Pointeurs multiples : étude et implémentation

Eric Lecolinet

GET/ENST – CNRS UMR 5141
Dept. INFRES, 46 rue Barrault
75013, Paris, France
eric.lecolinet@enst.fr - www.enst.fr/~elc

RESUME
Cet article décrit d'abord les principaux cas d'interaction multi-pointeurs et propose une notation synthétique (nommée UDP/C) pour faciliter leur classification et la mise en évidence de leur ressemblances ou dissemblances. Une solution technique utilisant deux types d'outils complémentaires est ensuite décrite. Cette double technologie, qui permet de répondre à la plupart des cas existants, repose actuellement sur le système graphique X-Window et la boîte à outils Ubit.

MOTS CLES : multipointeurs, interaction bi-manuelle, single display groupware, espaces de travail virtuels multi-écrans, collecticiels, notation UDP/C, toolkit Ubit.

ABSTRACT
This paper describes the main cases of multiple pointer interaction and proposes a new notation (named UDP/C) for classifying and comparing these systems. A technical solution that makes use of two complementary tools is presented in the second part of the paper. This implementation can support most cases of multiple pointer interaction. It is currently based on the X-Window windowing system and the Ubit toolkit.

KEYWORDS : multiple pointers, bi-manual interaction, single display groupware, multiple screen workspaces, synchronous groupware, UDP/C notation, Ubit toolkit.

INTRODUCTION

Depuis la popularisation de la souris par le Macintosh dans les années 80, un environnement informatique standard est composé d'une unité centrale, d'un (ou deux) écran(s), d'un clavier et d'un ou plusieurs dispositifs de pointage (souris, « trackpad », tablette, etc.) contrôlant un seul et unique pointeur à l'écran. Cette situation n'a guère évolué en ce qui concerne les environnements standard bien que de nombreux cas d'utilisation de pointeurs multiples aient été proposés par les laboratoires de recherche en interaction homme-machine. Cet article décrit d'abord les principaux cas d'interaction multi-pointeurs et propose une notation synthétique pour faciliter leur classification et la mise en évidence de leur ressemblances ou dissemblances. Une solution technique utilisant deux types d'outils complémentaires est ensuite décrite. Cette double technologie, qui permet de répondre à la plupart des cas existants, repose sur le système graphique X-Window mais pourrait être généralisée à d'autres types de plateformes.

INTERACTION MULTI-POINTEURS

Cette appellation générique recouvre en réalité des cas de figures assez différents. Comme nous le verrons ensuite, plusieurs de ces cas peuvent en fait être pris en charge par le même type de technologie. La suite de cette section liste les cas principaux.

Notation UDP/C

Dans la suite de l'article, nous utiliserons les abréviations suivantes pour simplifier l'expression : **U** pour utilisateur, **C** pour unité Centrale, **D** pour Dispositif d'affichage (ou « Display ») et **P** pour Pointeur. Par D nous entendons un ou plusieurs écrans gérés par une même carte graphique (ces deux cas n'entraînant pas de différence majeure en ce qui concerne l'interaction ni l'architecture logicielle). Un pointeur P peut être contrôlé par un ou plusieurs dispositifs physiques fonctionnant simultanément (par exemple une zone tactile sur un portable et une souris auxiliaire). Un pointeur a une représentation permanente à l'écran. Nous introduisons enfin la quantité **P/** ou **PPD** qui représente le nombre de Pointeurs Par Dispositif d'affichage. Cette quantité sera affinée par la suite.

Cas standard 1-1-1

Ce cas représente la situation habituelle où un utilisateur U interagit avec un dispositif d'affichage D et un seul et unique pointeur P via son unité centrale C. Habituellement, les applications interactives avec lesquelles U interagit s'exécutent sur C. Nous désignons ce cas de figure par la suite: **UDP/** = **1-1-1** signifiant : 1 ***U****tilisateur*,
1 ***D****isplay*, 1 ***P****ointeur par Display*.

Ce cas présente des variantes intéressantes. Certains systèmes permettent en effet d'interagir avec des applications qui ne s'exécutent pas sur l'unité centrale C qui contrôle P et D mais sur des Ci distantes liées à C via un réseau. C'est le cas du système de multi-fenêtrage X-Window, de RDP (Remote Display Protocol), initialement conçu pour MS Windows et de VNC [23]

(qui permet également la réplication de fenêtres comme nous le verrons ensuite). Nous identifions cette variante par la notation : **UDP/C** = **1-1-1-***, l'étoile signifiant que plusieurs **C** interviennent.

Interaction bi-manuelle 1-1-2

L'interaction bi-manuelle repose sur une idée qui peut sembler a priori assez naturelle : puisque nous effectuons nombre d'actions en utilisant simultanément les deux mains (essayez de couper votre steak avec une seule main !), l'utilisation simultanée de deux souris indépendantes (c'est-à-dire de deux pointeurs) devrait également faciliter l'interaction homme-machine. Cette idée a en particulier été proposée pour interagir avec des outils transparents (« see-through tools ») [2,3,14], la main non dominante (ou « main fourchette » pour reprendre notre métaphore) servant à positionner l'outil, l'autre main (« couteau ») permettant de sélectionner un interacteur placé sur l'outil et d'interagir avec ce dernier. Cette technique d'interaction multi-pointeurs correspond à la suite **1-1-2** (1 *u*, 1 *d*, 2 *ppd*) dans notre notation.

