Evolution du Drag-and-Drop : Du Modèle d'Interaction Classique aux Surfaces Multi-Supports

Mountaz Hascoët, Maxime Collomb, Renaud Blanch

Évolution du *drag-and-drop* du modèle d'interaction classique aux surfaces multi-supports

Mountaz Hascoët*, Maxime Collomb† et Renaud Blanch‡

* LIRMM, UMR 5506 du CNRS, 161 rue Ada
34392 Montpellier Cedex mountaz@lirmm.fr
† LIRMM, UMR 5506 du CNRS, 161 rue Ada
34392 Montpellier Cedex collomb@lirmm.fr
‡ LRI & INRIA Futurs – Université Paris-Sud XI
91405 Orsay Cedex blanch@lri.fr

Résumé

La visualisation d'information sur des dispositifs d'affichage et d'interaction augmentés au travers de surfaces multi-supports remet en cause l'une des techniques d'interaction les plus emblématiques de la manipulation directe: le drag-and-drop. *Ces dernières années de nouvelles techniques d'interaction sont apparues mais il reste encore beaucoup à faire pour que l'intégration de ces nouveautés se concrétise.*

L'objectif de cet article est de présenter les récentes innovations dans ce domaine et de montrer comment l'intégration de ces innovations peut s'envisager dans les mécanismes actuels. Nous présentons aussi une étude transversale des techniques de drag-and-drop *actuelles, mettant en évidence les écueils attendus.*

Mots-clés *visualisation multi-supports, surfaces augmentées, technique d'interaction, glisser-déposer,* drag-and-drop.

Abstract

Information visualisation on augmented display surfaces impose a revision on one of the most relevant techniques for direct manipulation: drag-and-drop. Over the past few years, new interaction techniques have emerged to encompass the limitations of traditional drag-and-drop for these new display surfaces.

In this paper, we present the main evolutions and discuss the possible integration of these innovative solutions into the

existing drag-and-drop framework. We further present the main principles underlying the implementation of drag-and-drop in most traditional systems, and we discuss as well the expected pitfalls.

Keywords *multi-displays information visualisation, augmented surfaces, interaction technique, drag-and-drop.*

1. INTRODUCTION

Il y a 30 ans, une révolution allait sans conteste changer radicalement la manière dont l'outil informatique permettrait de visualiser et de manipuler l'information. Il s'agissait de l'apparition de l'interface graphique (GUI – *Graphical User Interface*) et des techniques d'interaction associées, dont beaucoup mettent en jeu la souris. La souris, telle que nous la connaissons aujourd'hui, a été conçue par Douglas Englebart [14] l'institut de recherche de Stanford en 1964. En 1973, on pouvait trouver au *Xerox PARC* le premier poste disposant d'un environnement graphique. Le *Xerox Alto* qui proposait entre autres une souris à trois boutons, un affichage bitmap, des fenêtres graphiques et le réseau Ethernet. L'environnement graphique et la souris ne seront réellement utilisés par le grand public qu'à partir de 1984, lors de la commercialisation du Macintosh d'Apple.

L'avènement des dispositifs d'affichage et d'interaction augmentés au travers de surfaces multi-supports [12] et de nouvelles techniques d'interactions laissent présager aujourd'hui d'une nouvelle révolution dans le domaine de la visualisation interactive. Le *drag-and-drop* (glisser-déposer) que l'on peut considérer comme l'une des techiques d'interaction les plus génériques et les plus réussies du modèle de manipulation directe est déjà pris dans cette révolution. En effet, entre le *click-and-drag* précurseur que l'on trouve dans *Smalltalk* (méthode de sélection d'un élément d'un menu), le *drag-and-drop* à proprement parler (déplacement de fichiers ou d'objets) inventé un peu plus tard chez Apple [10] et les extensions récentes telles que le *pick-and-drop*, ce modèle a vu ses capacités décuplées.

En commençant par une présentation des évolutions récentes du *drag-and-drop* vers des surfaces augmentées ou multi-supports (section 2), l'objectif de cet article est de montrer que les mécanismes de *drag-and-drop* actuels doivent évoluer pour supporter cette évolution.

Dans les sections 3 et 4 nous présentons donc les mécanismes d'implantation du *drag-and-drop* classique dans trois systèmes de fenêtrage

significatifs que sont Mac OS X/Carbon, X-Window/Motif et Microsoft Windows. Pour aborder le problème à un niveau d'abstraction plus élevé, nous complétons ce panorama par la présentation du mécanisme de *drag-and-drop* mis en oeuvre dans Java/Swing, boîte à outils de plus haut niveau d'abstraction et de surcroît multi-plateformes puisqu'elle repose sur la machine virtuelle Java.

Une discussion sur l'ensemble de ces modèles de *drag-and-drop* fait ensuite l'objet de la section 3. Nous y mettons en évidence les points communs entre les modèles présentés ainsi que les principaux avantages et inconvénients des uns et des autres. Nous terminons par une discussion des possibilités d'intégration des extensions du *drag-and-drop* dans les modèles existants et des révisions du modèle de base que celles-ci imposent.

2. LES EXTENSIONS DU *DRAG-AND-DROP*

Au cours des dernières années, les progrès réalisés au niveau des surfaces multiples et des surfaces augmentées ont contribué à leur développement et plusieurs nouveaux paradigmes ont été proposés pour déplacer des objets ou des données dans des environnements augmentés. En effet, le *drag-and-drop,* tel que nous le connaissons, atteint ses limites lorsque l'on utilise des systèmes plus complexes que la traditionnelle configuration de la station de travail individuelle comportant une unité centrale, un écran et une souris. Et si la méthode du *drag-and-drop* est théoriquement utilisable sur les surfaces tactiles, elle s'avère cependant malcommode surtout si la surface est de grande taille (pour le SmartBoard par exemple). On comprend ainsi clairement le besoin d'évolution du *drag-and-drop*.

Historiquement, l'une des premières techniques étendant le *drag-and-drop* proposée fut le *pick-and-drop* [13].

2.1. Pick-and-drop

Le *pick-and-drop* se propose de permettre à l'utilisateur de faire une chose impossible à faire avec *drag-and-drop* : transférer des données par manipulation directe d'une machine à une autre. 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.

