable<Integer, Component>();
```

Lorsque qu'une manipulation de type *drag-and-drop* est détectée, la méthode *dmDetected()* de l'écouteur est appelée :

```
public void dmDetected(IDmSourceEvent sourceEvent) {
  try {
    if (hightlights.get(sourceEvent.getDeviceID()) != null) {

      currentData.put(sourceEvent.getDeviceID(),
        hightlights.get(sourceEvent.getDeviceID()));

      //... commandes supprimees pour faciliter la
      // lisibilite de ce listing

      sourceEvent.startDm(Component.class.getCanonicalName());
    }
  } catch (RemoteException e) {
```

```
      e.printStackTrace();
    }
  }
```

L'objet se trouvant sous le pointeur identifié dans l'évènement est stocké dans la collection de données (*currentData*). Puis l'opération de type *drag-and-drop* est validée par l'appel de la méthode *startDm()* en spécifiant le type des données manipulées.

A la fin de l'opération, lorsque les données doivent être transférées à la cible, les données stockées localement sont renvoyées, toujours en fonction de l'identifiant du pointeur :

```
public void dmDataRequired(IDmSourceEvent sourceEvent) {
  try {
    //... commandes supprimees pour faciliter la
    // lisibilite de ce listing

    sourceEvent.setData(currentData.get(
      sourceEvent.getDeviceID()));
  } catch (RemoteException e) {
    e.printStackTrace();
  }
}
```

L'exemple présenté dans cette section a pour objectif d'illustrer le support des manipulations concurrentes. Là où, dans une application classique ne supportant pas le multiplexage des entrées, un seul objet est utilisé, il est souvent nécessaire de gérer une collection d'objet pour permettre le support du multiplexage des entrées.

On se rend ainsi compte que les modifications nécessaires pour qu'une application existante supporte les pointeurs multiples ne sont pas négligeables. En plus de rendre le fonctionnement de cette application plus complexe, ces modifications peuvent s'avérer très fastidieuses.

VI.2.5 Bilan

Faisons maintenant le bilan sur le développement d'*Orchis* du point de vue de l'utilisation de *PoIP*. Rappelons que *Orchis* a été développé par Jérôme Cance, étudiant en Master 2 Pro.

Complexité Premièrement, nous pouvons affirmer que l'API *PoIP* est facilement utilisable par tout développeur. Du point de vue du modèle événementiel des techniques de type *drag-and-drop*, elle est d'une complexité comparable au modèle de *drag-and-drop* par défaut de Java, voire un peu moins complexe. Cependant, la grosse difficulté introduite par *PoIP* est la gestion des manipulations concurrentes. Il est en effet possible que deux utilisateurs effectuent simultanément des opérations de type *drag-and-drop*. Or, il convient que les composants sources et cibles des opérations de type *drag-and-drop* gèrent ces opérations concurrentes, c'est-à-dire qu'ils supportent le multiplexage des entrées (voir chapitre I).

Par exemple, lorsqu'un utilisateur fait un *drag-and-drop* sur un sommet du graphe, le composant responsable de l'affichage du graphe (qui est une source et une cible possible pour les opérations de type *drag-and-drop*) mémorise le sommet en cours de manipulation en fonction des coordonnées du pointeur au début de l'opération. Si deux *drag-and-drop* sont effectués simultanément par deux utilisateurs à partir du même graphe, il faut mémoriser deux sommets

sources et non plus un seul, chacun étant associé à un dispositif de pointage, ou plus exactement à son identifiant.

D'une manière générale, un composant qui est une source ou une cible potentielle pour une opération de type *drag-and-drop* doit, pour chaque information qu'il mémorise au cours de l'opération, gérer une table de hachage dont les clés sont les identifiants des dispositifs de pointage au lieu de gérer une simple variable. Bien que le concept soit relativement simple, cette contrainte complique le développement.

Stabilité La stabilité de l'API n'a pas été remise en cause par le développement de *Orchis*. Nous avons observé des défauts que nous avions déjà rencontrés lors du développement d'applications tests. Le plus représentatif étant le comportement parfois erratique du *Robot* qui est utilisé pour contrôler le pointeur système. Cependant, le fonctionnement interne de *PoIP* n'a pas montré de faiblesses.

Compatibilité *PoIP* est une API supportant les pointeurs multiples. Elle utilise des évènements augmentés : les évènements Java auxquels nous avons ajouté un identifiant permettant de savoir quel dispositif de pointage est à l'origine de l'évènement. Si deux utilisateurs interagissent avec le même composant, il est donc nécessaire que celui-ci prenne en compte l'identifiant du dispositif de pointage afin de se comporter correctement.

Nous avons augmenté les évènements Java plutôt que de créer de nouveaux types d'évènements en partie pour que l'API fonctionne avec les composants existants dans la mesure où plusieurs utilisateurs n'interagissent pas simultanément avec le même composant. Cependant, et comme nous aurions pu nous y attendre, force est de constater que le support des composants existants est limité. En effet, un certain nombre de composants simples fonctionnent parfaitement avec l'API (citons par exemple *JButton* ou *JTextField*) mais les composants plus complexes présentent un problème de plasticité et ne fonctionnent pas complètement. C'est par exemple des listes à choix multiples – *JComboBox* – qui utilise plusieurs sous-composants et qui ne place pas la liste déroulante au même niveau que les autres éléments de l'interface utilisateur. De ce fait, *PoIP* ne peut envoyer les évènements d'entrée à la liste déroulante.

D'autre part, il faut mentionner que, pour l'affichage des graphes d'*Orchis*, nous avions dans un premier temps opté pour l'utilisation d'une *toolkit* de visualisation de l'information existante, à savoir *Prefuse*[6]. Cependant, nous nous sommes retrouvés confrontés à un problème de compatibilité du au fait que *Prefuse* effectue par défaut un certain nombre de traitements sur les évènements d'entrée. Par exemple, *Prefuse* gère automatiquement la mise en surbrillance du sommet du graphe se trouvant sous un pointeur, l'utilisation simultanée de plusieurs pointeurs provoquant des problèmes d'affichage et des incohérences. Ne voulant pas modifier en profondeur les composants de *Prefuse*, Jérôme Cance a finalement écrit son propre composant pour l'affichage de graphe.

Ce manque de compatibilité est représentatif des difficultés de mettre au point un système supportant les pointeurs multiples. Le paradigme du pointeur unique (et du focus unique) est implanté très profondément dans les systèmes actuels et le travail est considérable pour transformer un système de manière à ce qu'il supporte les dispositifs de saisie multiples.

[6] http://prefuse.org/

Performances Du point de vue des performances, *PoIP* a été à la hauteur de nos espérances. Malgré le fait que *PoIP* dessine les pointeurs lui-même au dessus de l'espace de travail des utilisateurs, nous n'avons pas constaté de baisse des performances importante. L'affichage des *feedbacks* des instruments (techniques de type *drag-and-drop*) n'a pas eu non plus de conséquences importantes sur les performances globales de l'application.