Ce type d'interaction est malheureusement rarement disponible car les systèmes de multifenêtrage et les boîtes à outils standard ne proposent aucun moyen simple d'en faire usage. Deux types d'implémentations peuvent être envisagées, soit directement au niveau applicatif, soit via un serveur. Dans le premier cas, l'application utilise une bibliothèque permettant de gérer des dispositifs d'interaction supplémentaires [8,12]. Dans le second cas, c'est un serveur de gestion d'événements qui gère ces dispositifs et envoie les événements ad hoc aux applications. Ces dernières (ou la boîte à outils utilisée pour les construire) doivent alors être capables de distinguer les sources d'événements. C'est l'approche que nous présenterons dans la seconde partie.

L'utilisation de deux (ou plusieurs) pointeurs sur un même dispositif d'affichage pose en fait de multiples problèmes. Il est tout d'abord nécessaire de représenter à l'écran le deuxième (ou nième) pointeur(s). Aucun système de fenêtrage existant n'est prévu pour le faire. Il est donc nécessaire d'utiliser des solutions détournées qui ne peuvent garantir les mêmes performances que pour le pointeur « natif » (même si celles-ci peuvent être tout à fait acceptables comme nous le verrons par la suite). De ce point de vue, la solution « serveur » offre un avantage évident : le ou les pointeurs supplémentaires sont alors gérés de manière uniforme sur tout le dispositif d'affichage par ce serveur (au lieu d'être gérés par chaque application multi-pointeurs). Ceci peut également permettre un meilleur contrôle de la cohérence de l'interaction lorsque plusieurs applications multi-pointeurs s'exécutent simultanément. Notons enfin que certains systèmes de fenêtrage permettent de gérer des dispositifs d'interaction auxiliaires en entrée (par exemple l'extension Xinput de X-Window) mais ne gèrent pas pour autant la représentation de plusieurs pointeurs à l'écran ni la gestion de la logique de l'interaction.

Ce dernier point est d'ailleurs probablement le plus difficile. Que signifie en effet la notion de « focus », de « sélection », d'opérations interdites (« grisées ») lorsque l'on dispose de plusieurs pointeurs ?

Même en l'absence d'opérations de nature modale ou liées à des états particuliers (comme la sélection ou le focus) il n'est pas certain que la boîte à outils utilisée pour créer les applications supporte correctement l'utilisation simultanée de plusieurs pointeurs (même lorsqu'elle est étendue par un module gérant plusieurs sources d'événements). L'implémentation des interacteurs peut dépendre de variables d'état ne supportant qu'un seul flux d'interaction. Les problèmes typiques concernent l'utilisation des systèmes de menus (peut-on ouvrir plusieurs menus simultanément et sélectionner leurs items indépendamment ?), les ascenseurs (peut-on contrôler indépendamment un ascenseur vertical et un ascenseur horizontal), l'édition de texte, etc.

Single display groupware N-1-N

Ce cas de figure est relativement récent car résultant de la popularisation des techniques de vidéo projection. Dans cette situation, plusieurs utilisateurs interagissent avec un seul et même Dispositif d'affichage mais au moyen de plusieurs dispositifs d'interaction indépendants (et Pointeurs correspondants). Ce type d'interaction semble être particulièrement approprié pour l'enseignement (en particulier avec des enfants [26]), pour les présentations interactives, les séances de « brainstorming », etc. Ce cas est en fait assez proche du précédent : l'espace d'affichage est unique, il y a plusieurs pointeurs, mais ils sont manipulés par des intervenants différents. Nous l'identifierons par la suite par **N-1-N** (N *u*, 1 *d*, N *ppd*).

Sur le plan purement technique, il s'agit essentiellement d'une généralisation du cas précédent dans la mesure où il ne suffit plus de gérer 2 mais N pointeurs contrôlés par des sources distinctes. Les différences sont plutôt au niveau sémantique : dans le cas de l'interaction bi-manuelle, on peut par exemple faire l'hypothèse que les actions provoquées par les deux pointeurs sont complémentaires (un des pointeurs étant « dominant », et gérant le focus, la section, etc. l'autre n'ayant qu'une fonction auxiliaire). Cette hypothèse devient caduque si chaque pointeur correspond à un utilisateur différent. Une solution peut alors consister à éviter les interactions faisant intervenir des modes ou des variables d'état [26]. Inversement, les états peuvent être dupliqués pour chaque pointeur/utilisateur. Par exemple Myers [20] propose un logiciel de dessin où chaque utilisateur peut choisir la couleur et autres caractéristiques du tracé au

moyen des mêmes interacteurs. Ceci risque cependant de rendre l'interface assez complexe, voire confuse, puisque l'état des interacteurs n'est plus global mais dépendant de chaque pointeur (des symboles graphiques sont utilisés dans ce système pour indiquer les correspondances entre interacteurs à états et pointeurs). Il semble difficile d'utiliser une boîte à outils standard, même augmentée, dans ce dernier cas (le système évoqué utilise Amulet [19]). Même une boîte à outil spécialisée pourra difficilement gérer tous les cas de conflit ou d'ambiguïté de manière générique : nombre de ces cas devront être spécifiquement pris en compte par l'application.