Le *pick-and-drop* est symboliquement plus proche du copier-coller que du *drag-and-drop* mais il partage avec ce dernier l'avantage de dispenser

l'utilisateur d'avoir conscience de l'existence d'un presse-papier et d'en connaître le fonctionnement. De la même manière que le *drag-and-drop* est plus naturel pour l'utilisateur que le copier-coller, le *pick-and-drop* est plus naturel qu'un copier-coller inter-machine utilisant le réseau.

2.1.1. Utilisation

Cette technique ne peut être utilisée qu'entre deux surfaces tactiles acceptant le même type de stylos et dont les unités centrales sont connectées entre-elles. En effet, les données sont associées à un stylo (il n'est pas possible de prendre un objet avec un stylo et de le déposer avec un autre stylo), et elles sont transférées via le réseau.

Chaque stylo possède un identifiant (ID) et les données à transférer sont associées à l'ID du stylo avec lequel elles ont été «□prises□. Même si le système donne l'impression que le stylo «□porte□ les données, il n'en est rien□: chaque stylo possède son ID propre et un serveur sur le réseau est chargé de faire la correspondance entre ID du stylo et données qui ont été «□prises□.

2.1.2. Les évènements

Les évènements intervenant dans un *pick-and-drop* sont relativement simples□:

- L'utilisateur pose son stylo sur un objet. Si le stylo n'est pas relevé après un certain temps, une opération de *drag-and-drop* classique est déclenchée, et si le stylo est relevé rapidement, le système considère qu'il s'agit d'une opération de *pick-and-drop* et l'objet est copié.
- L'utilisateur pose son stylo sur une autre surface. L'objet est déposé et peut être glissé. L'opération se termine lorsque l'utilisateur relève son stylo.

On remarque donc, qu'au même titre que le *drag-and-drop*, cette technique possède un inconvénient par rapport au copier-coller classique□: à partir du moment où un objet est en cours de déplacement, aucune interaction n'est possible avant que l'objet ne soit déposé.

2.1.3. Les données

Dans le prototype présenté en□1997, toutes les données étaient gérées par des programmes écrits en Java et ces données étaient transférées via des objets héritant de la classe Serializable qui sont transformables en séquences

d'octets. La machine source transforme l'objet à transférer en une séquence d'octets, cette séquence est transmise via le réseau à la machine cible qui reconstruit l'objet.

En s'intéressant davantage au domaine du travail collaboratif, Rekimoto a aussi développé l'*hyperdragging*.

2.2. Hyperdragging

L'*hyperdragging* [12] est une extension permettant de réaliser des *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 murale et horizontale (table) 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.

2.2.1. Utilisation

L'utilisation de l'*hyperdragging* est transparente : l'utilisateur effectue 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). Cette aide visuelle est appelée « ancrage du curseur ».

L'auteur ne donne pas de précisions quant aux évènements et aux données utilisés.

2.3. Shuffle, throw or take it

En 1998, Geissler [8] proposait trois techniques pour travailler plus efficacement avec les murs interactifs. L'objectif était de proposer de nouvelles techniques d'interaction pour le DynaWall (combinaison de trois SmartBoard offrant une surface totale de 3072x1024 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 200x300□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 lancé 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□».

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.

Ces techniques n'ont été soumises à aucun test et n'ont pas été décrites plus en profondeur.

2.4. Drag-and-pop

Le *drag-and-pop*□[1] a été initialement créé par Baudisch pour permettre à l'utilisateur d'une ou plusieurs surfaces tactiles de réaliser des *drag-and-drop* dans des situations où cela était difficile voire impossible. Un exemple d'utilisation où le *drag-and-drop* est difficile est l'utilisation du DynaWall du fait de sa très grande surface. Un exemple d'utilisation où l'utilisation du *drag-and-drop* est impossible est le déplacement d'un objet entre deux surfaces tactiles□: il faut en effet soulever le stylo pour passer d'une surface à l'autre ce qui interromprait le *drag-and-drop.*

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 (FIG. 1). Celles-ci sont déterminées en fonction de la direction initiale du glisser.

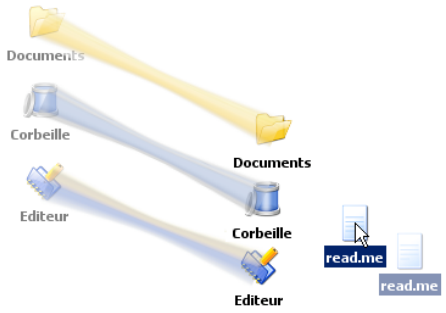

FIG. 1 – *Exemple de* drag-and-pop.

2.4.1. Utilisation

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 curseur 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.

Si le mouvement initial de l'utilisateur n'a pas la bonne direction, et que par conséquent l'icône visée n'est pas rapprochée, l'utilisateur peut faire disparaître les icônes fantômes en s'éloignant d'elles.

2.5. Drag-and-throw et push-and-throw

Le *drag-and-throw* et le *push-and-throw* [9] sont des techniques de lancer d'icônes adaptées aux surfaces multiples. Ces techniques permettent de déplacer des icônes à partir d'une surface vers une autre (que ces deux surfaces soient reliées à la même machine ou non).

Ces deux techniques sont basées sur les trois mêmes principes, à savoir : un retour visuel, une métaphore, et la définition explicite d'une trajectoire de l'objet.