CONCLUSION

Sommaire

Ce mémoire a pour objectif l'étude des interactions qui interviennent dans les environnements où plusieurs surfaces d'affichage et dispositifs de pointage sont mis à disposition du ou des utilisateurs, que ces environnements soient distribués ou non. Plus particulièrement, nous avons utilisé le mur-écran dans le cadre de nos recherches.

1 Contexte multi surface

Dans un premier temps, nous nous sommes intéressés au contexte multi-surface. Dans le chapitre I, nous avons dressé un état de l'art des systèmes permettant le partage des dispositifs d'entrée et de sortie.

Le partage des dispositifs d'entrée intervient lorsqu'un dispositif de pointage est utilisé sur une surface avec laquelle il n'est pas directement lié (unités centrales différentes) ou lorsque plusieurs systèmes de pointage sont utilisés simultanément sur la même surface.

Le partage des dispositifs d'affichage peut se manifester par l'utilisation de plusieurs surfaces d'affichage n'étant pas contrôlées par la même machine, ou encore par l'affichage de l'interface d'un programme sur une surface d'affichage distante, etc.

Pour caractériser les systèmes évoqués dans le chapitre I, nous avons été amenés à définir des critères permettant d'identifier les mécanismes mis en place par les différents systèmes. Nous avons identifié cinq mécanismes :

- La redirection des entrées
- Le multiplexage des entrées
- La redirection des sorties
- L'intégration des sorties
- La séparation des sorties

Par la suite, nous nous sommes attardés sur les techniques d'interaction dans ce contexte multi surface.

2 Modèle d'interaction et expérimentations

Dans le chapitre II, nous avons décrit un ensemble de techniques destinées à étendre les capacités du *drag-and-drop*. Et, à partir de là, nous nous sommes focalisés sur les performances et l'implémentation de cet ensemble de techniques d'interaction.

L'étude des performances de ces techniques – chapitre IV – nous a amené à modifier certaines techniques existantes et à en introduire 2 nouvelles qui se sont avérées très performantes lorsqu'elles étaient utilisées sur des surfaces augmentées. Nous avons, dans une première étude, évalué les performances des méthodes de lancer avec un dispositif de pointage indirect. Nous avons amélioré le *drag-and-throw* en permettant à l'utilisateur d'avoir une précision constante, ce qui était impossible dans les versions antérieures. Malgré l'utilisation d'un dispositif de pointage indirect qui favorisait le *drag-and-drop*, les performances des méthodes de lancer ont été encourageantes et nous avons réalisé une seconde étude.

La seconde étude a été beaucoup plus complète et a été réalisée dans des environnements multi surface augmentés. Un total de six techniques d'interaction de type *drag-and-drop* ont été évaluées dans cette étude, dont deux nouvelles : le *push-and-pop* et le *push-and-throw* accéléré. Les résultats de l'étude ont été très bons puisque les deux nouvelles techniques que nous avons proposées ont été plébiscitées : Le *push-and-pop* a été le plus performant et le

push-and-throw accéléré a obtenu les meilleurs performances parmi les techniques qui offraient le plus de possibilité (cf. exemple du réarrangement de bureau).

3 Modèle d'implémentation et modèle d'interaction instrumentale pour le drag-and-drop et ses évolutions

En réalisant ces études, nous avons été amenés à nous interroger sur l'implémentation du *drag-and-drop* et plus généralement des techniques de type *drag-and-drop*. En étudiant les implémentations du *drag-and-drop* dans plusieurs systèmes largement utilisés, nous avons constaté des différences parfois importantes entre les différentes implémentations (première partie du chapitre III). De plus, les différentes extensions du *drag-and-drop* n'ont jamais fait l'objet d'attentions particulières quant à leur implémentation, seul leur aspect novateur en terme d'interaction intéressait les chercheurs.

A partir de ces observations, nous nous sommes appliqués à définir de manière générale le modèle d'interaction du *drag-and-drop* mais aussi plus généralement de toutes les techniques de type *drag-and-drop*. Nous nous sommes reposés pour cela sur le modèle d'interaction instrumentale, particulièrement adapté à ces techniques d'interaction. Le modèle d'interaction instrumentale est un modèle conceptuel qui peut être utilisé comme base pour un modèle d'implémentation.

Les avantages principaux du modèle d'implémentation que nous avons proposé sont :

- Ouverture et modularité pour le développeur : peu d'efforts sont nécessaires pour développer une nouvelle technique de type *drag-and-drop*. De plus, le développeur qui veut simplement activer le support des techniques existantes dans ses interfaces n'a affaire qu'à un nombre limité d'entités alors que certaines implémentations du *drag-and-drop* sont très compliquées (voir chapitre III).
- Modularité pour l'utilisateur : il a la possibilité de choisir sa technique préférée. De la même que les utilisateurs choisissent leur environnement graphique (gnome, KDE, fvwm, etc.) dans le monde Unix, il devient possible de choisir sa technique de type *drag-and-drop* préférée. Celle-ci dépendra cependant plus de l'environnement matériel que des préférences personnelles de l'utilisateur.
- Et surtout le support des environnements distribués. Bien que le support des environnements distribués dépende plus de l'implémentation qui est faite du modèle que du modèle lui-même, celui-ci a été pensé avec l'optique d'une implémentation distribuée.

Ce modèle, de par sa nature, apporte plus de simplicité et de cohérence. Un maximum de propriétés est fixé *a priori* dans la gestion de l'interaction par le système. Ainsi, les composants de l'interface qui entrent en jeu dans l'interaction ont une marge de manœuvre moindre – ils ont moins de possibilité de « personnaliser » l'interaction – mais le résultat est plus cohérent mais également plus simple pour le développeur. En effet, un reproche que l'on peut faire au *drag-and-drop* est qu'il n'apporte pas au développeur la même simplification d'utilisation qu'à l'utilisateur final. Notre modèle a donc le mérite, au prix du sacrifice de quelques fonctionnalités annexes, de simplifier la gestion des opérations de type *drag-and-drop*.

De plus, la simplicité du modèle n'est pas remise en question par le support des environnements distribués, qui est complètement transparent pour le développeur qui désire activer les opérations de type *drag-and-drop*. D'autre part, un mécanisme de communication est intégré pour faciliter la tâche du développeur qui désire implémenter de nouvelles techniques

d'interaction de type *drag-and-drop*.

4 Réalisation

L'API *PoIP*, qui implémente le modèle du chapitre III, est présentée dans le chapitre V.

Selon les critères définis dans le chapitre I, *PoIP* met en place la redirection et le multiplexage des évènements d'entrée. D'autre part, la gestion des évènements de saisie par *PoIP* se fait avec une granularité moyenne : les évènements sont abstraits et augmentés afin de permettre l'identification du dispositif à l'origine de l'évènement.

	Entrées		Sorties		
	Redirection	Multiplexage	Redirection	Intégration	séparation
PoIP	✓	✓			

5 Perspectives

Les développements de *PoIP* et d'*Orchis* nous a ouvert quelques perspectives intéressantes au niveau du partage des surfaces est de la gestion de la topologie et de la gestion du focus.

