THÈSE
présentée par

Christophe Lachenal

pour obtenir le titre de
DOCTEUR de L'UNIVERSITE JOSEPH-FOURIER-GRENOBLE I
(arrêtés ministériels du 5 juillet 1984 et du 30 mars 1992)
Spécialité : Informatique

Modèle et infrastructure logicielle pour l'interaction multi-instrument multisurface

Composition du Jury :

Directeur de thèse :	Mme. Joëlle Coutaz
Rapporteurs :	Mr. Marc Nanard Mr. Michel Riveill
Jury :	Mr. James. L Crowley Mr. Eric Lecolinet Mr. Gilles Privat

Thèse soutenue le 17 décembre 2004

Thèse préparée au sein du laboratoire de Communication Langagière et
Interaction Personne-Système
Fédération IMAG
Université Joseph Fourier - Grenoble I

Remerciements

Au début de l'année scolaire 1998-1999, Laurence Nigay et Frédéric Vernier ont proposé un sujet de magistère intitulé "Utilisation réflexive du flux vidéo en Interaction Homme-Machine". Ce sujet de recherche proposait d'étudier le principe des pixels miroirs. La description du sujet vantait le matériel informatique mis à disposition. Cet argument a su me convaincre. C'est donc par l'étude des pixels-miroirs que mes années au sein de l'équipe Ingénierie de l'Interaction Homme-Machine du laboratoire CLIPS de Grenoble ont commencé.

Malheureusement aujourd'hui elles s'achèvent par ce manuscrit. J'espère que sa lecture vous convaincra de l'intérêt du domaine de l'Interaction Homme-Machine dans la recherche en Informatique.

Outre les connaissances techniques que j'ai pu acquérir, je garde surtout de très bons souvenirs. Des souvenirs d'une aventure humaine au sein d'une équipe où l'ambiance a toujours été au beau fixe, où la recherche est un plaisir partagé et où les idées foisonnent. Des souvenirs de réunions où présenter son travail est parfois difficile, surtout à partir du deuxième transparent... Des souvenirs d'avion manqué lorsque avec Joëlle nous étions trop occupés à discuter de trajectoires souris dans un environnement multisurface. Des souvenirs de stress mais aussi de joie lorsque à Göteborg il a fallu réinstaller les machines qui avaient voyagé en pièces détachées et faire marcher notre démonstration pour le projet européen Gloss. Des souvenirs de réflexions intenses en B204, surtout quand la porte est fermée (Heureusement même avec la porte fermée le réseau fonctionne). Des souvenirs de baseball avec la bande à toto. Etc,
etc...

C'est pourquoi je tiens à remercier tous les ihmiens de Grenoble pour ces moments uniques et notamment les personnes suivantes :
Jöelle, Laurence, Gaëlle, Sophie, François, Fred, Manu, David, Yann, Christian, Gaëtan, Nicolas, Philippe, Benoît, Chaouki, Olfa, Sylvain, Jullien, Julien, Sylvie, Alexandre, Lionel, Bérangère, Julie, Léon, Cyril, Thung et tous les "petits" nouveaux que je n'ai pas eus le temps de connaître.

Merci encore Joëlle pour m'avoir guidé dans mes recherches et permis d'apprendre tant de choses dans ton équipe.

Merci Maman pour m'avoir donné le goût des études. Merci Maryline pour ton soutien sans faille pendant ces années de recherche.

Bonne lecture et bon code...

Table des matières

Liste des figures

Chapitre I *Introduction*

Mes travaux de recherche doctorale concernent le logiciel de base pour la mise en œuvre d'interfaces distribuées et migratrices au sein d'espaces interactifs dynamiques multisurface, multi-instrument. Cette étude trouve sa justification dans la convergence progressive des nouvelles technologies et l'absence d'outil en Interaction Homme-Machine (IHM) pour couvrir les nouvelles possibilités que cette convergence permet d'envisager.

I.1. Convergence matériel-réseau-système-perception-IHM

L'informatique se prépare à une nouvelle évolution avec la convergence des réseaux sans fil, des micro- et nanosystèmes et la maturation de trente années de recherche en systèmes répartis, en perception artificielle et en Interaction Homme-Machine. Point commun de tous les projets menés dans ces domaines : la nécessité de définir une infrastructure multi-échelle (de la planète à l'infiniment petit) permettant de développer des services à finalités humaines où l'Interaction Homme-Machine limitée pour l'instant au contrôle d'une machine, se fond progressivement dans le monde physique comme l'a pressenti M. Weiser [Weiser 1991].

Sur les idées fondatrices de Weiser, se déclinent aujourd'hui l'informatique ubiquiste (ubiquitous computing), l'ordinateur évanescent (disappearing computer), l'informatique pervasive (pervasive computing), l'informatique ambiante (ambiant computing), l'informatique augmentée (augmented computing), ou encore les objets communicants et les systèmes embarqués. Cette diversité de nomenclature traduit l'instabilité des concepts et un espace problème aux contours mal cernés.