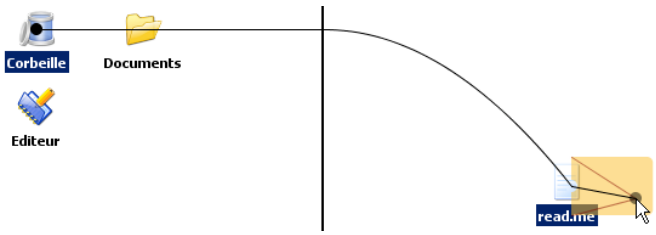

FIG. 2 – *Exemple de* drag-and-throw

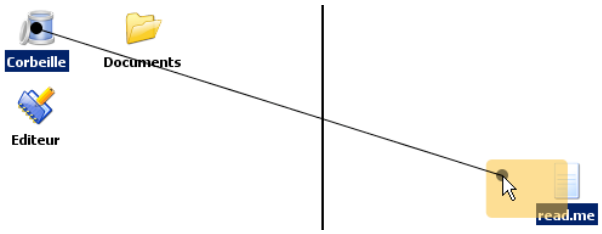

FIG. 3 – *Exemple de* push-and-throw

On peut distinguer trois types de retours visuels□

- la trajectoire□ affichage de la trajectoire de l'objet à déplacer comme une ligne joignant sa position actuelle et la position où il serait déplacé si le bouton de la souris était relâché. Notons que la trajectoire peut être aimantée par les cibles possibles (si on cherche à déplacer l'icône d'un document, la trajectoire sera aimantée par les objets compatibles□ corbeille ou dossier par exemple).
- la mire□affichage d'un point terminant la trajectoire.
- la zone de «□décollage□ : c'est l'affichage sur la surface source d'une zone semi-transparente qui se projette sur la totalité de la surface cible. Cette aide visuelle permet à l'utilisateur de comprendre où il peut déplacer le curseur (ou le stylo, s'il s'agit d'une surface tactile). On peut assimiler cette zone de décollage à une vue radar de la surface cible (stylisée par un simple rectangle de couleur unie).

La différence entre le *drag-and-throw* et le *push-and-throw* se situe au niveau des trajectoires et de la méthode de projection de la zone de décollage sur la surface cible.

Le *drag-and-throw* utilise la métaphore du tir à l'arc (FIG. 2), c'est-à-dire que l'utilisateur effectue un mouvement dans le sens opposé à la direction dans laquelle il veut lancer l'objet. S'il veut envoyer l'objet plus haut, il

doit déplacer le curseur vers le bas. La trajectoire associée au *drag-and-throw* se compose d'une courbe de Bézier sur la surface source et d'une ligne droite sur la surface cible.

Le *push-and-throw,* quant à lui, exploite la métaphore du pantographe (FIG. 3), c'est-à-dire que l'utilisateur fait un petit mouvement de souris qui est amplifié dans la même direction pour déterminer où lancer l'objet. La trajectoire est dessinée quant à elle par une simple ligne droite joignant le curseur de la souris au point où sera lancé l'objet.

2.5.1. Utilisation

L'utilisateur débute un *drag-and-drop* classique, et, après un petit mouvement (quelques pixels – suffisamment pour déterminer la direction du glisser-déposer), les aides visuelles apparaissent : une zone de décollage sur la surface source, une mire sur la surface cible, et une trajectoire joignant la mire au curseur de la souris. L'utilisateur ajuste ensuite son lancer puis relâche le bouton de la souris ce qui a pour effet de déplacer immédiatement l'objet.

L'avantage incontestable de ces techniques de lancer est qu'elles permettent un contrôle précis du lancer. En effet, l'utilisateur connaît la position où sera déplacé l'objet avant de le lancer grâce aux indications visuelles. Il a donc la possibilité d'ajuster précisément son lancer. De plus, le calcul des trajectoires [9] permet d'éviter le comportement chaotique du lancer résultant de trajectoires basiques telles que celles imaginées dans [8].

2.5.2. Les évènements

Le prototype du *drag-and-throw* et du *push-and-throw* a été développé en Java et le système d'évènements utilisé est très proche de celui du *drag-and-drop* de Swing.

L'entité DntManager est destinée à prendre en charge le déroulement d'un déplacement par *drag-and-throw*. Si un composant est une cible potentielle, il s'enregistre auprès du DntManager en fournissant un *Listener* permettant au DntManager de notifier le composant lorsqu'une opération de lancer sera en cours. Les évènements reçus par le composant cible sont dntStart(), dntOver() et dntEnd() ayant respectivement lieu lorsqu'un lancer débute, est en cours et se termine.

Du côté du composant source, le fonctionnement est également très proche de ce qui passe pour le *drag-and-drop* de Swing. Le composant doit se munir d'un DragGestureListener lui permettant d'être notifié lorsqu'un

lancer débute. Si le lancer est validé par le composant, celui-ci sera alors notifié des mêmes évènements que le composant cible, à savoir dntStart(), dntOver() et dntEnd(). Cependant, les données reçues par chacune de ces méthodes (propriétés de l'évènement) sont différentes de celles reçues par le composant cible (par exemple, lors de l'évènement dntStart(), le composant source n'a pas besoin d'être informé des données en transfert puisqu'il est censé les connaître étant lui-même à l'origine de ce transfert).

2.5.3. Les données

Par souci de simplicité, les données transférées sont uniquement textuelles et sont au format XML. Il est donc nécessaire que le composant cible connaisse ce format pour pouvoir en extraire les données.

2.6. Conclusion

Les techniques présentées traduisent un besoin de solutions au problème du nombre de surfaces d'affichage croissant, ainsi qu'à l'hétérogénéité grandissante des dispositifs utilisés par une personne ou par un groupe de personnes collaborant à une tâche. Les techniques développées par les chercheurs reposent souvent sur des mécanismes *ad hoc* qui démontrent la pertinence et la faisabilité des approches envisagées. Cependant, à mesure que les dispositifs réservés aux laboratoires touchent un public plus vaste, il faudra envisager d'intégrer un support à ces techniques au sein même des systèmes. Pour déterminer l'ampleur de cette tâche, nous proposons maintenant d'étudier l'existant, à savoir les modèles de *drag-and-drop* présents dans les principaux systèmes de fenêtrage.

3. LES MODELES D'EVENEMENTS

L'implantation 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 étudions plus succinctement dans la section suivante. Ces deux aspects sont présents dans les quatre systèmes présentés : MacOS X/Carbon, X-Window/Motif, Microsoft Windows, et Java/Swing.

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 glissement de la souris, un bouton particulier étant 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 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 déposer 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 deux types de diagrammes. Le premier (voir FIG. 4 pour l'exemple de Mac OS X) présente du point de vue de ces trois acteurs et de ces cinq phases l'articulation fonctionnelle du *drag-and-drop*. Le second (voir FIG. 5 pour un exemple) présente plus formellement l'interaction entre les divers objets impliqués à l'aide d'un diagramme de séquence. Dans ce diagramme, la manipulation des données échangées n'est néanmoins pas représentée dans un souci de clarté.