5.1 Partage

Pour gérer le partage des surfaces, *PoIP* agit au niveau de la machine virtuelle Java. Nous n'avions pas les moyens de modifier un système de fenêtrage. Notre système permet donc de partager une surface qui est en réalité la surface occupée par un composant (classe *Component*). Il serait intéressant de s'interroger sur la granularité des surfaces partagées. Il pourrait par exemple être possible de partager un ensemble de fenêtres, ou une partie de l'espace de travail, indépendamment des fenêtres se trouvant dans cette espace.

D'autre part, il serait également intéressant d'offrir plus de contrôle aux utilisateurs sur le partage des surfaces. Ainsi, un utilisateur pourrait choisir les autres utilisateurs qui auront accès à son espace partagé. Il se peut aussi qu'un utilisateur veuillent accéder à un espace partagé sans pour autant partager son propre espace de travail.

5.2 Topologie

Nous n'avons proposé qu'une réponse simple à la problématique de la topologie. Nous avons opté pour un système où les utilisateurs spécifient eux-mêmes la configuration des différents affichages, à la manière de ce qu'il se passe dans Synergy [Schoeneman] ou PointRight [Johanson et al., 2002b]. Une autre approche consiste à concevoir un système de détection des positions des affichages. Ainsi, la topologie est entièrement gérée par le système, à la manière de ce que l'on peut trouver dans les projets *Augmented Surfaces* [Rekimoto and Saitoh, 1999] ou I-AM [Lachenal, 2004]. Cependant, un tel système est peut s'avérer lourd et coûteux à mettre en place. D'autre part, lorsqu'il est très difficile de déterminer une topologie des affichages, il nous paraît préférable que l'utilisateur la spécifie, ce qui évitera qu'il ne comprenne pas la topologie utilisée par le système.

D'autre part, une idée intéressante serait de spécifier une topologie locale. Par exemple, dans le second scénario de la section VI.2.3, qui met en scène deux utilisateurs, leurs machines

portables et un mur interactif, deux topologies simples pourraient être gérées. Au lieu de gérer une topologie dans laquelle sont placés les trois affichages, il serait intéressant que chaque utilisateur spécifie sa propre topologie par rapport au mur interactif. Cette idée rejoint l'idée d'une gestion plus fine du partage des surfaces. En effet, avec deux topologies distinctes, chacune associé à un utilisateur (ou à son dispositif de saisie), un utilisateur n'a plus accès qu'à sa machine et au mur interactif partagé. Et il n'a plus accès à la machine du second utilisateur.

5.3 Gestion du Focus

Un point qui mériterait plus d'attention est la gestion du focus. Dans l'implémentation que nous avons faite de *PoIP*, nous voulions avoir la possibilité de réutiliser l'existant, c'est à dire de pouvoir utiliser les composants graphiques de l'API de Java. Or, le fonctionnement de l'API Java, comme la plupart des interfaces graphiques utilisées à grande échelle, repose en partie sur le postulat suivant : à un moment donné, un unique composant possède le « focus » et est donc susceptible de recevoir les évènements du clavier. Or, ceci est totalement incompatible avec l'utilisation simultanée de plusieurs dispositifs de saisie par différents utilisateurs. Dans ce contexte, nous avons opté pour une solution simple : nous supposons que chaque clavier et couplé à un dispositif de pointage et nous redirigeons les évènements du clavier vers le composant se trouvant sous le pointeur du dispositif de pointage couplé. Cependant, cette solution n'a pas fait l'objet de réflexions plus approfondies et mériterait d'être étudiée plus en profondeur et éventuellement d'être améliorée.

5.4 Déploiement

D'autre part, une perspective pour le modèle présenté dans ce mémoire est d'être implémenté dans un contexte plus professionnel. Bien que nous ayons apporté le plus d'attention possible aux problématiques d'implémentation, le travail présenté ici reste malgré tout un travail de recherche avec les moyens correspondant. Nous espérons donc que ce modèle sera à terme implémenté de manière durable dans les systèmes courants ou, de manière plus réaliste, que notre modèle inspirera une implémentation plus ouverte et modulaire du *drag-and-drop*.

5.5 Redirection, intégration et séparation des sorties

Enfin, la suite du travail présenté dans ce mémoire et qui se traduit par l'application de la redirection et du multiplexage des évènements d'entrée est de s'intéresser à la redirection, à l'intégration et à la séparation des sorties. En effet, l'aboutissement de ce mémoire est la présentation d'un système – chapitres V et VI – où les pointeurs sont totalement libres d'évoluer sur un espace de travail constitué de différents affichages ayant pour seul point commun de s'être enregistrés auprès d'un serveur de surfaces partagées. Il est même possible de déplacer des objets par *drag-and-drop* (ou toute autre méthode du même type) dans cet espace de travail. L'étape naturelle qui suit est de permettre aux utilisateurs de déplacer des applications ou plus précisément leurs interfaces utilisateur d'une surface à l'autre. Il n'y aurait alors plus de limite à l'utilisation des surfaces partagées. L'espace de travail complet formé par les différentes surfaces partagée offrirait exactement les mêmes fonctionnalités que les espaces de travail dont nous avons l'habitude aujourd'hui.

ANNEXE A
ACTIVATION À DISTANCE

Dans ce mémoire, et plus particulièrement dans les chapitres II et IV, nous nous sommes concentrés sur la possibilité de déplacer un objet vers une destination difficilement accessible. Au final, deux méthodes sont sorties du lot : le *push-and-pop* et le *push-and-throw* accéléré.

Cette annexe présente une réflexion sur l'extension de ces techniques afin qu'elles puissent également être utilisées dans d'autres conditions. En particulier pour l'activation à distance. En effet, grâces à ces techniques de type *drag-and-drop*, il est aisé de déplacer un objet vers des zones difficiles d'accès. Mais elles ne permettent pas de déplacer des objets se trouvant initialement dans ces mêmes zones. Un simple clic sur un objet se trouvant dans ces mêmes zones n'est pas non plus possible.

Pour commencer cette réflexion, nous présenterons les techniques existantes permettant l'activation à distance puis nous explorerons les possibilités d'adaptation du *push-and-pop* et du *push-and-throw* accéléré.

1 Les alternatives

1.1 Le drag-and-pick

Le *drag-and-pick* [Baudisch et al., 2003] a été proposé parallèlement au *drag-and-pop* (voir section II.2.4, p. 41). Ces deux techniques aident l'utilisateur à manipuler des icônes. Le *drag-and-pick* permet de double-cliquer sur une icône distante. Pour cela, un groupe d'icônes est temporairement dupliqué auprès du pointeur (Figure A.1). Ce groupe d'icônes est déterminé en fonction de la direction du mouvement d'activation. L'utilisateur peut ensuite sélectionner l'icône sur laquelle il désire double-cliquer.

Le *drag-and-pick* répond dans une certaine mesure au problème que nous essayons de résoudre dans cet article, mais il présente l'inconvénient de ne prendre en compte que les icônes.