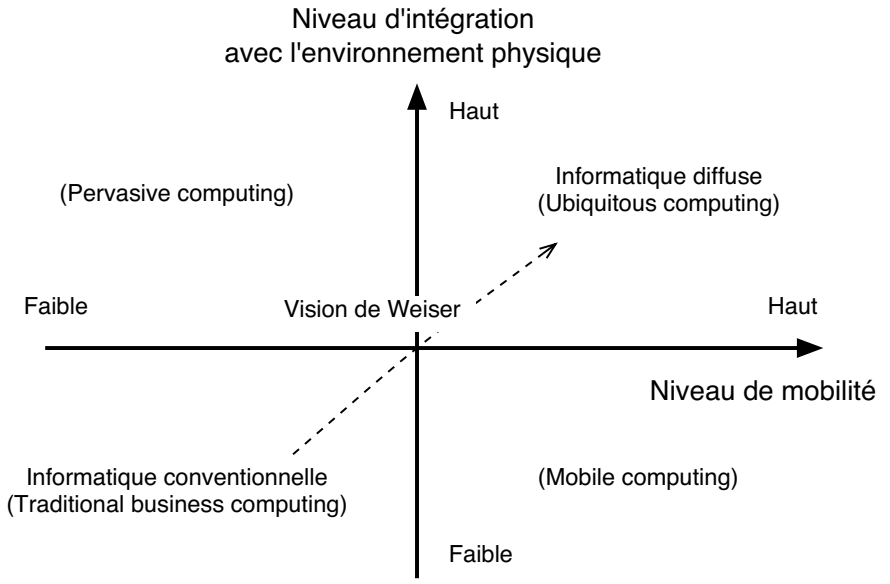

Figure 1. Les niveaux d'intégration et de mobilité : les axes de Lyytinen et Yoo.

En réponse à cette prolifération, Lyytinen et Yoo introduisent deux axes de classification : le niveau de mobilité (level of mobility) et le niveau d'intégration (level of embeddedness) des systèmes [Lyytinen et Yoo 2002].

- La mobilité revient à disposer, en tout lieu, du service numérique souhaité. En quelque sorte, le service nous suit (ou donne l'impression de nous suivre). Cette possibilité résulte de la miniaturisation des dispositifs d'interaction qui, de confinés au bureau, deviennent portables et connectables au réseau sans fil. Toutefois, un service purement mobile ne tient pas nécessairement compte de son environnement. Cette dépendance à l'environnement relève, selon Lyytinen et Yoo, du niveau d'intégration.
- Le niveau d'intégration dénote la capacité du système à faire corps avec le monde physique et réciproquement. Le système est alors en mesure d'extraire de l'information sur l'environnement pour construire des modèles numériques adaptés au milieu d'interaction. Réciproquement, l'environnement sait détecter les allées et venues des unités de calcul, forgeant ainsi une dépendance étroite entre les mondes physique et numérique.

Comme le montre la figure 1, l'informatique conventionnelle se range dans le cadre « absence de mobilité, faible niveau d'intégration» ; et la vision de M. Weiser se situe à l'autre extrême avec un « haut niveau de mobilité et d'intégration». Mes travaux de recherche s'inscrivent dans ce dernier cadre.

I.2. Des IHM centralisées aux l'IHM distribuées et migratrices

Avec l'informatique ambiante, l'Interface Homme-Machine d'un système n'est plus confinée à la seule station de travail. Reprenant les dimensions de la figure 1, le service IHM devient mobile et s'insère dans l'environnement. En même temps, l'environnement sert de support à l'interaction avec le monde numérique. Ensemble, environnement physique et monde numérique constituent des espaces interactifs. Dans ces espaces, certains objets physiques constituent des moyens d'interaction entre l'utilisateur et le système : ce sont des ressources d'interaction.

Par exemple, dans l'espace FAME [Fame] qu'illustre la figure 2, l'IHM est répartie entre une table et deux murs. La table permet de rechercher de l'information au moyen de jetons en carton, tandis que les réponses aux requêtes migrent de la table vers l'un des murs avec effet d'animation. Les sujets de discussion, que les utilisateurs évoquent entre eux oralement et que l'environnement reconnaît, sont projetés sur un second mur sous forme d'une fontaine de mots. Ces mots sont ensuite sélectionnables pour plus ample information sur le sujet. Ici, jetons, table et murs sont des ressources d'interaction.

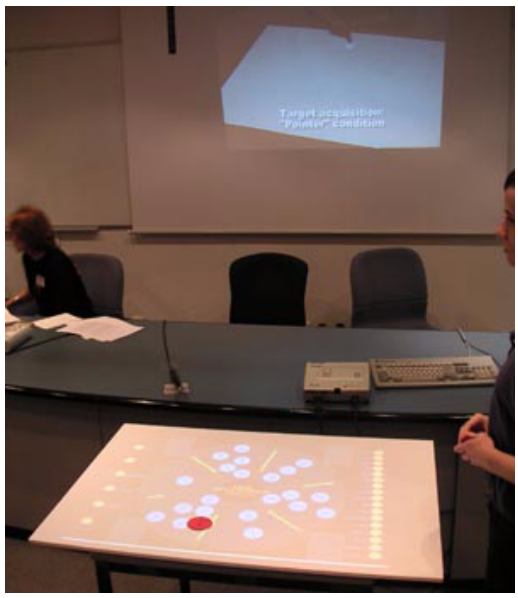

Figure 2. L'espace FAME : une table et un mur. Avec un effet d'animation l'interface graphique peut migrer de la table vers les murs.

Dans l'espace FAME, le nombre d'unités de calcul est fixe, mais les jetons, en nombre quelconque, peuvent apparaître et disparaître à volonté. De manière générale, dans la vision de l'informatique ambiante, l'utilisateur peut prêter, assembler, et emprunter des ressour-

ces d'interaction au gré des opportunités. L'utilisateur passe de sujet à acteur. Il devient créateur d'espaces interactifs. Par exemple, la figure 3 à droite, illustre la situation où deux amis se rencontrant fortuitement assemblent leur PDA. Ils obtiennent ainsi une surface d'affichage suffisamment grande pour engager une collaboration. À gauche, le voyageur à proximité d'un mur actif, obtient des renseignements que l'environnement transfère automatiquement au bon format sur le PDA.