Notre notation permet aussi de prendre en compte les variantes suivantes :

a) chaque participant bénéficie de l'interaction bi-manuelle : notation **N-1-2N**

b) plusieurs unités centrales entrent en jeu : **N-1-N-*** C'est par exemple le cas de figure du projet *Pebbles* [20] où un écran partagé est géré par l'intermédiaire de plusieurs PDAs.

En conclusion, il est difficile d'apporter une solution complètement générale à ce cas de figure sans remettre en cause certains des paradigmes usuels de la « métaphore du bureau ». De plus, certains problèmes purement technologiques peuvent difficilement être résolus sans utiliser des boîtes à outils spécialement conçues à cet effet. Il est néanmoins possible de créer des applications intéressantes en augmentant des boîtes à outils standard (par exemple *Icon* [8] ou *Jazz MID* [12] qui sont basés sur Java/Swing) à condition d'accepter certaines imperfections en ce qui concerne la gestion de l'interaction.

« Mighty-mouse et Point-right » *-N-1

Ce cas de figure correspond – paradoxalement – à la fois à une simplification et à une généralisation du cas précédent. Le type de situation est le même : une séance de présentation et/ou de travail collaboratif où les participants affichent leurs documents sur de grands espaces d'affichage (vidéoprojecteurs, écrans plasma, *SMARTBOARD* [25]). Mais, à la différence du cas précédent, plusieurs Dispositifs d'affichage sont utilisés (chacun étant contrôlé par une unité Centrale différente). La restriction vient du fait que plusieurs participants ne peuvent pas utiliser *en même temps* le même D.

La situation est en quelque sorte inverse de ce qui a été présenté jusqu'à maintenant : plusieurs D sont gérés simultanément mais pas par plusieurs utilisateurs. Plus précisément, il semble que dans le cas de *Mighty mouse* [4] un utilisateur (ou plusieurs mais *successivement*) peut gérer l'ensemble des D, tandis que dans le cas de *Point-right* [13] plusieurs utilisateurs peuvent interagir simultanément à condition qu'ils ne sélectionnent pas le même D.

Mis à part les différences techniques, ces deux cas de figures sont en réalité très proches et c'est pourquoi nous les regroupons dans une même section. Le premier cas (*Mighty mouse*) correspond à la notation **1-N-1** (1 *u*, N *d*, 1 *ppd*) le second (*Point right*) à la notation **N-N-1** (N *u*, N *d*, 1 *ppd*). On remarque que dans les deux cas, le nombre de PPD (pointeur par display) est le même et égal un 1 (impossibilité d'avoir deux pointeurs sur le même D) contrairement aux cas précédents où le PPD était égal à 2 (interaction bi-manuelle) ou à N (single display groupware).

Assez logiquement, ce cas de figure induit des contraintes technologiques radicalement différentes. Il n'impose aucune contrainte sur les applications et boîtes à outils utilisées (puisque deux pointeurs ne peuvent jamais interagir avec la même application). Comme les systèmes proposés ne prennent en compte que les événements souris et clavier de bas niveau ils sont multi-plateformes (MS Windows, X-Window, MacIntosh dans un des cas) et fonctionnent avec des matériels hétérogènes. Ces systèmes permettent enfin de considérer un ensemble de dispositifs d'affichage gérés par des unités centrales distinctes comme un espace de travail virtuel unifié. Des *topologies* d'espaces d'affichage D[n] peuvent être définies pour assurer une continuité logique : lorsque le pointeur « sort » d'un Di il acquiert le contrôle du Di+1 défini par rapport à cette topologie (*Point right* [13]).

Une extension évidente, envisagée par les auteurs de ces deux systèmes, serait de combiner la possibilité de gérer à la fois plusieurs D et plusieurs P par D. Nous la noterions : **N-N-N** (synthèse de ce cas de figure avec les cas précédents).

Logiciels collaboratifs synchrones

Nous regroupons sous ce terme générique les outils collaboratifs permettant à plusieurs utilisateurs d'afficher et/ou d'éditer simultanément des documents partagés en temps « plus ou moins réel » depuis des environnements informatiques localisés en des lieux différents. Etant donné la richesse et l'étendue du domaine nous nous en tiendrons à cette définition générique.

Les premiers systèmes présentés au début de cet article considéraient plusieurs sources d'événements (venant de plusieurs ou du même utilisateur) mais partageant le même dispositif d'affichage. Les systèmes suivants permettaient d'utiliser plusieurs dispositifs d'affichage, mais tous les participants étaient néanmoins censés pouvoir en consulter l'ensemble. Le dernier cas nécessite à la fois la gestion de plusieurs sources d'entrées (sur des unités centrales distinctes) et de plusieurs dispositifs d'affichage. De plus, ces derniers sont physiquement séparés (chaque utilisateur ne voyant que son propre D), ce qui introduit une dimension supplémentaire. Ceci impose l'utilisation de techniques de réplication pour

que l'objet du travail collectif soit visible de tous les participants sur leur D respectif. Il est d'autre part nécessaire que chaque participant ait conscience de l'activité des autres participants.