1.2 Frisbee

Frisbee [Khan et al., 2004] est un système permettant d'interagir avec une zone distante de l'espace de travail. C'est une fenêtre sur une zone distante cible (Figure A.2).

Ce système est adapté à une interaction d'une certaine durée avec la zone cible. Mais il est trop long à mettre en place pour une interaction simple telle q'un clic.

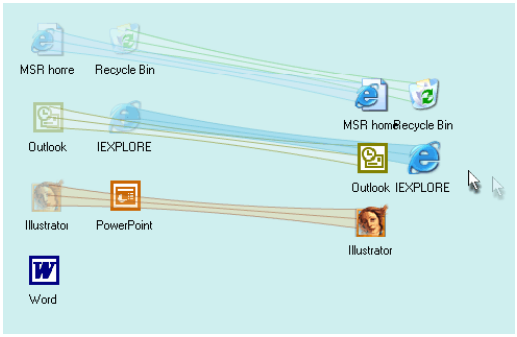

Fig. A.1: *drag-and-pick : les cibles potentielles s'approchent du pointeur [Baudisch et al., 2003].*

Fig. A.2: *Le système frisbee contient un télescope et une cible. En interagissant à l'intérieur du télescope, l'utilisateur interagit avec la cible [Khan et al., 2004].*

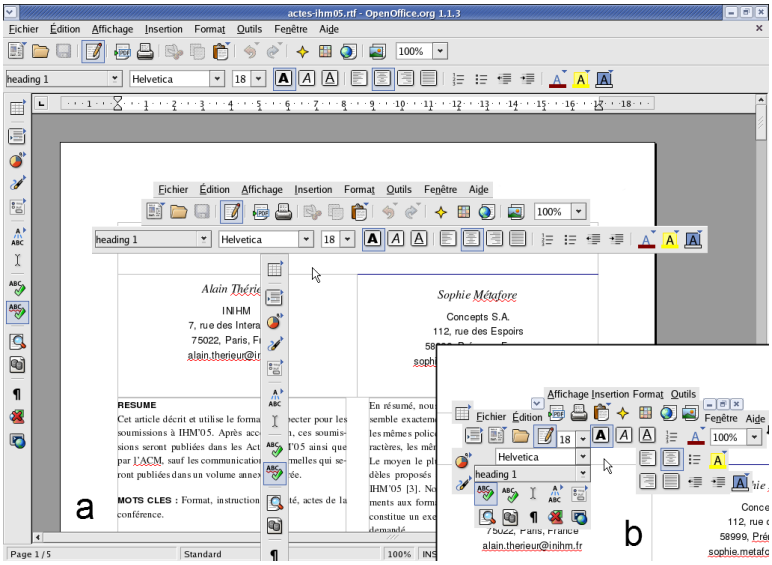

Fig. A.3: *Deux exemples de cibles se rapprochant du pointeur dans un traitement de texte.*

1.3 TractorBeam

TractorBeam [Parker et al., 2005] est une technique d'interaction destinée aux tables augmentées. Elle permet de passer d'une manière transparente d'un système de pointage direct (toucher) pour des objets proches de l'utilisateur à un système de pointage distant pour des objets plus lointains (Figure A.3).

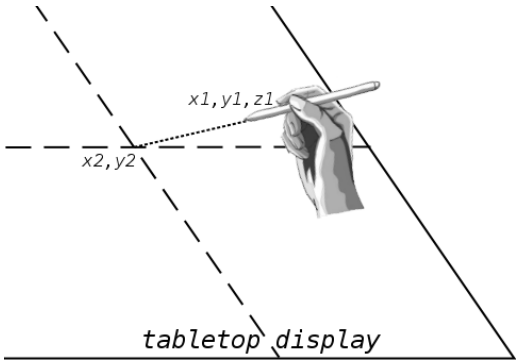

Fig. A.4: *Pointer avec le TractorBeam. Le système est capable de projeter les coordonnées du stylo dans l'espace (x1, y1, z1) sur la table (x2, y2).*

L'inconvénient de cette technique est la nécessité d'un matériel spécifique (un stylo dont on puisse calculer la direction de pointage).

2 Pointer à distance

Comme nous l'avons vu dans le chapitre III, il existe deux approches opposées pour les techniques de types *drag-and-drop*. La première consiste à agir sur la cible du pointage pour la rapprocher du pointeur (cible-vers-pointeur) et la seconde à éloigner le pointeur pour le rapprocher de la cible (pointeur-vers-cible).

2.1 Cible-vers-pointeur

Parmi les techniques de type *drag-and-drop* utilisant l'approche cible-vers-pointeur, on trouve le *drag-and-pop* et le *push-and-pop*.

L'adaptation de cette approche pour l'activation à distance a déjà été proposée avec le *drag-and-pick* (Figure A.1). Cependant, cette technique n'a été ni implémentée, ni testée, et elle se restreint à la manipulation d'icônes.

Généralisation du drag-and-pick

Même si le drag-and-pick semble utilisable dans le cas de la manipulation d'icônes, sa généralisation à l'ensemble des composants que l'on trouve aujourd'hui dans les interfaces graphiques pose problème. En effet, comme on peut le voir sur la figure A.4–a, les barres d'outils, qui sont présentes dans la plupart des logiciels, ont des formes difficiles à dupliquer auprès du pointeur. En effet, on se retrouve limité au niveau des déplacements du fait qu'elles utilisent souvent toute la largeur ou toute la hauteur de l'écran. Une solution pourrait être de ne plus

considérer les barres d'outils mais les boutons qui les composent lors de la duplication des cibles (figure A.4–b). Un autre inconvénient majeur apparaît alors : la topologie de l'espace de travail n'est plus respectée et les positions des composants sont difficilement prévisibles. L'utilisateur doit alors chercher le bouton sur lequel il veut cliquer parmi un amas de composants.

La sélection des cibles

L'approche cible-vers-pointeur présente encore une limite importante : pour pouvoir dupliquer les cibles potentielles auprès du pointeur, il faut connaître toutes les cibles potentielles pour une activation à distance. En effet, dans l'état actuel des choses, il n'est pas possible de détecter les différents composants se trouvant dans une fenêtre. Ceci est dû à la diversité des boites à outils de composants utilisées. C'est particulièrement difficile avec les boites à outils « légères » (e.g. *Java/Swing*).

Pour les techniques de type *drag-and-drop*, il est également nécessaire que les cibles s'enregistrent avant de pouvoir recevoir un objet. Cette étape d'enregistrement est cependant acceptable étant donné qu'elle est nécessaire pour toutes les techniques de type *drag-and-drop*, sans exception.

Cette contrainte est cependant beaucoup plus problématique pour la simple activation d'un composant (e.g. un clic sur un bouton). Il faudrait donc que chaque composant acceptant les clics s'enregistre auprès du système. Ceci nécessiterait d'agir au niveau des logiciels existant et demanderait un nombre de modifications important.