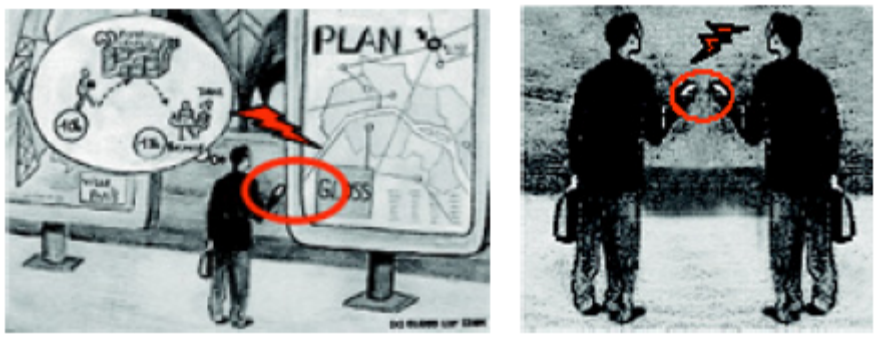

Figure 3. Exemples de couplages de surfaces.

Face à cette évolution qu'impose la vision des espaces interactifs spontanés, le développeur d'IHM se trouve bien dépourvu. Cette situation justifie mes travaux de recherche doctorale.

I.3. Motivation de la recherche

Le développeur d'IHM dispose aujourd'hui d'outils logiciels éprouvés : systèmes de fenêtrage, boîtes à outils, squelettes d'application, et générateurs d'IHM. Mais ces outils sont adaptés aux IHM centralisées pour une classe donnée de stations de travail supposées dotées, pour toute ressource d'interaction, d'un seul espace d'affichage (l'écran), d'un dispositif de saisie (le clavier) et d'un dispositif de pointage (la souris). Il convient donc de s'interroger sur le bien fondé de ces outils et notamment des plus fondamentaux d'entre eux, au regard des requis des IHM en informatique ambiante.

En particulier, les systèmes de fenêtrage de demain doivent être capables de gérer l'assemblage opportuniste d'unités de calcul et de ressources d'interaction. Ce requis général appelle deux questions, l'une technique, la seconde de nature interactionnelle :

- Comment gérer la découverte mais aussi l'hétérogénéité des ressources de calcul, de communication et d'interaction ?
- Quels modèles d'interaction proposer? Autrement dit, le paradigme « écran-dispositif de saisie-dispositif de pointage » est-il encore

valide ? Qu'offrir au-delà du desktop, puisque le lieu d'interaction est maintenant l'espace. La notion de fenêtre fait-elle encore sens ?

Mes travaux de thèse tentent de répondre à la première question sans pour autant ignorer les aspects interactionnels.

I.4. Objectifs et approche

L'objectif de cette thèse est de réaliser un intergiciel capable d'assurer la découverte de ressources de calcul et d'interaction et d'en masquer l'hétérogénéité pour former, du point de vue du développeur, un espace uniforme d'interaction. Ainsi, il doit être possible à un utilisateur de relier plusieurs machines exécutant des systèmes d'exploitation différents (par exemple, MacOS et Windows) et gérant chacune des ressources d'interaction différentes. Du point de vue programmation, ces ressources doivent donner l'impression d'être gérées par une seule machine : la multiplicité des machines et des systèmes d'exploitation, l'allocation des ressources d'interaction à ces machines, et l'hétérogénéité des machines, des systèmes et des ressources doivent être transparentes au programmeur, mais accessibles si le besoin s'en fait sentir.

Si les objectifs de cette thèse sont essentiellement techniques, l'approche adoptée pour aboutir à une solution, s'appuie sur les principes d'une interaction où les mondes physiques et numériques fusionnent pour constituer des espaces interactifs. Dans ces espaces, certaines entités physiques servent de ressources d'interaction. Les unes jouent le rôle d'instrument, d'autres celui de surface. Dans l'exemple de l'espace FAME, la table et les murs tiennent lieu de surface tandis que les jetons en carton servent d'instruments.

Le choix conceptuel, instrument-surface, est guidé par la force de l'expérience :

- De tout temps, l'Homme a créé des instruments pour l'aider à façonner son environnement : le marteau, la pelle, le stylo, mais aussi … la souris et le clavier.
- Les surfaces structurent l'espace pour former des endroits, tels le hall de gare ou la salle de séjour, favorisant l'émergence ou la conduite d'activités spécifiques : fresques et tableaux d'art, feuillets annotés de livre, tableau d'école, mais aussi … écrans d'ordinateur.
- Surfaces et instruments entretiennent avec nous des relations interactionnelles privilégiées : ils s'appréhendent par les sens, se manipulent par nos actes moteurs et servent un objectif. L'un tient dans la main, l'autre ne se transporte pas. Certains sont déformables et pliables, d'autres sont rigides mais orientables. L'un est pérenne, l'autre

jetable.

Les résultats de cette recherche, conceptuels et techniques, constituent les deux parties principales du mémoire.

I.5. Structure du mémoire

La première partie du mémoire, qui comporte 4 chapitres, reflète l'analyse amont nécessaire à la bonne compréhension du problème technique à résoudre.