3.1. Mac OS X

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 [4]. Nous considérons ici uniquement le fonctionnement du *drag-and-drop* pour la *toolkit* Carbon [6] 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 FIG. 4, sur laquelle le système inclut donc le DragManager, donne un aperçu du processus.

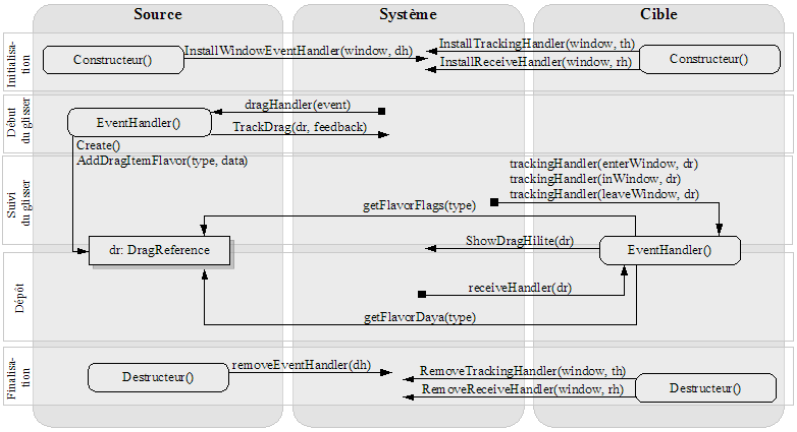

FIG. 4 - *Fonctionnement du* drag-and-drop *sous Carbon.*

C'est le DragManager qui permet initialement à une application d'enregistrer ses fenêtres comme étant des cibles pour le *drag-and-drop* grâce aux fonctions InstallTrackingHandler() et InstallReceiveHandler() (FIG. 4 et FIG. 5). 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 contenues par le glisser, 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 curseur (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 : le fait qu'un glisser entre dans leurs fenêtres, le fait qu'il survole les fenêtres (cet évènement est reproduit périodiquement au cours du survol) et enfin le fait qu'il sorte de la fenêtre. La cible peut alors vérifier la compatibilité de l'objet glissé en testant les types qu'il contient grâce à GetFlavorFlags() et adapter son *feedback (drag-under effect)*. Le

DragManager permet un *feedback* par défaut (surlignage 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().

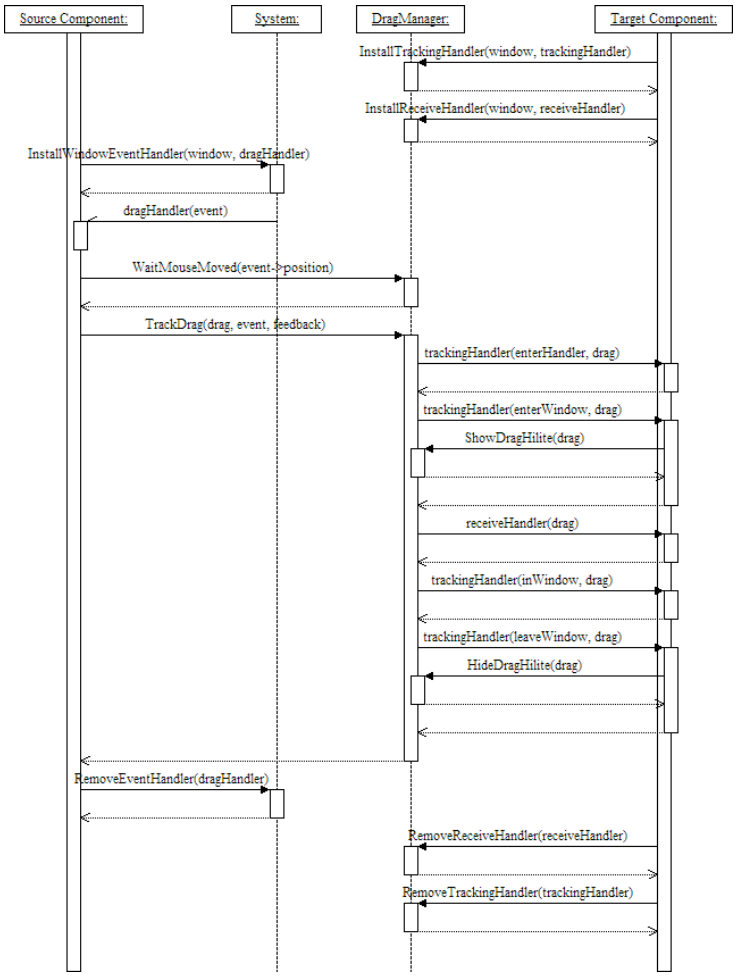

FIG. 5 – *Diagramme de séquence d'un* drag-and-drop *sous* Carbon.

Enfin, lors du déposer, le DragManager notifie l'application concernée si elle est enregistrée. La cible peut alors tester le type du contenu du glisser 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 déposer. Si le déposer 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.

3.2. X-Window

Dans la *toolkit* Motif, le modèle de *drag-and-drop* 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. Il s'appuie également sur le modèle de transfert uniforme (UTM - *Uniform Transfert Model*) introduit avec la version 2.0 de Motif et présenté dans la section 4.

Le mécanisme de communication inter-applications de base de X-Window repose sur la notion de propriété, d'atomes et de sélection. Avant de rentrer dans le détail du modèle de *drag-and-drop* sous Motif, il est donc nécessaire de rappeler brièvement quelques définitions.

Dans X-Window, une propriété représente toute information utile à une application. Elle est de plus nécessairement associée à une fenêtre. Des propriétés peuvent être associées à la fenêtre racine de l'écran principal, celle qui englobe toutes les autres. Dans ce cas elle est accessible par tout client X, et donc par toute application qui apparaît à l'écran. Ces propriétés sont définies par un nom et un type. Les atomes sont des identifiants permettant de référencer de manière unique ces propriétés. Un certain nombre d'atomes standard sont définis et utilisés par la plupart des applications qui peuvent demander à être notifiées des modifications faites sur certaines propriétés.