Des inconvénients rédhibitoires

Ces inconvénients sont très gênants. Il serait frustrant de ne pouvoir utiliser une telle technique qu'avec certains composants : ceux des logiciels ayant été prévus pour supporter cette technique. De plus, ce serait indubitablement un frein à l'utilisation de cette technique. Il nous apparaît donc nécessaire de proposer une technique d'interaction qui puisse être utilisée sans modifier les bases logicielles existantes. Nous pensons que la solution se trouve au niveau des approches pointeur-vers-cible.

2.2 Pointeur-vers-cible

Parmi les techniques de type *drag-and-drop*, une approche pointeur-vers-cible est utilisée par des techniques d'interaction telles que le *push-and-throw* et ses variantes.

Les limites de cette approche

Le fait de déporter le pointeur de l'utilisateur est contraignant pour l'utilisateur. D'une part, il est obligé d'avoir le pointeur déporté dans son champ de vision. Le domaine d'utilisation des techniques utilisant cette approche s'en trouve donc limité. D'autre part, durant l'opération de pointage, l'utilisateur doit constamment ajuster son mouvement en fonction des retours visuels dont il dispose. Ce réajustement permanent du geste de l'utilisateur représente donc une charge supplémentaire pour lui. nous avons vu dans le chapitre IV qu'en terme de temps de complétion des tâches élémentaires, cette approche conduisait à des performances moindres que l'approche cible-vers-pointeur.

Déporter le pointeur

Malgré ces limitations, l'approche pointeur-vers-cible présente des avantages déterminants par rapport à ses alternatives. D'une part, elle est possible là où les techniques cible-vers-pointeur ne le sont plus : pour la sélection dans un menu par exemple. D'autre part, elle ne dénature pas l'arrangement spatial du bureau ou des applications : les objets restent à leur place. L'utilisateur peut s'appuyer sur l'utilisation de la mémoire spatiale alors qu'avec les techniques cible-vers-pointeur, les cibles potentielles sont réarrangées. Cette caractéristique a une importance en terme de confort d'utilisation plus qu'en terme de performance. En effet, le déplacement et réarrangement des objets, qui est inévitable avec les techniques de pointage cible-vers-pointeur, a un effet perturbant sur l'utilisateur mais il ne compromet pas son efficacité (voir chapitre IV). Enfin, il est relativement aisé de mettre en place une technique d'interaction utilisant cette approche en toute transparence : les applications existantes peuvent être compatibles avec ce mode de pointage sans modification.

3 Le drag-and-click

Nous proposons donc une technique utilisant une approche pointeur-vers-cible. Elle permet de remplacer un clic sur un objet distant par un petit *drag-and-drop*. On nommera cette technique *drag-and-click*.

Le *drag-and-click* consiste en réalité à transformer un système de pointage direct en un système de pointage indirect. Ainsi tout se passe de la même manière que lors de l'utilisation d'une souris : l'utilisateur déplace la souris en réajustant en permanence son mouvement en fonction de la position du pointeur à l'écran.

Comme on peut le voir sur la figure A.5, le pointeur déporté est lié au pointeur original par une bande élastique de type « copie temporaire » (voir la section IV.2.1, p. 94).

Détaillons maintenant la manière dont le pointeur déporté se déplace en fonction des mouvements du pointeur original. Nous déterminerons ensuite comment sera activée cette nouvelle technique.

3.1 Précision

Dans le cadre de l'activation d'un composant à distance, il est nécessaire de disposer d'une précision comparable à celle dont on peut disposer lorsqu'on utilise un pointage classique. En effet, il n'est pas rare de devoir cliquer sur des boutons de petite taille (e.g. une vingtaine de pixels de côté pour les boutons d'une barre d'outils).

Parmi les techniques de type *drag-and-drop*, l'approche pointeur-vers-cible est utilisée par le *drag-and-throw*, le *push-and-throw* et le *push-and-throw* accéléré. Le *push-and-throw* et le *drag-and-throw* présentent un manque de précision du fait qu'ils se contentent d'amplifier de manière linéaire les mouvements de l'utilisateur.

Une variante, le *push-and-throw* accéléré, a été proposée pour offrir une plus grande précision à l'utilisateur (voir section IV.2.1, p. 90). Ainsi, l'amplification du mouvement n'est plus faite linéairement mais de manière plus progressive : un mouvement lent du pointeur original ne sera pas amplifié alors qu'un mouvement rapide du pointeur original se traduira par un mouvement très rapide du pointeur déporté. Cette accélération variable est comparable à ce

qui se fait pour les souris : il est possible de parcourir rapidement de grandes distances tout en conservant une bonne précision pour les petits déplacements.

Le facteur d'accélération du *push-and-throw* accéléré dépend également de la position du pointeur déporté : s'il se trouve sur une cible potentielle, alors le facteur d'accélération est moindre que si le pointeur se trouve sur un espace « vide ». Ce comportement est issu du pointage sémantique [Blanch et al., 2004] mais ne peut pas être utilisé dans le cas qui nous intéresse pour les mêmes raisons qui nous ont poussées à écarter l'approche cible-vers-pointeur. C'est-à-dire qu'il nous est impossible de déterminer a priori les cibles potentielles d'une activation sans remettre en cause les bases logicielles existantes.

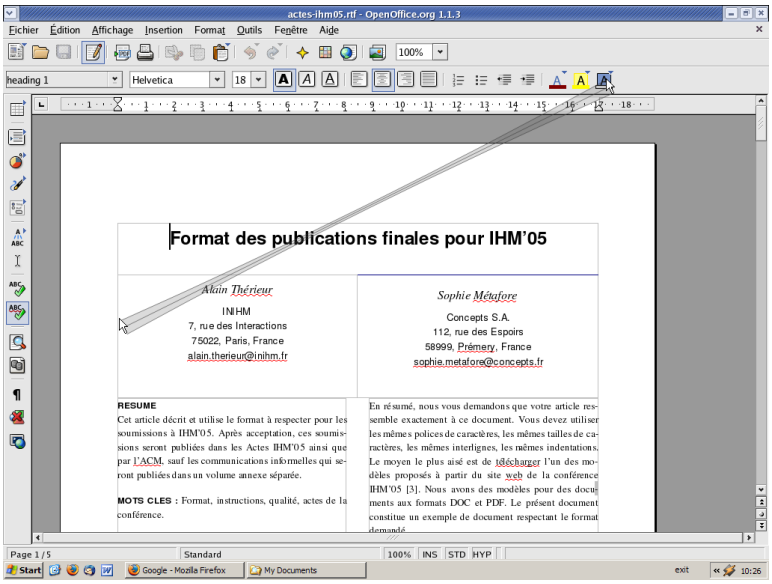

Fig. A.5: *utilisation du drag-and-click.*

3.2 Activation de la technique d'interaction

Le *drag-and-click* consiste à remplacer un clic qu'il faudrait faire loin du pointeur par un petit *drag-and-drop* à proximité du pointeur.