Le chapitre 2 présente le cadre conceptuel dans son ensemble. On y définit avec précision les concepts (et leurs relations) jugés utiles pour la conception d'outils logiciels de mise en œuvre d'IHM en informatique ambiante. On se limitera aux IHM de nature graphique. Ce cadre conceptuel, on le rappelle, est résolument guidé par l'idée de fusion entre les mondes physiques et numériques. Les concepts de surface et d'instrument servent de conduits à ce rapprochement. À leur tour, ils donnent naturellement naissance à la notion d'interaction multisurface, multi-instrument.

Les chapitres suivants développent les éléments fondamentaux et originaux du cadre conceptuel et largement sous-exploités par les outils de développement traditionnels :

- Au chapitre 3, les relations spatiales : parce que l'interaction s'inscrit dans l'espace, les relations spatiales entre les ressources d'interaction et l'utilisateur ne peuvent plus être ignorées.
- Au chapitre 4, le couplage de ressources d'interaction : parce que l'utilisateur a la possibilité de façonner son espace, ajouter ou retirer des ressources, le problème du couplage devient un sujet central.
- Au chapitre 5, la projection de l'espace numérique sur l'espace physique : puisqu'il y a fusion des mondes physiques et numériques, il convient de s'interroger sur la façon de les mettre en correspondance.

Le chapitre 6 introduit de la deuxième partie du mémoire. Il s'appuie sur les éléments conceptuels des chapitres précédents pour proposer une synthèse des requis logiciels et une classification des systèmes multi-instrument multisurface. Les outils de l'état de l'art alimentent l'analyse et justifient ma contribution technique.

Le chapitre 7 présente les principes de la réalisation technique de mes travaux : I-AM (Interaction Abstract Machine). I-AM remplit les objectifs énoncés en s'appuyant sur le cadre conceptuel.

Enfin, dans le chapitre 8 de conclusion, nous évaluons notre contribution technique I-AM au regard des requis exprimés au chapitre 6 et nous présentons quelques perpectives à notre travail.

Chapitre II *Ontologie pour la conception d'outils de réalisation d'IHM en informatique ambiante*

Le cadre conceptuel ici présenté vise trois objectifs :

- Comprendre la nature de l'interaction en informatique ambiante,
- Identifier les requis logiciels pour la mise en œuvre de ces nouvelles IHM, et
- Fournir des éléments de classification pour situer et comparer les solutions actuelles de l'état de l'art.

Une approche à la compréhension d'un nouveau problème est l'élaboration d'une ontologie capable d'offrir une structure systématique au raisonnement. Ce chapitre en présente le résultat, le Modèle du Processeur Humain (MPH) [Card 83] servant de source d'inspiration. MPH structure l'acteur humain en trois systèmes : perceptif, moteur et cognitif. Comme nous l'avons vu dans le chapitre d'introduction, les espaces interactifs sont doués de capacités de perception et d'action, et savent élaborer des modèles adaptés à la situation. En conséquence, il est raisonnable d'appliquer la décomposition du Modèle du Processeur Humain au cas des espaces interactifs. On obtient le schéma de la figure 4.

Une situation d'interaction met en scène deux acteurs, l'utilisateur et le système, en symétrie par rapport à un ensemble d'entités physiques. Ils disposent chacun d'effecteurs pour agir sur ces entités physiques et de capteurs pour les observer. Les entités physiques, lorsqu'elles servent d'objets de médiation entre les acteurs, tiennent lieu de ressources d'interaction. Par exemple, au moyen de ses capteurs, caméra et microphone, l'espace FAME détecte la présence de jetons sur une table. De même, il capte la conversation des acteurs humains. Ces informations alimentent des modèles internes (l'équivalent du système cognitif humain) qui déterminent la position des jetons sur la table, de même le sujet de conversation, et le service demandé. FAME produit une réaction au moyen de ses effecteurs, haut-parleurs et projecteurs vidéo. Symétriquement, les utilisateurs observent le changement d'état des murs et de la table, raisonnent et agissent en conséquence par action

sur les jetons, par lecture des informations projetées sur les murs, ou encore, par écoute des réponses vocales.

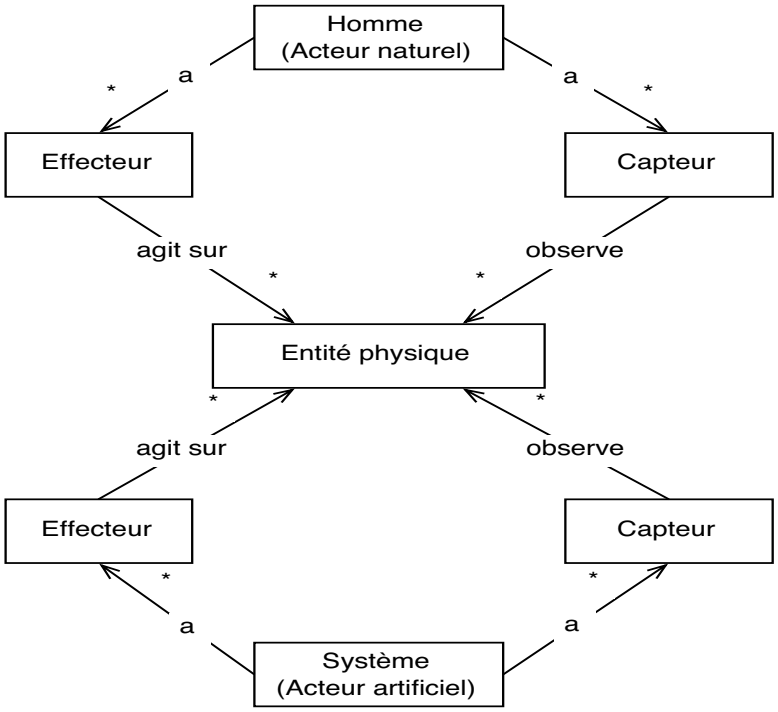

Figure 4. Les acteur agissent sur les entités physiques et les observent.