Dans le cas d'applications collaboratives synchrones à fort niveau de couplage, les problèmes liés à la gestion de l'interaction sont en fait assez proches de ceux rencontrés dans le cas du single display groupware (en ce qui concerne la gestion des modes et des états). Cette similarité est encore plus grande si l'architecture est centralisée (un seul programme contrôle les réplications de l'interface sur tous les D). Ceci peut permettre, comme nous le verrons dans la deuxième partie, d'utiliser des outils logiciels communs dans les deux cas de figures. Nous ne détaillerons pas davantage les fonctionnalités sophistiquées de certains types de collecticiels (comme par exemple la gestion des droits d'accès aux différentes parties des documents partagés), ni les modèles et architectures logiciels, (qui ont fait l'objet de nombreuses publications comme par exemple [7, 10,17] etc.) D'autre part les collecticiels à faible niveau de couplage (type Lotus Notes) n'entrent pas dans le cadre de cette étude.

Les objets peuvent être partagés à divers niveaux d'abstraction, ceci entraînant divers types de réplication des interfaces sur les écrans des participants [7]. Dans le cas le plus simple (comme par exemple VNC [23]) un espace de travail complet (et les fenêtres des applications concernées) est visuellement répliqué à l'identique sur tous les dispositifs d'affichage. Ce type de technologie a l'avantage d'être indépendant des applications utilisées, mais celles-ci ne peuvent être manipulées simultanément par plusieurs participants. Les collecticiels plus sophistiqués utilisent généralement des boîtes à outils spécifiques (par exemple *GroupKit* [24]) qui permettent une bien plus grande flexibilité tant en entrée qu'en sortie.

Dans tous les cas, il est nécessaire de répliquer certaines vues, même si elles n'apparaissent pas exactement de la même façon, sur tous les dispositifs d'affichage et d'indiquer aux utilisateurs distants les lieux d'interaction des autres utilisateurs. Ceci est généralement effectué à l'aide de *télépointeurs* (qui sont des réplications du pointeur de l'un des utilisateurs du système partagé). Contrairement aux *multi-pointeurs* que nous avons jusqu'à présent considérés, les télépointeurs ont pour caractéristique de ne pas pouvoir être contrôlés *en entrée* par les utilisateurs. Nous noterons ce cas de figure de la manière suivante : **N-N-(1,N-1)**. Ceci signifie que chacun des N utilisateurs contrôle un pointeur en entrée sur son propre D et peut voir les réplications des N-1 pointeurs des autres utilisateurs.

Le fait que les vues ne soient pas nécessairement identiques sur tous les D implique deux conséquences importantes. D'une part, chaque vue peut mettre en évidence les divers modes et états liés à l'interaction d'un utilisateur sur son propre D. Ceci simplifie les problèmes de représentation d'états concourants propres au *single display groupwar*e (mais sans les résoudre totalement car l'utilisateur a besoin de savoir ce que font les autres participants et donc, éventuellement, de connaître ces états). D'autre part, la réplication des pointeurs distants ne peut plus se faire de manière triviale par simple transfert des coordonnées écran du D où le pointeur est actif vers les autres. Les fenêtres de l'application partagée peuvent être disposées à des endroits différents de l'écran selon les D, avoir des tailles différentes ou ne pas contenir les mêmes interacteurs [24]. Ceci signifie que les *télépointeurs* ne sont pas de simples multi-pointeurs : ils ne peuvent être gérés par des serveurs génériques liés à chaque D, mais doivent être directement contrôlés par l'application partagée.

Cas	**Notation UDP/C**
a) Cas standard *avec exécution à distance*	1-1-1 *1-1-1-**
b) Interaction bi-manuelle	1-1-2
c) Single display groupware *avec exécution à distance*	N-1-N *N-1-N-**
d) Mighty-mouse	1-N-1
d') Point-right	N-N-1
d'') Point-right généralisé	N-N-N
e) Collecticiels	N-N-(1,N-1)

Conclusion de la première partie

Nous avons sommairement présenté les divers cas d'utilisation de pointeurs multiples en essayant de mettre en évidence les similarités et différences. Ce travail a pour but de permettre l'implémentation de techniques génériques pouvant être réutilisées (ou combinées) pour fournir les services nécessaires aux différents cas de figure précédemment cités. Il est d'ailleurs à la base des solutions proposées dans la seconde partie. Nous proposons également une notation synthétique qui tente de mettre en évidence les dimensions principales du problème. Nous en avons recensé trois :

- le nombre d'Utilisateurs
- le nombre de Dispositifs d'affichage (*Displays*)
- le nombre de Pointeurs actifs et passifs (télépointeurs) par Dispositif d'affichage

Cette notation est bien sûr imparfaite et ne peut rendre compte à elle seule de tous les aspects mis en jeu. Elle nous semble cependant assez bien montrer les similarités ou dissimilarités des divers cas de figure tout en étant très concise. Elle pourrait de plus être affinée pour indiquer la nature des *Displays*, par exemple pour

préciser si ce sont de simples écrans de PC, des PDAs ou de grands écrans partagés. Notons enfin que nous n'avons pas traité les aspects liés au clavier dans cette étude.