Cependant, la question de l'activation du *drag-and-click* n'est pas encore résolue. Plusieurs possibilités s'offrent à nous. La première serait de ne permettre les opérations de *drag-and-click* que sur les zones ne gérant pas le *drag-and-drop*. Cependant, cette solution réduit considérablement le champ d'utilisation du *drag-and-click*. En effet, un espace de travail contient souvent une zone centrale prédominante gérant le *drag-and-drop* (e.g. un éditeur quelconque).

Une seconde solution serait d'utiliser une touche spéciale (alt, ctrl). Cette solution peut également poser des problèmes car, dans le cas d'utilisation des surfaces augmentées de grande taille, l'utilisateur dispose rarement d'un clavier à portée de main.

La troisième solution serait d'utiliser la technique du clic et demi que les utilisateurs de pavés tactiles d'ordinateurs portables connaissent. Cette technique consiste à effectuer un clic suivi immédiatement d'un *drag-and-drop*.

Enfin, nous pensons que la solution idéale serait de disposer d'un bouton sur le stylo de la surface augmentée. Ce bouton permettrait de passer d'un système de pointage direct à un

système de pointage à distance et vice-versa. En attendant l'avènement de tels dispositifs, la solution du clic et demi nous paraît être la plus satisfaisante.

4 Conclusion

Dans cette annexe, nous avons présenté la démarche de conception d'une nouvelle technique d'interaction : le *drag-and-click*. Cette technique est une adaptation des techniques de type *drag-and-drop* et permet l'activation de composants à distance.

Le *drag-and-click* se destine aux surfaces augmentées de grande taille et permet de pallier aux difficultés qui peuvent survenir pour atteindre certaines parties de l'affichage, ou lorsqu'il y a plusieurs affichages.

ANNEXE B
QUESTIONNAIRE UTILISÉ POUR L'ÉTUDE PRÉSENTÉE AU CHAPITRE IV

Sexe :
- ❍ homme
- ❍ femme

Quel age avez-vous ?
- ❍ 18-24
- ❍ 25-34
- ❍ 35-44
- ❍ 45-60

Depuis combien de temps utilisez-vous un ordinateur (mac ou pc) ?
- ❍ moins de 5 ans
- ❍ entre 5 et 10 ans
- ❍ plus de 10 ans

En moyenne, combien de temps passez-vous devant votre ordinateur ?
- ❍ rarement
- ❍ une à deux fois par semaine
- ❍ plusieurs fois par semaine
- ❍ une à deux fois par jour
- ❍ tout le temps

Etes-vous droitier ou gaucher ?
- ❍ Droitier
- ❍ Gaucher

Avez-vous déjà utilisé un tableau interactif (type SmartBoard) ?
- ❍ souvent
- ❍ parfois
- ❍ jamais

Les questions suivantes concernent l'experience

Quelle est votre technique préférée ?

- ❍ Drag-and-drop
- ❍ Pick-and-Drop
- ❍ Drag-and-Pop
- ❍ Push-and-Throw
- ❍ Push-and-Pop
- ❍ Push-and-throw accéléré

Pouvez-vous classer ces techniques par ordre de préférence (la meilleure en premier) ?

- ❑ Drag-and-drop
- ❑ Pick-and-Drop
- ❑ Drag-and-Pop
- ❑ Push-and-Throw
- ❑ Push-and-Pop
- ❑ Push-and-throw accéléré

Avec un peu plus d'entraînement sur votre technique favorite, vous seriez :

- ❍ un peu meilleur
- ❍ bien meilleur
- ❍ je ne sais pas

Ce que je préfère dans ma technique favorite :

- ❍ l'aspect prévisible
- ❍ la précision
- ❍ la rapidité
- ❍ la facilité d'utilisation

Avez-vous rencontré des problèmes particuliers ?

Avez-vous des idées d'amélioration pour ces techniques ?

BIBLIOGRAPHIE

Apple developer connection. Drag manager programmers guide. http://developer.apple.com/documentation/carbon/conceptual/dragmgrpro-grammersguide/dragmgrprogrammersguide, a.

Apple developer connection. Drag and drop. http://developer.apple.com/documentation/cocoa/conceptual/draganddrop/, b.

Nicolas Barralon, Christophe Lachenal, and Joëlle Coutaz. Couplage de ressources d'interaction. In *Actes de la 16ème conférence francophone sur l'Interaction Homme-Machine (IHM'04)*, pages 13–20. ACM Press, 2004.

Bartels Media. Maxivista. http://www.maxivista.com/.

P. Baudisch, E. Cutrell, D. Robbins, M. Czerwinski, P. Tandler, B. Bederson, and Z. Zierlinger. Drag-and-pop and drag-and-pick : Techniques for accessing remote screen content on touch- and pen-operated systems. In *Human-Computer Interaction – INTERACT'03*, pages 57–64. IOS Press, (c) IFIP, 2003. URL http://www.idemployee.id.tue.nl/g.w.m.rauterberg/conferences/INTERACT2003/INTERACT2003-p57.pdf.

P. Baudisch, D. Tan, M. Collomb, D. Robbins, K. Hinckley, M. Agrawala, S. Zhao, and G. Ramos. Phosphor : Explaining transitions in the user interface using afterglow effects. In *UIST'06*, pages ? ?– ? ?, 2006.

Patrick Baudisch, Nathaniel Good, Victoria Bellotti, and Pamela Schraedley. Keeping things in context : a comparative evaluation of focus plus context screens, overviews, and zooming. In *CHI '02 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 259–266. ACM Press, 2002. ISBN 1-58113-453-3. doi : http://doi.acm.org/10.1145/503376.503423. URL http://www.patrickbaudisch.com/publications/2002-Baudisch-CHI02-KeepingThingsInContext.pdf.

Michel Beaudouin-Lafon. Instrumental interaction : an interaction model for designing post-wimp user interfaces. In *CHI '00 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 446–453, New York, NY, USA, 2000. ACM Press. ISBN 1-58113-216-6. doi : http://doi.acm.org/10.1145/332040.332473.

Anastasia Bezerianos and Ravin Balakrishnan. The vacuum : facilitating the manipulation of distant objects. In *CHI '05 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 361–370, New York, NY, USA, 2005. ACM Press. ISBN 1-58113-998-5. doi : http://doi.acm.org.gate6.inist.fr/10.1145/1054972.1055023.

Eric A. Bier, Steve Freeman, and Ken Pier. Mmm : The multi-device multi-user multi-editor. In *CHI '92 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 645–646, New York, NY, USA, 1992. ACM Press. ISBN 0-89791-513-5. doi : http://doi.acm.org/10.1145/142750.143065.

Renaud Blanch, Yves Guiard, and Michel 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. ISBN 1-58113-702-8. doi : http://doi.acm.org.gate6.inist.fr/10.1145/985692.985758.

Renaud Blanch, Michel Beaudouin-Lafon, Stéphane Conversy, Yannick Jestin, Thomas Baudel, and Yun Peng Zhao. Indigo : une architecture pour la conception d'applications graphiques interactives distribuées. In *Actes de la 17ème conférence francophone sur l'Interaction Homme-Machine (IHM'05)*. ACM Press, 2005.