Ayant introduit le principe du cadre conceptuel, ce chapitre en présente maintenant les concepts et leurs relations. Ensuite, nous traiterons en détail les notions d'entité physique (section 2), de leur rôle comme ressource d'interaction (section 3) et notamment leur usage en tant que surface et instrument (sections 4 et 5).

II.1. Présentation générale

Le diagramme UML de la figure 5 sert de support à l'expression de notre ontologie. La symétrie de la figure 4 se traduit maintenant ainsi : l'utilisateur, qui est un Acteur Naturel, et le système, qui est un Acteur Artificiel, sont des Acteurs. Reprenant à notre compte la décomposition MPH, un acteur comprend :

- des capteurs pour observer l'état d'Entités physiques du monde réel et élaborer de l'Information. Cette aptitude est représentée dans le diagramme par la classe associative Observation, sous-classe

d'Information ;

- un système cognitif pour raisonner et élaborer de l'information ;
- des effecteurs pour agir sur des entités du monde réel, d'où la classe associative Action entre Effecteurs et Entités physiques.

Nous verrons plus loin l'importance des notions d'observation et d'action dans l'interaction et notamment aux chapitres.

Un acteur, ses capteurs et ses effecteurs sont aussi des entités physiques. Toute entité physique peut être composée d'entités, peut agir sur une autre (par exemple, une boule de billard heurtant une autre boule). Une entité physique entretient des relations spatiales avec d'autres entités, elle peut être couplée à une autre entité pour fournir une nouvelle fonction et aussi, elle peut jouer un ou plusieurs rôles de ressource d'interaction : un rôle de surface et/ou d'instrument.

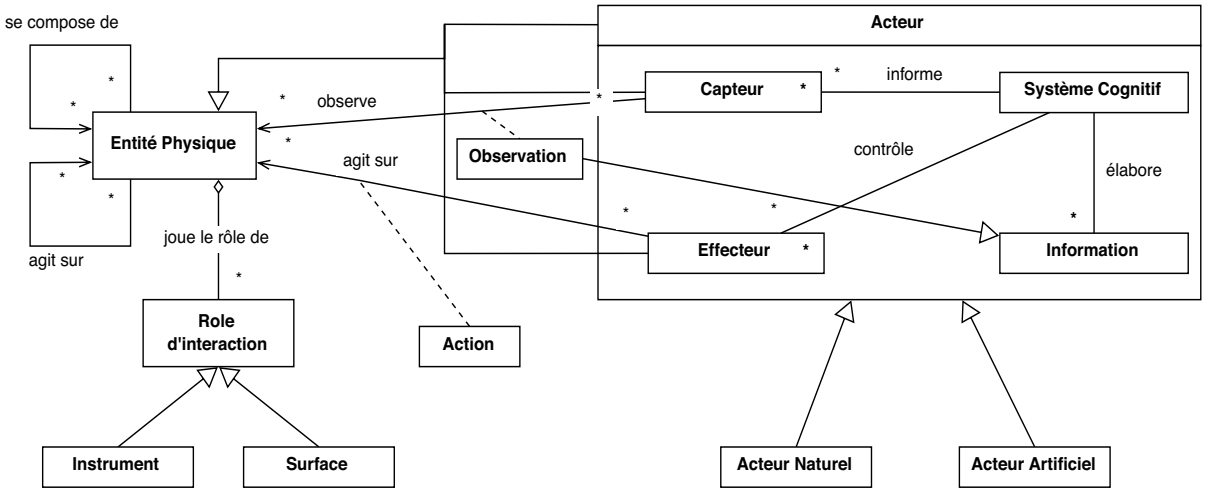

Figure 5. Diagramme de classe synthétisant les associations entre les concepts clés d'entité physique, d'acteur, d'action, d'observation, d'instrument et de surface.

Considérant maintenant le cas particulier du système, cet acteur exécute une application (au sens large du terme) qui gère/inclut des interacteurs, entités logicielles constitutives de l'Interface Homme-Machine. Les interacteurs sont rendus observables par une fonction de projection sur des entités physiques ayant le rôle de surface. Certains interacteurs servent à représenter l'état d'entités physiques ayant le rôle d'instruments. Les fonctions de projection et l'affectation des rôles peuvent être choisies par un acteur (système ou utilisateur).

Les relations spatiales, le couplage et la projection sont présentés dans les trois chapitres qui suivent. Ici, nous poursuivons plus avant la description des entités physiques (section 2), des ressources d'interaction (section 3) et leur rôle de surface et d'instrument (sections 4 et 5).

II.2. Entité physique

Dans notre modèle, l'entité physique est centrale : tout objet matériel de la vie courante est une entité physique, mais les acteurs, leurs capteurs et leurs effecteurs sont aussi des entités physiques. Ceci signifie qu'un acteur peut en observer un autre (et ses constituants) et/ou agir dessus. Ce choix de modélisation permet de rapprocher les mondes physiques et numériques en donnant aux entités du monde différents statuts selon les besoins, le contexte, le point de vue.