Un certain nombre d'objets ont été introduits pour faciliter le *drag-and-drop* (FIG. 6 et FIG. 7). L'objet qui permet d'initier un glisser est un objet de type DragSource. Un objet de type DragContext est utilisé pour stocker l'information pendant le déplacement. Le DropTransfer suit l'information pendant le déposer. L'objet DragIcon représente la forme du pointeur pendant le transfert. Enfin un objet de type DropSite gère l'information sur toutes les zones d'une application susceptible de recevoir un déposer. À chacun de ces objets sont associées des ressources et des *callbacks* qui forment ensemble le modèle de *drag-and-drop* de Motif. Nous donnons ici un peu plus de détails sur les objets qui se trouvent au cœur du mécanisme : DragSource, DragContext, DropSite et DragIcon.

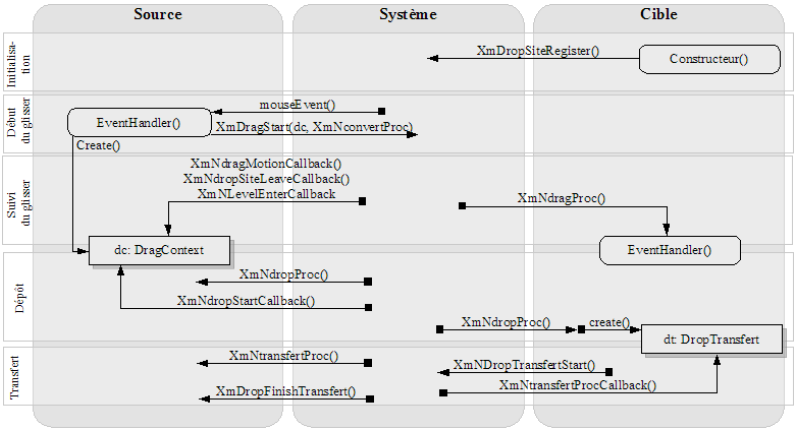

FIG. 6 – *Fonctionnement du* drag-and-drop *sous X-Window/Motif.*

Pour qu'un composant puisse devenir un DragSource, il faut qu'il reconnaisse les évènements de type ButtonPress pour le bouton du milieu de la souris. Lorsqu'un évènement de ce type est reconnu, l'appel à la routine XmStartDrag() permet de créer un objet de type DragContext. L'objet de type DragContext permet de gérer la suite du déplacement et stocke les informations utiles au travers de ressources telles que :

- XmNdragOperations qui spécifie les opérations permises sur l'objet DragSource ;
- XmNexportTargets qui détermine les types de données supportés ;
- XmNsourceCursorIcon, XmNoperationCursorIcon et XmNstateCursorIcon définissent certaines des images utilisées par l'objet de type DragIcon représentant l'objet en déplacement.

Des *callbacks* peuvent en outre être associées à un objet de type DragContext pour permettre de personnaliser le *feedback* visuel effectué par la source lors du déplacement *(drag-over effect)*.

Enfin, la procédure XmNconvertProv() doit être définie à la création de l'objet de type DragContext, elle sera utilisée pour convertir les données de la source dans le format demandé par la cible au moment du déposer.

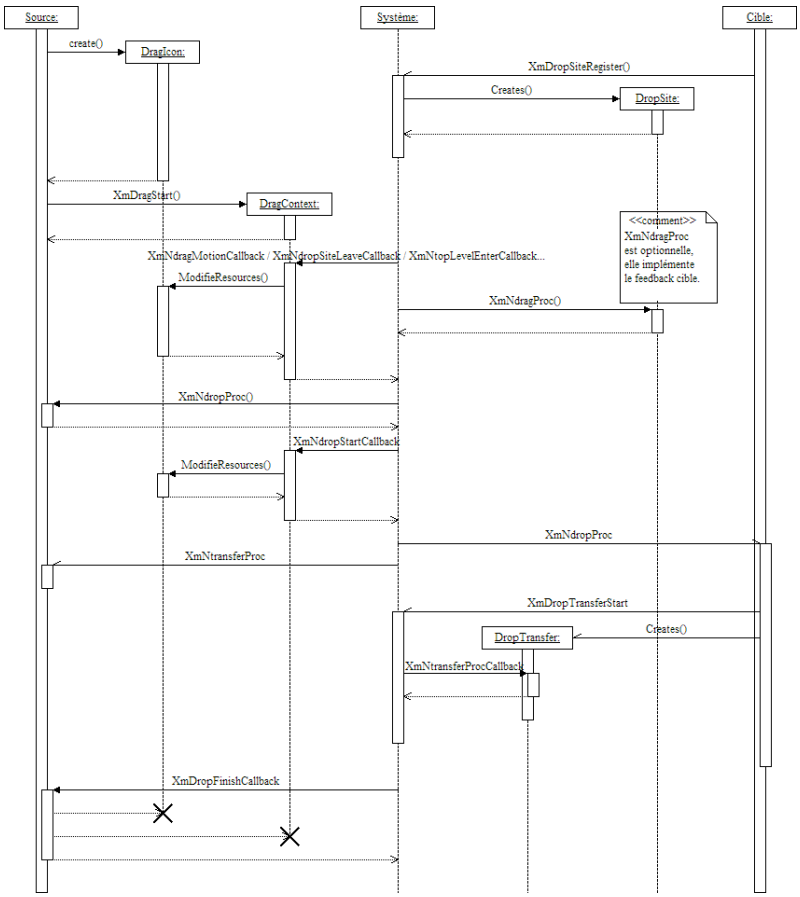

FIG. 7 - *Diagramme de séquence d'un* drag-and-drop *sous* Motif

.

Pour que le déposer puisse se faire, encore faut-il disposer de zones acceptant de recevoir un déposer. Dans *Motif*, tout composant peut devenir une cible appelée DropSite en s'enregistrant au moyen de la fonction XmDropSiteRegister(). Une cible doit ensuite spécifier quels types de données elle veut recevoir et quelles opérations elle supporte (copier, déplacer, faire un lien, annuler, etc.) Elle peut également définir les effets visuels utiles lorsqu'un objet la survole *(drag-under effect)*. De manière analogue à

l'objet de type DragContext créé par la source, la cible stocke ce type d'informations par l'intermédiaire de ressources telles que :

- XmNimportTargets et XmNnumImportTargets qui déterminent les types de données acceptés ;
- XmNdropSiteOperations qui définit les opérations supportées ;
- XmNanimationStyle qui définit des effets visuels animés de base lorsque le curseur passe au-dessus d'une cible.