SOLUTION PROPOSEE

Serveur multipointeurs

Le serveur multi-pointeurs (ou **UMS** pour *Ubit Mouse Server*) permet de gérer plusieurs pointeurs simultanément sur un même dispositif d'affichage. Il fonctionne en collaboration avec le serveur X dont il étend les fonctionnalités (les implémentations présentées ici étant basées, pour l'instant, sur le système de multi-fenêtrage X- Window). Il peut fonctionner avec des applications X quelconques, mais certaines fonctionnalités avancées ne peuvent être exploitées que par des applications « compatibles », comme nous le verrons ultérieurement. Il est à la base de l'implémentation des cas de figure b), c) et d*) du précédent tableau (conjointement avec d'autres outils dans certains cas).

L'UMS est basé sur la notion de *source d'événements* et de *flux d'événements*. Une *source d'événements* correspond à un dispositif physique donné. Le système est actuellement capable de prendre en compte des sources d'événements de type souris série ou *MIMIO* (un dispositif de capture de tracés sur grand écran). D'autres types de dispositifs seront ultérieurement pris en compte, éventuellement via l'utilisation de l'extension *Xinput*. L'UMS n'impose pas de limitation sur le nombre de sources d'événements pouvant être gérées simultanément.

Un *flux d'événements* peut être vu comme un « canal » d'envoi d'événements aux applications. Le système peut supporter un nombre arbitraire de flux, chacun possédant sa propre identification. La correspondance entre les sources et les flux d'événements se fait au lancement du serveur via des options appropriées. Ce mécanisme simple permet de préserver l'identité des événements produits par les différentes sources (sauf bien sûr lorsque plusieurs sources sont associées au même flux d'événements, ce que l'UMS permet également de faire).

Afin de préserver la compatibilité avec les applications X standard, les événements sont envoyés à celles-ci via le serveur X (bien qu'il soit également possible de les envoyer directement via une connexion *socket*). Une application standard traitera donc de la même manière les événements initialement produits par le serveur X ou par le serveur UMS, puisque ce dernier utilise ensuite le serveur X comme intermédiaire. Une application « compati-ble avec l'UMS » pourra de plus déterminer l'identifiant du flux dont l'événement provient. Ceci est rendu possi-ble par une convention de codage des événements X. Cette modification n'intervient que pour les événements envoyés aux applications « compatibles ». Elle ne peut donc pas perturber le fonctionnement des applications standard ignorant cette convention. Inversement, une application « compatible » réagira normalement aux événements provenant d'un serveur X standard.

A chaque flux correspond un pointeur qui est affiché sur le dispositif d'affichage par le serveur UMS. X-Window ne fournissant pas de moyen de gérer plusieurs pointeurs, ceux-ci sont simulés par la création de petites fenêtres sans bordure sur lesquelles un pointeur est dessiné. Leur couleur peut être spécifiée au lancement du serveur. Cette technique est tout à fait satisfaisante, tant en ce qui concerne les performances que l'effet visuel (d'autant qu'il est possible de créer des fenêtres de forme quelconque). Son principal inconvénient est qu'elle entraîne de multiples rafraîchissements des fenêtres des autres applications. Etant donné la taille du pointeur (quelques pixels), ceci est généralement peu gênant sauf dans quelques cas particuliers (applications mal programmées ayant des fenêtres de grande taille qui sont entièrement rafraîchies sans « double buffering » ou lorsqu'une application tarde à se réafficher pendant l'exécution d'une action très consommatrice en CPU).

Pour permettre une meilleure généricité le serveur fournit également un pseudo-flux qui contrôle le flux d'événements natif du serveur X-Window. Il est ainsi possible d'associer une source d'événements gérée par l'UMS à ce pseudo-flux afin qu'elle contrôle le pointeur natif du serveur X. Ceci permet d'utiliser des dispositifs d'interaction non standard comme un *MIMIO* pour contrôler le pointeur natif sans avoir à développer un pilote spécifique au serveur X.

L'UMS permet également de modifier certains types d'événements (par exemple pour simuler une souris à 2 ou 3 boutons sur un PDA en pressant les boutons physiques qui figurent sur ce dernier). Il est enfin pilotable à distance depuis une application située sur une autre unité centrale (celle-ci pouvant être un autre serveur UMS comme expliqué dans la section suivante).

Les correspondances entre les fonctionnalités du serveur et les cas de figures décrits dans la première partie sont résumées ci-après :

Cas b) Interaction bi-manuelle 1-1-2

l'UMS permet de gérer un second pointeur. Les événements émis par celui-ci pourront être identifiés par les applications « compatibles » afin de faire les traitements adéquats. Les autres applications ne pourront évidemment pas distinguer les deux pointeurs.

Cas c) Single display groupware N-1-N

l'UMS permet de gérer autant de pointeurs que nécessaire et d'identifier leurs flux. Ce cas de figure ne nécessite pas obligatoirement d'utiliser des applications

« compatibles ». On peut en effet imaginer une situation ou plusieurs utilisateurs voudraient afficher des documents complémentaires sur l'écran. L'UMS permettra à ces utilisateurs de manipuler simultanément leurs documents *respectifs* via des applications standard. Par contre les événements se « mélangeront » lorsque deux utilisateurs interagiront en même temps avec la même application (sauf si cette application est « compatible »).