Par exemple, l'espace FAME observe des jetons, le Tableau Magique observe des doigts (voir figure 6), le système VideoPlace de Krueger [Krueger 1990] (voir figure 7) observe un personnage. Du point de vue du système, les jetons, les doigts et le personnage sont des entités physiques tenant lieu d'instrument (il s'agit d'instrument, puisque ces entités en action modifient l'état du système). Du point de vue de l'acteur humain, le jeton sert d'instrument, ses doigts (cas du Tableau Magique) et son corps sont des effecteurs. Dans un système de vidéo-surveillance, les personnes ont le statut de simples entités physiques.

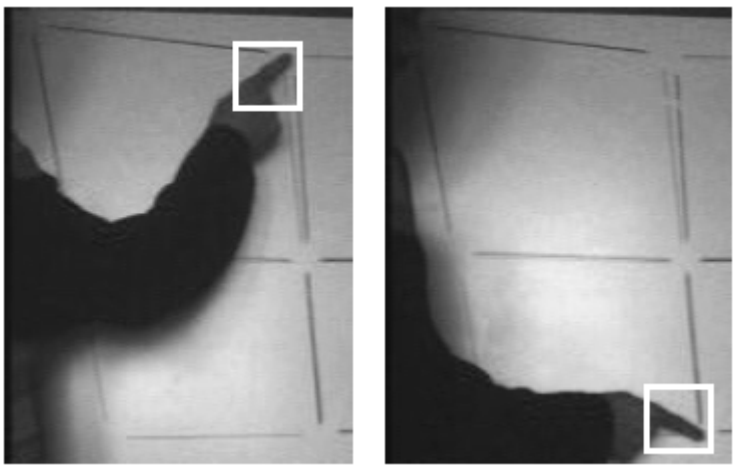

Figure 6. Le tableau magique : une surface augmentée qui permet une interaction au doigt.

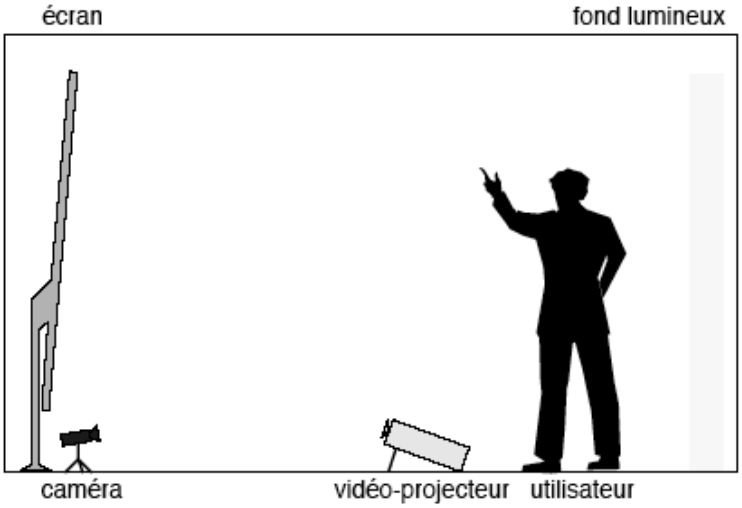

Figure 7. Le dispositif de VideoPlace d'après [Krueger 1990].

En somme, le statut d'une entité physique dépend du point de vue de l'acteur qui en use. À son tour, l'usage qui en est fait (observer et/ou agir sur) dépend des caractéristiques intrinsèques de l'entité et des propriétés qui en découlent. Par conséquent, la connaissance des caractéristiques des entités et le calcul de leurs propriétés deviennent incontournables dans la conception et la mise en œuvre d'espaces interactifs. C'est pourquoi nous proposons, sans viser l'exhaustivité, une liste d'attributs et de propriétés souvent ignorés, au mieux sous-exploités, dans les outils actuels de développement d'IHM. La figure 8 présente le diagramme de classe correspondant.

Entité Physique
forme géométrique taille poids matériaux constituants texture position
mobilité solidité, fluidité, nébulosité rigidité, souplesse opacité, transparence hétérogénéité réfraction, réflexion

Figure 8. Diagramme de classe de l'objet Entité Physique. Les attributs et propriétés d'une entité physique doivent pouvoir être modèlisées.

Nous distinguons les attributs de localisation présentés en détail au chapitre suivant et les attributs liés à la morphologie de l'entité :

- *Forme géométrique* (rectangulaire, circulaire, sphérique, etc.) et *taille*.
- *Matériau constitutif* (densité), *texture*, *couleur*, *réfraction*.

Les propriétés, déduites des attributs et que nous retenons, traduisent des qualités interactionnelles :

- *Mobilité* : dérive des attributs de forme, de taille et de matériau (le poids). Une entité immobile, par exemple un mur, peut, pour un acteur système, servir de référence dans la définition de relations spatiales. Inversement, les entités mobiles devront être repérées et suivies en permanence. Un objet lourd et volumineux ne peut pas tenir dans la main et donc servir d'instrument.
- *Solidité/Fluidité/Nébulosité* : se déduit du matériau constituant l'entité. Un objet fluide peut difficilement servir d'instrument, mais on peut le traverser !
- *Rigidité/Souplesse* : comme précédemment, ces propriétés se déduisent du matériau. Elles expriment la capacité de l'entité à changer de forme. Une entité physique rigide est, si elle est incassable, de forme immuable. Une entité souple peut être pliée ou, au contraire, déroulée. Élastique, elle peut être étirée puis relâchée pour reprendre sa forme initiale. La connaissance de ces propriétés a, notamment, un fort impact sur les systèmes de suivi en vision par ordinateur. La taille d'un tableau blanc déroulant peut être ajustée en fonction des besoins. La connaissance, par le système, de cette variabilité, peut le conduire à remodeler la présentation de l'information projetée sur ce tableau. Nous parlons alors d'IHM plastique [Thevenin 99].
- *Opacité/Transparence* : Une entité transparente s'enrichit des informations environnementales. La transparence favorise l'ouverture tout en formant une frontière.
- *Hétérogénéité* : L'hétérogénéité peut inciter les acteurs à utiliser différemment les parties distinctes de l'entité. Par exemple, la table de l'espace GLOSS (voir figure 9) comprend une planche de bois sertie d'un tableau blanc. La partie en bois sera plutôt utilisée pour déposer des objets (papier, stylos), tandis que la partie tableau servira