Pour définir des effets visuels de type *drag-under effect* plus sophistiqués il faut définir une *callback* XmNdragProc qui sera invoquée à chaque fois qu'il se passera quelque chose au-dessus d'une cible.

Par ailleurs, une cible doit être associée à la *callback* XmNdropProc déterminant les actions à effectuer en cas de déposer. XmNdropProc fait elle-même appel à la routine XmDropTransfertStart() qui crée un objet de type DropTransfer gérant le transfert des données et déterminant l'issue de l'opération (succès ou échec). Pendant cette étape de déposer, les effets visuels ne peuvent pas être personnalisés. Si le déposer est un succès les deux icônes fusionnent. Si le déposer est un échec, l'icône associée à la source est renvoyée d'où elle venait pour indiquer à l'utilisateur que les données n'ont pas pu être transférées.

Pendant le glisser, la forme du pointeur est celle d'un objet de type DragIcon. Cet objet est responsable du *feedback* visuel dit *drag-over effect* qui peut indiquer le cas échéant :

- le passage de l'objet source au-dessus d'une zone potentiellement cible ;
- l'action en cours (déplacement, copie, etc.) ;
- l'état du transfert ;
- le type des données transférées.

Un objet de type DragIcon est en fait composé de 3 icônes : l'icône source, l'icône d'état et l'icône d'opération. La première reste la même pendant tout le glisser, elle représente le type des données transférées. Les deux autres peuvent changer au cours du glisser. Elles illustrent respectivement le passage de l'objet source au-dessus de zones cibles potentielles et le type d'opération à effectuer.

Ainsi, la personnalisation du *feedback* de glisser peut-elle se faire à différents niveaux : pour une modification globale du *feedback*, on choisira d'agir directement sur les objets DragIcon d'une application, et pour une modification plus spécifique à une opération de *drag-and-drop* particulière, on changera des objets dynamiquement en utilisant les objets de type DragContext et les *callback* associées.

3.3. Microsoft Windows

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)* [15][16] 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() (FIG. 8 et FIG. 9).

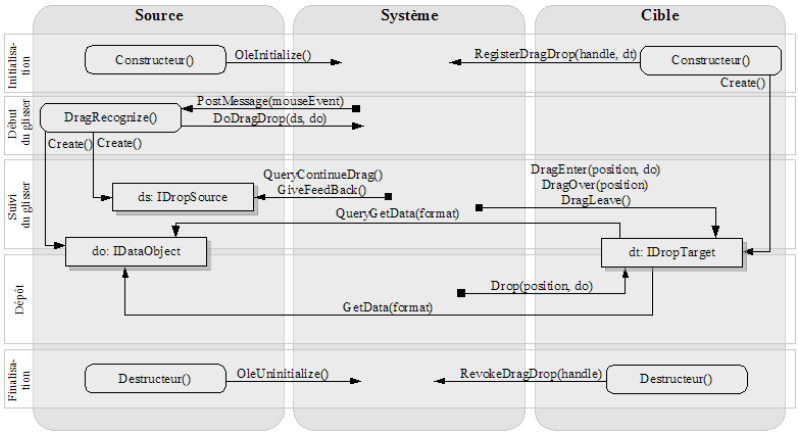

FIG. 8 – *Fonctionnement du* drag-and-drop *d'OLE.*

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*).

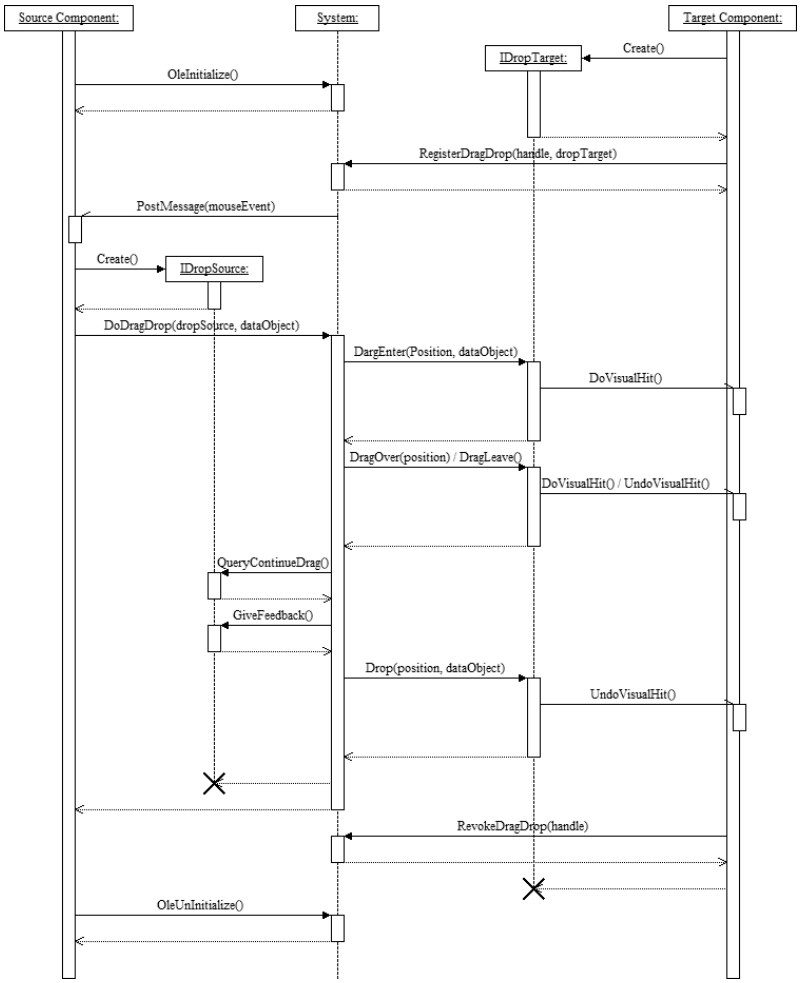

FIG. 9 – *Diagramme de séquence d'un* drag-and-drop *OLE.*

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 cible est notifié de l'avancement de l'opération□ lorsque le curseur 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 curseur 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().