Kellogg S. Booth, Brian D. Fisher, Chi Jui Raymond Lin, and Ritchie Argue. The "mighty mouse" multi-screen collaboration tool. In *UIST '02 : Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 209–212, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-488-6. doi : http://doi.acm.org/10.1145/571985.572016.

James Brown. Ole drag and drop from scratch. http://www.catch22.net/tuts/dragdrop.asp.

Barry Brumitt, Brian Meyers, John Krumm, Amanda Kern, and Steven A. Shafer. Easyliving : Technologies for intelligent environments. In *HUC '00 : Proceedings of the 2nd international symposium on Handheld and Ubiquitous Computing*, pages 12–29, London, UK, 2000. Springer-Verlag. ISBN 3-540-41093-7.

Stéphane Chatty. Extending a graphical toolkit for two-handed interaction. In *UIST '94 : Proceedings of the 7th annual ACM symposium on User interface software and technology*, pages 195–204, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-657-3. doi : http://doi.acm.org.gate6.inist.fr/10.1145/192426.192500.

Citrix. Citrix presentation server. http://www.citrix.com/English/ps2/products/product.asp?contentID=186&ntref=hp_nav_US.

Jr. Dan R. Olsen and Travis Nielsen. Laser pointer interaction. In *CHI '01 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 17–22, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-327-8. doi : http://doi.acm.org.gate6.inist.fr/10.1145/365024.365030.

Gene De Lisa. How to drag and drop with java 2. http://www.javaworld.com/javaworld/jw-03-1999/jw-03-dragndrop.html.

Digital tigers. Sidecar. http://www.digitaltigers.com/sidecar.shtml.

Sarah A. Douglas, Arthur E. Kirkpatrick, and I. Scott 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. ISBN 0-201-48559-1. doi : http://doi.acm.org.gate6.inist.fr/10.1145/302979.303042.

Jörg Geißler. Shuffle, throw or take it ! working efficiently with an interactive wall. In *CHI '98 : CHI 98 conference summary on Human factors in computing systems*, pages 265–266. ACM Press, 1998. ISBN 1-58113-028-7. doi : http://doi.acm.org/10.1145/286498.286745. URL http://portal.acm.org/citation.cfm?id=286745.

Yves Guiard, Renaud Blanch, and Michel 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. ISBN 1-56881-227-2.

Mountaz Hascoët. Throwing models for large displays. In *HCI'2003, Designing for society, Volume 2*, pages 73–77. British HCI Group, 2003.

Mountaz Hascoët and Frederic Sackx. Exploring interaction strategies with wall-screen : A new dual-display device for managing collections of web pages. In *IV*, pages 719–725, 2002. URL http://csdl.computer.org/comp/proceedings/iv/2002/1656/00/16560719abs.htm.

Ken Hinckley, Gonzalo Ramos, Francois Guimbretiere, Patrick Baudisch, and Marc Smith. Stitching : pen gestures that span multiple displays. In *AVI '04 : Proceedings of the working conference on Advanced visual interfaces*, pages 23–31, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-867-9. doi : http://doi.acm.org.gate6.inist.fr/10.1145/989863.989866.

Gnome Developper's Homepage. Drag and drop – functions for controlling drag and drop handling. http://developer.gnome.org/doc/API/gtk/gtk-drag-and-drop.html.

Bruce Horn. On xerox, apple and progress. http://www.applehistory.com/frames/body.php?page=gui_horn1.

Juan Pablo Hourcade and Benjamin B. Bederson. Architecture and implementation of a java package for multiple input devices (MID). Technical Report CS-TR-4018, HCIL, 1999. URL citeseer.ist.psu.edu/hourcade99architecture.html.

Juan Pablo Hourcade, Benjamin B. Bederson, and Allison Druin. Building kidpad : an application for children's collaborative storytelling. *Softw. Pract. Exper.*, 34(9) :895–914, 2004. ISSN 0038-0644. doi : http://dx.doi.org/10.1002/spe.598.

Dugald Ralph Hutchings, John Stasko, and Mary Czerwinski. Distributed display environments. *interactions*, 12(6) :50–53, 2005. ISSN 1072-5520. doi : http://doi.acm.org.gate6.inist.fr/10.1145/1096554.1096592.

Fraunhofer institute. The dynawall project. http://www.ipsi.fraunhofer.de/ambiente/english/projekte/projekte/dynawall.html.

Shahram Izadi, Harry Brignull, Tom Rodden, Yvonne Rogers, and Mia Underwood. Dynamo : a public interactive surface supporting the cooperative sharing and exchange of media. In *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 159–168, New York, NY, USA, 2003. ACM Press. ISBN 1-58113-636-6. doi : http://doi.acm.org/10.1145/964696.964714.

Brad Johanson, Armando Fox, and Terry Winograd. The interactive workspaces project : Experiences with ubiquitous computing rooms. *IEEE Pervasive Computing*, 1(2) :67–74, 2002a. ISSN 1536-1268. doi : http://dx.doi.org/10.1109/MPRV.2002.1012339.

Brad Johanson, Greg Hutchins, Terry Winograd, and Maureen Stone. Pointright : experience with flexible input redirection in interactive workspaces. In *UIST '02 : Proceedings of the 15th annual ACM symposium on User interface software and technology*, pages 227–234, New York, NY, USA, 2002b. ACM Press. ISBN 1-58113-488-6. doi : http://doi.acm.org/10.1145/571985.572019.

Azam Khan, George Fitzmaurice, Don Almeida, Nicolas Burtnyk, and Gordon Kurtenbach. A remote control interface for large displays. In *UIST '04 : Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 127–136, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-957-8. doi : http://doi.acm.org.gate6.inist.fr/10.1145/1029632.1029655.

Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in smalltalk-80. *J. Object Oriented Program.*, 1(3) :26–49, 1988. ISSN 0896-8438.

Christophe Lachenal. *Modèle et infrastructure logicielle pour l'interaction multi-instrument multisurface*. PhD thesis, Université Joseph Fourier, 2004. URL http://iihm.imag.fr/publs/2004/theseLachenal.pdf.

Eric Lecolinet. A molecular architecture for creating advanced guis. In *UIST '03 : Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 135–144, New York, NY, USA, 2003a. ACM Press. ISBN 1-58113-636-6. doi : http://doi.acm.org/10.1145/964696.964711.

Eric Lecolinet. Multiple pointers : a study and an implementation. In *IHM 2003 : Proceedings of the 15th French-speaking conference on human-computer interaction on 15eme Conference Francophone sur l'Interaction Homme-Machine*, pages 134–141, New York, NY, USA, 2003b. ACM Press. ISBN 1-58113-803-2. doi : http://doi.acm.org/10.1145/1063669.1063688.

Ben Levitt. Collaborative vnc. http://benjie.org/software/linux/collaborative-vnc/.