essentiellement à écrire.

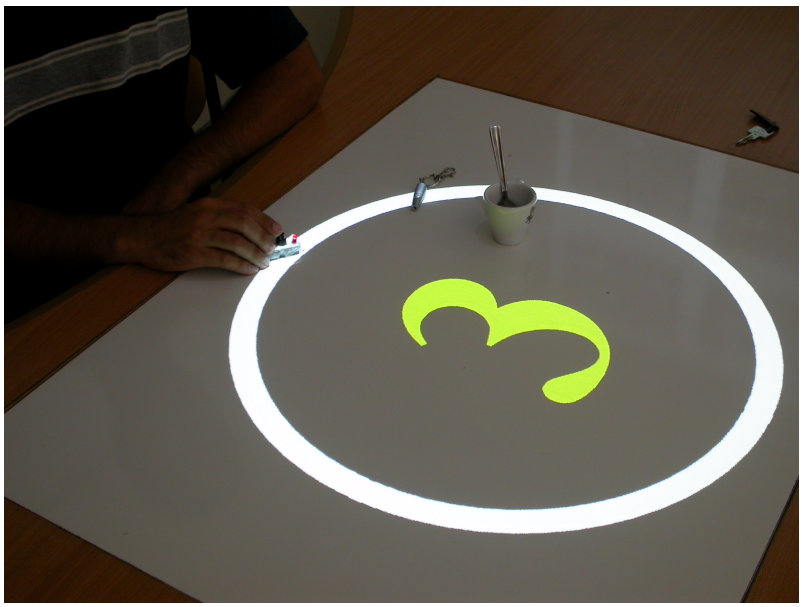

Figure 9. La table Gloss. Les caractéristiques des entités physiques et notamment celles jouant le rôle de surface peuvent influencer la conception du système interactif.

En résumé, selon l'acteur et les caractéristiques intrinsèques d'une entité physique, celle-ci peut ou non avoir un rôle d'interaction. Si c'est le cas, cette entité est une ressource d'interaction.

II.3. Ressource d'interaction

Après une définition formelle de la notion de ressource d'interaction (paragraphe 3.1) , nous présentons au paragraphe 3.2, les situations d'usage les plus caractéristiques pour terminer en 3.3 sur la notion de rôle d'interaction.

II.3.1. DÉFINITIONS

Afin de définir formellement une ressource d'interaction, nous introduisons au préalable le concept de canal d'interaction. Soient :

- a_1 et a_2, deux acteurs distincts,
- A_1 une séquence (éventuellement vide) d'actions produites par a_1 sur une entité physique E_1, A_2 une séquence (éventuellement vide) d'actions produites sur une entité E_2 comme conséquence de A_1, …, et A_n une séquence (éventuellement vide) d'actions produites sur E_n

comme conséquence de A_{n-1},

- O_{2n}, l'observation de E_n par a_2.

On note :

- A*O, la séquence $A_1 A_2 \ldots A_n O_{2n}$,
- Premier(A*), le premier élément de la séquence A*.

Définition : Un *canal d'interaction* est établi entre deux acteurs a_1 et a_2, s'il existe une chaîne A*O telle que Premier(A*) est une action produite par a_1 et O, une observation effectuée par a_2. Si A* est nulle ou si Premier(A*) est nulle, alors l'entité observée par a_2 est l'acteur a_1 (et/ou ses constituants).

La notion de chaîne d'actions modélise la répercussion d'actions entre entités (telle l'action d'un acteur sur une boule de billard qui en heurte d'autres et ainsi de suite). En bout de chaîne, un acteur peut observer sur la dernière entité le résultat cumulé de ces actions. Par exemple, par une action sur ses cordes vocales, l'utilisateur produit un son. Ce son agit sur la membrane d'un microphone qui se met à vibrer. Le microphone traduit les vibrations de la membrane et produit un signal analogique observé par un ordinateur. Bien sûr, la granularité des entités et des actions considérées dans une chaîne dépend du problème à traiter.

Nous sommes maintenant en mesure de fournir une définition précise de la notion de ressource d'interaction.

Définitions :

Une *ressource d'interaction* est une entité physique intervenant dans un canal d'interaction.

Une *ressource d'interaction initiale* correspond à la première entité du canal d'interaction.

Une *ressource d'interaction terminale* désigne la ressource observée du canal d'interaction.

Une *ressource d'interaction directe* est à la fois initiale et terminale.

Une *ressource d'interaction indirecte* ou *transducteur* désigne une ressource d'interaction qui n'est pas initiale, ni terminale.