Le cas **N-1-N-*** où chaque pointeur est contrôlé par une machine distante (par exemple un PDA comme dans le cas de *Pebbles* [20]) est également pris en compte par notre système. Chaque pointeur de l'UMS est alors contrôlé par une application distante s'exécutant sur un PDA (ou toute autre unité centrale). La connexion entre l'application et l'UMS se fait directement via une *socket* et un protocole de communication que le programmeur n'a pas besoin de connaître, une boîte à outils fournissant les abstractions nécessaires.

En outre, ce mécanisme peut être aussi utilisé pour contrôler les applications à distance sans manipuler de pointeur. L'UMS fournit un service « d'annuaire » permettant d'identifier les fenêtres par leur nom (ou par le texte contenu dans leur barre de titre) et de leur envoyer des pseudo-événements ou des messages textuels. Ceci permet de réaliser des *télécommandes*. Par exemple, la télécommande d'un navigateur Web pourrait activer les principaux boutons de cette application en leur envoyant des clics souris. Cette télécommande aurait seulement besoin de connaître le nom de l'application et la position des boutons relativement à la fenêtre de l'application (ce qui suppose que ceux-ci ne se déplacent pas, une hypothèse souvent vérifiée pour des raisons d'utilisabilité).

Cas d) Mighty-mouse 1-N-1

Plusieurs serveurs UMS peuvent être « chaînés » entre-eux de manière à ce que le pointeur passe d'un dispositif d'affichage à un autre lorsqu'il atteint les bords de l'un d'entre-eux. La technique employée est la même que précédemment, chaque serveur UMS télécommandant celui du dispositif d'affichage suivant (la topologie étant spécifiée au lancement des serveurs). L'implémentation actuelle n'est pas centralisée : les événements sont envoyés de proche en proche par les serveurs. Les connexions entre serveurs sont lancées dynamiquement, en fonction des mouvements du pointeur sur les dispositifs d'affichage. Afin d'améliorer les performances, elles restent ensuite actives jusqu'à l'arrêt des serveurs UMS.

Cas d') Point-right N-N-1
cas d'') Point-right généralisé N-N-N

La fonctionnalité de base d'un serveur UMS, qui est de gérer plusieurs pointeurs, est préservée lorsque plusieurs serveurs sont chaînés comme dans le cas précédent. Nous sommes donc en fait déjà dans le cas N-N-1 où plusieurs participants peuvent travailler simultanément sur plusieurs écrans (mais pas sur le même écran), et même dans le cas étendu N-N-N où cette contrainte est levée. Ce cas correspond à une généralisation du simple display groupware où plusieurs displays sont visibles et peuvent être contrôlés simultanément par tous les membres d'un groupe de travail.

Cas e) Collecticiels N-N-(1,N-1)

Le serveur UMS n'intervient pas dans ce cas de figure d'autres techniques devant alors être utilisées, comme par exemple celle qui est proposée dans la section suivante. Ceci résulte en particulier du fait que les télépointeurs dépendent des applications et ne peuvent être gérés génériquement par un serveur lié à un dispositif d'affichage.

Limitations

Actuellement seul le pointeur natif gère le focus clavier. Nous étudierons d'autres voies ultérieurement lorsque l'UMS sera également capable de prendre en compte les entrées de plusieurs claviers. L'implémentation actuelle nécessite donc d'être « polie » afin de fournir des services totalement équivalents aux cas d) et d') (bien qu'elle en soit une généralisation par ailleurs). Cette implémentation ne fonctionne qu'avec le système de fenêtrage X-Window, mais l'architecture retenue (un serveur auxiliaire collaborant avec le serveur X) devrait faciliter les portages, comparativement à une implémentation qui aurait modifié le serveur X.

D'autre part, la forme des pointeurs auxiliaires gérés par l'UMS ne peut être changée dynamiquement. Ce problème pourrait être résolu dans le cas des applications « compatibles » en enrichissant le protocole de communication de l'UMS. Enfin, l'UMS ignore les « grab souris ». Sans vouloir entrer dans des détails trop techniques, précisons que ce dernier point peut interdire une interaction correcte avec certaines applications. Ce problème se pose rarement (ou n'a que des effets mineurs) avec des programmes écrits à l'aide de boîtes à outils raisonnablement récentes (comme par exemple Qt, Gtk ou même Motif). Il n'est réellement gênant qu'avec des applications anciennes ou un peu particulières (par exemple, certains gestionnaires de fenêtres ou éditeurs de texte comme Emacs).

Néanmoins, cette approche marche bien et offre de bonnes performances dans la plupart des cas. Elle suppose cependant d'accepter les limitations précédemment citées. Celles-ci peuvent sembler raisonnables étant donné les possibilités offertes par ailleurs.