3.4. Java Swing

Le modèle d'évènements de Java étant basé sur le principe de délégation, la gestion des évènements est réalisée par des objets dédiés [5]. 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 (FIG. 10 et FIG. 11).

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 glisser-déposer : 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 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 curseur 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().

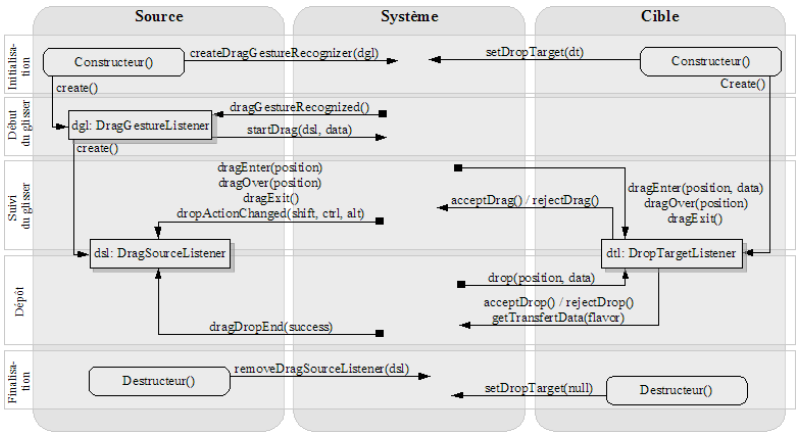

FIG. 10 – *Fonctionnement du* drag-and-drop *de Java/Swing*

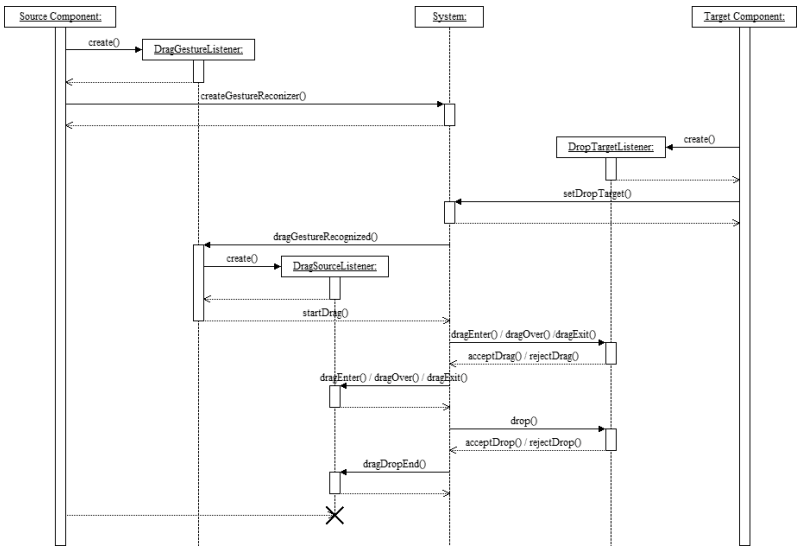

FIG. 11 – *Diagramme de séquence d'un* drag-and-drop *Java*

4. LES STRUCTURES DE DONNEES ET LES 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).

4.1. 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.

4.2. X-Window

Sous X-Window/Motif, il existe différentes possibilités de communication lors d'un *drag-and-drop*. Le transfert des données peut se faire en s'appuyant sur le mécanisme de *X-Selection* (il passe alors par le serveur X) ou bien faire l'objet d'un protocole dédié d'échange directement entre deux clients X comme c'est le cas dans le mécanisme de *drag-and-drop* décrit dans la partie 3.

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.

Bien que ce ne soit pas le protocole utilisé dans le mécanisme de *drag-and-drop* décrit dans la partie 3, son orientation plus généraliste mérite d'être mentionnée. Malheureusement, une fois énoncé le principe de fonctionnement de base qui peut paraître simple, les détails du mécanisme font ensuite apparaître un nombre important de complications techniques [7]. Pour masquer cette complexité, le comportement par défaut de certains *widgets* Motif standard gère une bonne partie de ces complications. Cependant, implémenter un *drag-and-drop* sur des *widgets* pour lesquels le

mécanisme n'est pas déjà en partie implémenté replace le programmeur face à cette complexité et compromet l'issue et la pérennité de son implémentation.

4.3. 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.

4.4. 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 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)*.

5. 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 multi-support.

Pour faciliter la discussion, nous numérotons ici les cinq principales étapes mises en évidence dans FIG. 4, FIG. 6, FIG. 8, FIG. 10 de la manière suivante : (1) Initialisation, (2) Début de glisser, (3) Suivi du glisser, (4) Déposer, (5) Finalisation.

5.1. 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*. Sous X/Motif au contraire, le modèle prévoit deux protocoles : le protocole dynamique et le protocole préenregistré. La différence majeure entre ces deux protocoles réside au niveau de la callback (XmNDragProc()) qui permet de gérer l'effet de *drag-under* et qui ne peut exister que dans le cas du protocole dynamique. Ainsi, X/Motif peut-il permettre un effet de *drag-under* sophistiqué avec le protocole dynamique ou pas d'effet de *drag-under* du tout avec le protocole préenregistré.

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-il 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 (Option sur Mac-OS-X/Carbon) ou MAJ ou CTRL+MAJ 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 évènementiels de Java/Swing et Mac-OS-X/Carbon.

Lors de l'étape 4 de déposer, on retrouve un point commun d'un système à l'autre : la cible est notifiée en premier, accepte ou refuse le déposer, et c'est seulement ensuite que le composant source est notifié de la fin du *drag-and-drop*. Cependant, le transfert des données ne présente pas un schéma similaire pour tous les systèmes : il est intégré dans la notification du déposer pour MS-Windows/OLE et Java/Swing mais c'est une étape à part entière dans X/Motif (elle a lieu dans ce dernier cas après les notifications 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 (voir Section 3). 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 que nous présenterons dans la Section 3. La discussion de ces différences dépasse le cadre de cet article. 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.

5.2. 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 (le seul écart à cette règle est X/Motif dans lequel cette étape n'est tout simplement pas prévue).

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. X/Motif diffère également des autres en cela que c'est un clic sur le bouton du milieu de la souris qui permet de démarrer le glisser. 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.