Sheng Feng Li, Quentin Stafford-Fraser, and Andy Hopper. Integrating synchronous and asynchronous collaboration with virtual network computing. *IEEE Internet Computing*, 4 (3) :26–33, 2000. URL citeseer.ist.psu.edu/article/li00integrating.html.

Simon Lok, Steven K. Feiner, William M. Chiong, and Yoav J. Hirsch. A graphical user interface toolkit approach to thin-client computing. In *WWW '02 : Proceedings of the 11th international conference on World Wide Web*, pages 718–725, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-449-5. doi : http://doi.acm.org/10.1145/511446.511540.

Brad A. Myers, Herb Stiel, and Robert Gargiulo. Collaboration using multiple pdas connected to a pc. In *CSCW '98 : Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 285–294. ACM Press, 1998. ISBN 1-58113-009-0. doi : http://doi.acm.org/10.1145/289444.289503. URL http://portal.acm.org/citation.cfm?coll=GUIDE&dl=GUIDE&id=289503.

Microsoft Developer Network. Overview of gdi+. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/gdicpp/GDIPlus/aboutGDIPlus/introductiontoGDIPlus/overviewofGDIPlus.asp.

Newnham research. Usb nivo. http://www.newnhamresearch.com/products/usbnivo.htm.

J. Karen Parker, Regan L. Mandryk, and Kori M. Inkpen. Tractorbeam : seamless integration of local and remote pointing for tabletop displays. In *GI '05 : Proceedings of the 2005 conference on Graphics interface*, pages 33–40, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 2005. Canadian Human-Computer Communications Society. ISBN 1-56881-265-5.

Atul Prakash and Hyong Sop Shim. Distview : support for building efficient collaborative applications using replicated objects. In *CSCW '94 : Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 153–164, New York, NY, USA, 1994. ACM Press. ISBN 0-89791-689-1. doi : http://doi.acm.org.gate6.inist.fr/10.1145/192844.192895.

Jun 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. ACM Press, 1997. ISBN 0-89791-881-9. doi : http://doi.acm.org/10.1145/263407.263505. URL http://www.csl.sony.co.jp/person/rekimoto/papers/uist97.pdf.

Jun Rekimoto and Masanori Saitoh. Augmented surfaces : A spatially continuous work space for hybrid computing environments. In *CHI*, pages 378–385, 1999. URL citeseer.ist.psu.edu/rekimoto99augmented.html.

Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1) :33–38, 1998. URL citeseer.ist.psu.edu/richardson98virtual.html.

Kurt Saar. Virtus : a collaborative multi-user platform. In *VRML '99 : Proceedings of the fourth symposium on Virtual reality modeling language*, pages 141–152, New York, NY, USA, 1999. ACM Press. ISBN 1-58113-079-1. doi : http://doi.acm.org/10.1145/299246.299287.

Chris Schoeneman. Synergy. http://synergy2.sourceforge.net/.

Christian Schuckmann, Lutz Kirchner, Jan Schümmer, and Jörg M. Haake. Designing object-oriented synchronous groupware with coast. In *CSCW '96 : Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 30–38, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-765-0. doi : http://doi.acm.org/10.1145/240080.240186.

Chia Shen, Frédéric D. Vernier, Clifton Forlines, and Meredith Ringel. Diamondspin : an extensible toolkit for around-the-table interaction. In *CHI '04 : Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 167–174, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-702-8. doi : http://doi.acm.org/10.1145/985692.985714.

SMART Technologies. Smart board. http://www.smarttech.com/.

Sony Japan. Flyingpointer. http://www.sony.jp/products/consumer/pcom/software_02q1/flyingpointer.

Jason Stewart, Benjamin B. Bederson, and Allison 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, 1999. URL http://citeseer.ist.psu.edu/stewart99single.html.

Desney S. Tan, Brian Meyers, and Mary Czerwinski. Wincuts : manipulating arbitrary window regions for more effective use of screen space. In *CHI '04 : CHI '04 extended abstracts on Human factors in computing systems*, pages 1525–1528, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-703-6. doi : http://doi.acm.org/10.1145/985921.986106.

Virtual Ink. Mimio interactive. http://www.mimio.com/.

Wikimedia. Wikipedia, the free encyclopedia. http://www.wikipedia.org/.

Xdmx project. Distributed multihead x. http://dmx.sourceforge.net/.

Résumé

Ce mémoire a pour objectif l'étude des interactions qui interviennent dans les environnements où plusieurs surfaces d'affichage et dispositifs de pointage sont mis à disposition du ou des utilisateurs, que ces environnements soient distribués ou non. Nous nous sommes plus particulièrement intéressés aux techniques de type *drag-and-drop* dans le contexte du mur-écran.

Le *drag-and-drop* est une technique très répandue qui atteint ses limites dans certaines conditions d'utilisation. Ces limites peuvent être physiques – avec des écrans interactifs par exemple – ou dues à l'implémentation. D'une part, ce mémoire propose plusieurs évolutions du *drag-and-drop*, avec comme objectif de dépasser les limites physiques du *drag-and-drop*. En particulier, il devient possible de dépasser les limites de la loi de Fitts ainsi que les limites physiques des surfaces d'affichage. Un ensemble de techniques de types *drag-and-drop* issues de l'état de l'art ou proposées dans ce mémoire font l'objet d'études utilisateurs. A l'issue de ces études, deux techniques ressortent principalement : le *push-and-pop* et le *push-and-throw* accéléré.

D'autre part, afin de dépasser les limites dues à l'implémentation du *drag-and-drop*, nous proposons un modèle d'implémentation pour le *drag-and-drop* et l'ensemble de ses évolutions. Ce modèle d'implémentation s'appuie sur le modèle d'interaction instrumentale. Ce modèle d'implémentation est mis en œuvre dans l'API *PoIP*, elle-même utilisée dans l'application *Orchis*. Grâce à ce modèle, il est possible d'effectuer des opérations de type *drag-and-drop* distribuées, la technique d'interaction pouvant être personnalisée en fonction des contraintes matérielles ou des préférences de l'utilisateur.

Abstract

This work examines the interactions at stake in environments where several display surfaces and pointing devices are used. These environments can be distributed or not. We are more particularly interested in drag-and-drop like techniques in the context of wall-size displays.

Drag-and-drop is probably the most emblematic and widespreadly used technique of direct manipulation paradigms. Nevertheless it reaches its limits with new emerging distributed display surfaces and wall-size displays. Its limits are of different kinds : material – direct input devices and large surfaces are difficult to handle with regular drag-and-drop – or software Ű- drag-and-drop over distributed windowing systems is not an easy game. Several alternatives to drag-and-drop emerged over the past years either proposed by us or others. In this work we compared all of them both analytically and empirically. This comparison leads us to design two new alternatives : push-and-pop and accelerated push-andŰthrow.

Furthermore, we provided an implementation model for drag-and-drop and its alternatives. This model is based on the instrumental interaction model. We further built PoIP which is an API based on our implementation model to carry out drag-and-drop like operations. The main contribution of this API is in its support for both distributed surfaces and heterogeneous interaction styles.