Boîte à outils Ubit

Ubit est une boîte à outils graphique qui a pour principale caractéristique d'être basée sur une

architecture « atomique » [15,16,28]. Au lieu d'être définis statiquement dans des classes qu'il est relativement difficile d'augmenter, sauf pour effectuer des modifications mineures, les widgets sont constitués de combinaisons d'objets élémentaires (ou *briques*). Ces briques peuvent être combinées dynamiquement pendant l'exécution avec une grande liberté afin d'assurer un maximum de flexibilité et de réutilisabilité des divers « services » fournis par le toolkit. Les widgets sont des combinaisons préétablies de briques qu'il est toujours possible de modifier dynamiquement à l'exécution. Ils sont pour cette raison appelés *« brickgets »*. Les briques sont des « petits » composants logiciels qui implémentent une fonction spécifique et bien définie. Elles se répartissent en six catégories : les composants d'affichage élémentaires (chaînes de caractères, images, symboles graphiques, etc.), les attributs graphiques (fontes, couleurs, décorations, etc.), les composants servant à spécifier le comportement des brickgets, les objets de call-back, les gestionnaires de rendu et de disposition spatiale et divers types de conteneurs qui servent à regrouper les autres objets et à créer les brickgets.

Une propriété intéressante de cette architecture est que tous les objets peuvent être partagés par plusieurs parents, y compris les brickgets et les hiérarchies de brickgets (Fig. 1). Ces derniers sont automatiquement répliqués visuellement dans tous leurs parents (sauf dans le cas des fenêtres telles que boîtes de dialogue où cela n'aurait pas de sens). Cette propriété est récursive : les enfants des brickgets répliquées sont eux-mêmes répliqués et ainsi de suite.

Cette propriété a récemment été étendue au cas des applications partagées, la boîte à outils *Ubit* étant désormais capable d'ouvrir plusieurs fenêtres simultanément sur plusieurs dispositifs d'affichage. Le partage d'objets simplifie considérablement l'écriture de telles applications, la spécification des liens de parenté permettant de répliquer automatiquement sur plusieurs écrans un intéracteur, ou une hiérarchie d'interacteurs, ou le contenu intégral d'une ou plusieurs fenêtres. Ces objets sont synchronisés automatiquement et supportent l'interaction multi-pointeurs (qu'ils soient ou non sur le même dispositif d'affichage).

Les briques élémentaires peuvent également être partagées par des fenêtres apparaissant sur plusieurs dispositifs d'affichage. Cette propriété n'est pas aussi anodine qu'elle le semble à première vue, ces dispositifs pouvant avoir des caractéristiques physiques différentes (par exemple une couleur codée sur 24 bits sur l'écran d'un PC et sur 15 bits sur l'écran d'un PDA). La « bonne » couleur sera néanmoins affichée dans les deux cas bien qu'elle soit contrôlée par un seul et même objet.

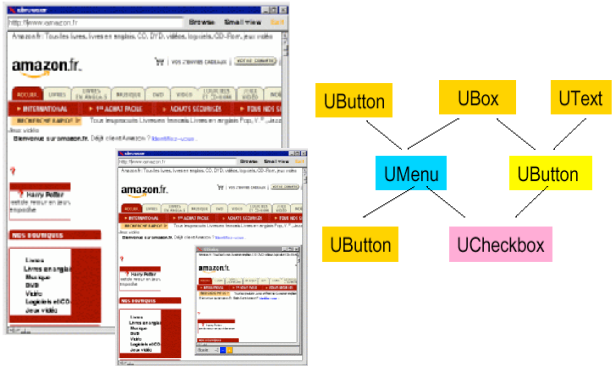

Figure 1 : à gauche : vues répliquées avec différents niveaux de zoom ; à droite : exemple de partage de *brickgets* dans un graphe d'instanciation.

Le fait que le contenu des fenêtres puisse être partagé sur divers dispositifs d'affichage, n'implique pas nécessairement un affichage identique. Ces fenêtres peuvent avoir été retaillées, avoir des coefficients de zoom différents, ou avoir été programmées à l'aide de spécifications conditionnelles (une propriété du toolkit qui permet de générer des vues différentes à partir d'une même spécification). La boîte à outils fournit un service de télépointage qui permet de répliquer les pointeurs actifs sur les autres dispositifs d'affichage en fonction de leur *position logique* (c'est-à-dire en tenant compte du brickget sur lequel ils se trouvent dans la vue d'origine et en essayant de déterminer où ils devraient être situés dans les autres vues).

La boîte à outil *Ubit* est évidemment compatible avec les serveurs UMS. Contrairement aux autres applications X, elle peut déterminer le flux d'origine des événements. Chaque flux est traité via une machine à états séparée. Toute application *Ubit* est donc susceptible de supporter l'interaction bi-manuelle ou le single display groupware. Il est ainsi possible d'interagir avec un système de menus avec plusieurs pointeurs, de manipuler simultanément plusieurs ascenseurs ou d'implémenter facilement des outils transparents (Fig. 2).

Conclusion de la seconde partie

L'utilisation conjointe de *l'UMS* et du toolkit *Ubit* permet donc de répondre aux cas **1-1-2** (interaction bi-manuelle) et **N-1-N** (single display groupware). Le cas **N-N-(1,N-1)** est résolu via l'implémentation de télépointeurs logiques, le partage d'objets facilitant par ailleurs l'écriture de ce type d'applications. L'utilisation d'une architecture centralisée ne peut évidemment pas résoudre tous les cas de collecticiels synchrones mais simplifie la gestion de la cohérence des vues et permet de bonnes performances lorsque l'interaction se limite à quelques dispositifs d'affichage avec un réseau performant.

Figure 2 : Un exemple d'application utilisant un Control menu [21] et un outil transparent qui sont manipulables en utilisant une ou deux mains.