5.3. Intégration des techniques avancées

Le protocole de *drag-and-drop* devra évoluer pour s'adapter aux nouvelles exigences des différentes extensions décrites dans la première partie. Pour permettre l'intégration de ces techniques, il faudra 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 évènementielle 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 ont d'ores et déjà été proposées, notamment dans le système PointRight [11]. 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 du tout 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 5. 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 liste des nouvelles contraintes est longue et la construction d'un nouveau modèle de *drag-and-drop* ne se fera pas en un jour. Parmi les modèles existants, un certain nombre de mécanismes offrent déjà des ouvertures et des possibilités d'intégration. La délégation du suivi du *drag-and-drop* à un tiers tel que le DragManager sous Mac OS X/Carbon, la délégation de la gestion *drag-under* et *drag-over effect* à des gestionnaires d'évènements dédiés dans Java/Swing, ou à des objets fantômes dans X/Motif ou Mac OS X/Carbon, sont autant de pièces à considérer dans le puzzle. L'utilisation des principes d'interaction instrumentale [2][3] permettrait certainement de construire un modèle cohérent, suffisamment ouvert pour exprimer tous les besoins des nouveaux modèles de *drag-and-drop.*

6. CONCLUSION

Les différents modèles d'implémentations du *drag-and-drop* ont beau être relativement variés, aucun d'entre eux ne permet d'intégrer proprement les extensions telles que le *pick-and-drop,* le *drag-and-throw,* ou *l'hyperdragging*. Une évolution radicale est incontournable car actuellement, comme le montre la mosaïque de techniques présentées, l'ajout des nouveautés dans les systèmes de fenêtrage existants relève plus d'un processus ad hoc que d'une intégration réelle.

Le défi est de taille pour plusieurs raisons. Pour commencer, il s'agit d'homogénéiser les protocoles de transferts, les modèles d'émission et de traitement des évènements. En effet, comment un *drag-and-drop* intégré pourrait-il devenir distribué autrement ? Si la source se trouve sous MS Windows et que l'objet est déplacé sur une cible gérée par une machine sous Mac OS X par exemple, l'échange pourra avoir lieu uniquement si un protocole abstrait de la plateforme dirige la communication. Or le nombre de tentatives d'homogénéisation avortées

par le passé n'est pas de très bon augure. On peut cependant espérer que si l'homogénéisation n'a pas encore eu lieu c'est seulement par manque de maturité du domaine et que le temps devrait venir à bout de ce problème.

Mais il y a plus à faire encore pour réussir l'intégration des extensions du *drag-and-drop* qui ont émergé ces dernières années, poussées par des machines de plus en plus connectées, et aux capacités de représentation de l'information de plus en plus performantes. En effet, ces modèles requièrent un pouvoir d'expression accru tant du point de vue de la présentation (variété des *feedbacks*) que du traitement de l'information (variété du contenu échangé).

Or, la relève du *drag-and-drop* constitue un réel défi car il faut en somme, pour cette évolution, un modèle universel plus simple et plus ouvert à la fois que tout ceux qui existent pour le moment.

7. POUR ALLER PLUS LOIN

A l'adresse http://www.lirmm.fr/vag/dragging/, les lecteurs désirant plus de détails pourront trouver des démonstrations de la plupart des évolutions du *drag-and-drop* présentées dans cet article, des implantations du *drag-and-drop* dans les différents systèmes présentés ici, ainsi que les schémas de cet article en haute résolution.

8. REFERENCES

[1] Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P. Bederson, B., and Zierlinger, A. Drag-and-Pop and Drag-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-operated Systems. In Proceedings of Interact 2003. 2003. Zurich Switzerland.

[2] Beaudouin-Lafon Michel. Instrumental interaction : An interaction model for designing post-WIMP user interfaces. In Proc. ACM Conference on Human Factors in Computing Systems (CHI'2000), The Hague - Netherlands, April 2000. ACM Press.

[3] Beaudouin-Lafon Michel, Mackay Wendy. Reification, Polymorphism and Reuse: Three Principles for Designing Visual Interfaces. Proc. Advanced Visual Interfaces. AVI 2000, Palerme (Italie), Mai 2000, ACM Press, pp 102-109.

[4] Cocoa : Drag and Drop http://developer.apple.com/documentation/Cocoa/Conceptual/DraganddDrop/index.html

[5] De Lisa Gene. How to drag and drop with Java. http://www.javaworld.com/javaworld/jw-03-1999/jw-03-dragndrop.html

[6] Drag Manager Programmer's Guide. http://developer.apple.com/documentation/Carbon/Conceptual/DragMgrProgrammersGuide/DragMgrProgrammersGuide.pdf

[7] Fountain Antony, Huxtable Jeremy, Ferguson Paula & Heller Dan, Volume 6A: Motif Programming Manual, 2nd Edition 2002 , O'Reilly Associates.

[8] Geissler, J., Shuffle, Throw or Take It! Working Efficiently with an Interactive Wall, in Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems (Summary).□1998. p. 265-266.

[9] Hascoët, M. Throwing models for large displays. In Proceedings of the conference HCI'2003. 2003. Bath, GB: British HCI Group.

[10] Horn Bruce. On Xerox, Apple and progress. http://www.apple-history.com/frames/body.php?page=gui_horn1.

[11] Johanson B., Hutchins, G., Winograd, T., and Stone, M. (2002a) PointRight: Experience with Flexible Input Redirection in Interactive Workspaces. In Proc. UIST '02, pp. 227–234.

[12] Rekimoto J. and Masanori Saitoh, "Augmented Surfaces: A Spatially Continuous Workspace for Hybrid Computing Environments", Proceedings of CHI'99, 1999.

[13] Rekimoto, J., Pick-and-Drop: A Direct Manipulation Technique for Multiple Computer Environments, in Proceedings of the ACM Symposium on User Interface Software and Technology.□1997.

[14] Rogers, Rosemary. The mouse site. http://sloan.stanford.edu/MouseSite/

[15] The Microsoft Developer Network. http://msdn.microsoft.com/

[16] Wideman Graham. OLE Drag and Drop Theory. http://www.wideman-one.com/gw/tech/Delphi/dragdrop/DragDropTheory.htm