ETAT COURANT ET TRAVAIL FUTUR

Le serveur *UMS* et le toolkit *Ubit* sont écrits en C++. Ils sont accessibles à l'URL www.infres.enst.fr/~elc/ubit ainsi qu'une vidéo et divers exemples. Les travaux futurs concernent l'amélioration des divers outils ainsi que le développement d'applications sophistiquées, afin de pouvoir évaluer les techniques proposées et de les améliorer. Par exemple, nous travaillons actuellement sur un système de partage de transparents, notes de cours et annotations entre un grand écran et des dispositifs de petite taille (PDAs, eBooks, etc.). Le portage des outils présentés vers d'autres plateformes est également envisagé.

BIBLIOGRAPHIE

1. Abdel-Wahab H., Kim O. *Xtv: a Framework for Sharing X- Window Clients in Remote Synchronous Collaboration*, Proc. IEEE Tricomm 1991, Communications for Distributed Software, Chappel Hill, 159-167.
2. Beaudouin-Lafon M. *The Architecture and Implementation of CPN2000, a Post-WIMP Graphical Application*, Proc. UIST 2000, ACM, 181-190.
3. Bier E.A., Stone M.C., Pier K., Buxton W., DeRose T.D. *Toolglass and Magic Lenses: The See-Through Interface,* Proc. SIGGRAPH 1993, ACM, 73-80.
4. Booth K.S. et al. *The "Mighty Mouse" Multi-Screen Collaboration Tool,* Proc UIST 2002, ACM, 209-212.
5. Chatty S., *Extending a Graphical Toolkit for Two-Handed Interaction,* Proc UIST 1994, ACM, 195-204.
6. Chung G., Dewan P. Flexible *Support for Application-Sharing Architecture*. Proc ESCSW, 2001, Kluwer, 99-118.
7. Dewan, P., Choudhary, R. *A high-level and flexible framework for implementing multiuseruser interfaces*. ACM Trans. on Information Systems, 1992, 10(4), 345-380.
8. Dragicevic P, Fekete J-D. *Input Device Selection and Interaction Configuration with ICON*, Proc. IHM-HCI 2001, Springer-Verlag, 543-548.
9. Fowler A. *A Swing Architecture Overview*. http://www.javasoft.com/products/jfc/tsc
10. Greenberg S, Roseman M. *Groupware Toolkits for SynchronousWork*. Trends in CSCW, Edited by Michel Beaudouin-Lafon, Wiley, 1996 .
11. Hill, R.D., Brinck, T., Rohall, S.L., Patterson, J.F., Wilner, W. *The Rendezvous architecture and language for constructing multi-user applications*. ACM ToCHI, 1994, 1(2), 81–125.
12. Hourcade J-P, Bederson B.B. *Architecture and Implementation of a Java Package for Multiple Input Devices (MID)*, Technical Report CS-TR-4018, 1999, Univ. of Maryland.
13. Johanson B., Hutchins G., Winograd T., Stone M., *PointRight: Experience with Flexible Input Redirection in Interactive Workspaces,* Proc. UIST 2002, ACM, 227-234.
14. Kurtenbach G., Fitzmaurice G.W., Owen R.N., Baudel T. *The Hotbox: Efficient Acces to a Large Number of Menu Items*. Proc CHI 1999, ACM, 231-237.
15. Lecolinet E., *A Brick Construction Game Model for Creating Graphical User Interfaces*. Proc. INTERACT 1999, IOS Press, 510-518.
16. Lecolinet E., *A molecular architecture for creating advanced interfaces*. Proc. UIST 2003, ACM.
17. Larillau Y., Nigay L. *Le modèle d'architecture Clover pour les collecticiels*. Proc. IHM 2002, ACM, 113-120.
18. Myers B.A., Hudson S.E., Paush R., *Past present and future of user interface software tools*, ACM ToCHI 2000, 43(3), 82-89.
19. Myers B.A. et al.. *The Amulet Environment: New Models for Effective User Interface Software Development*. IEEE Trans. on Software Engeenering, 1997, 23(6), 347-365.
20. Myers, B., Stiel H, Gargiulo R.. *Collaboration Using Multiple PDAs Connected to a PC*. Proc. CSCW' 98, 258-294.
21. Pook S., Lecolinet E., Vaysseix G., Barillot E. *Control Menus: Execution and Control in a Single Interactor*. Proc. CHI 2000, ACM, 263-264.
22. Qt toolkit. http:www.trolltech.com/qt
23. Richardson T, Stafford-Fraser Q., Wood K.R., Hopper A, *Virtual Network Computing*, IEEE Trans. on Internet Computing, 1998, 2(1), 33-38.
24. Roseman, M. and Greenberg, S. *Building Real Time Groupware with GroupKit, a Groupware Toolkit*. ACM ToCHI, 1996, 3(1), 66-106.
25. Smart Technologies SMART Board, http://www.smarttech.com/SmartBoard/.
26. Stewart, J., Bederson, B.B, Druin, A. *Single display groupware: A model for co-present collaboration*. Proc. CHI 1999, ACM. 286-293.
27. Peter Tandler: Software Infrastructure for Ubiquitous Computing Environments:. Proc. UbiComp 2001. Springer.
28. Ubit toolkit : www.enst.fr/elc/ubit