Les claviers, souris et écrans de nos stations de travail répondent à ces définitions. D'ailleurs, ils ont été conçus pour remplir la fonction de ressource d'interaction. En vérité, notre définition couvre toute entité physique de monde réel, pourvu qu'elle soit impliquée dans un canal d'interaction. Considérons les "earcons" [Blattner et al 1989]. Par ana-

logie avec le "phicon" [Ishii et Ullmer 1997], un "earcon" est une entité physique manipulable et permet de représenter l'état d'un service informatique sous forme sonore. Par exemple, un acteur humain peut choisir d'utiliser une enveloppe comme représentant du courrier électronique. En raison des propriétés de l'enveloppe, l'utilisateur peut la déplacer et la poser où bon lui semble. L'acteur système observe les actions de déplacement de l'enveloppe grâce à un suivi de vision par ordinateur. À l'arrivée d'un nouveau message, le système émet un son spatialisé de manière à ce que ce son observé par l'acteur humain, semble provenir de l'enveloppe.

Être ressource d'interaction n'est pas une classe, mais une propriété qui dépend de dynamiquement des caractéristiques et propriétés de l'entité mais aussi des capacités d'action et d'observation des acteurs en présence. Autrement dit, une entité qui, à un instant t, est une ressource d'interaction, peut, à l'instant t+1, ne plus l'être et inversement. En outre, selon ses capteurs et ses effecteurs, et selon les caractéristiques des entités physiques en présence, un acteur peut observer et/ou agir sur plusieurs entités physiques simultanément. De même, une entité donnée peut être impliquée dans plusieurs canaux d'interaction simultanément. Ces situations permettent de raisonner sur le niveau de partage des entités physiques et sur leur multiplexage spatio-temporel.

Il peut arriver que l'interaction entre deux acteurs nécessite la connexion en séquence de plusieurs canaux d'interaction. C'est le cas notamment de la communication interpersonnelle médiatisée où un individu peut observer les actions d'un autre (et réciproquement) [Mantei et al 1991] :

- Un premier canal d'interaction est établi entre l'utilisateur source et son ordinateur local (qui observe les actions de l'utilisateur source),
- Un second canal existe entre l'ordinateur local et l'ordinateur distant (qui observe les actions d'écriture sur le réseau de l'ordinateur source), et
- Un troisième canal lie l'ordinateur distant et l'utilisateur destinataire qui observe les actions de sa machine (affichage de pixels, émission de son, etc.). Ces actions sont supposées refléter les actions de l'utilisateur source.

La spécification explicite de plusieurs canaux d'interaction en chaîne dépend du niveau d'abstraction souhaité dans la modélisation du système. Quant à nous, nous nous limitons aux chaînes A*O entre un acteur humain et un acteur système et ignorons les canaux d'interaction entre acteurs humains et ceux établis entre acteurs système.

En synthèse, rôle d'interaction, partage et multiplexage sont des qualités qui évoluent et qui dépendent des caractéristiques et propriétés des entités physiques en présence mais aussi des capacités d'action et

d'observation des acteurs. Bien veiller aux ressources d'interaction indirectes d'une chaîne A*O, car si celles-ci ne sont pas en contact direct avec les acteurs, elles n'en sont pas moins indispensables pour que l'interaction ait lieu.

Passons maintenant en revue les situations les plus caractéristiques.

II.3.2. Situations caractéristiques

Les situations les plus courantes sont celles où les acteurs humain et système sont liés par une entité physique. La figure 10 illustre les quatre situations types.

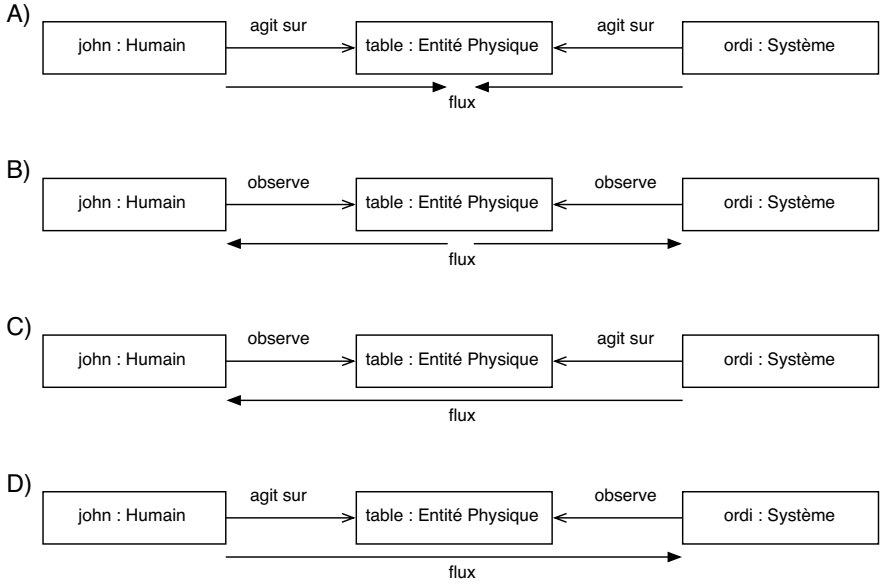

Figure 10. Les quatre situations types entre un acteur humain, un acteur système et une entité physique.

A) Les deux acteurs peuvent agir et seulement agir, sur l'entité physique. Alors, les acteurs ne peuvent obtenir de retour d'information sur les actions qu'ils produisent. Dans ce cas, aucune situation d'interaction n'est possible (cas haut gauche de la figure 10). On dit que l'entité physique est un puits d'actions car les actions ne semblent avoir aucun effet.

B) Les deux acteurs ne peuvent qu'observer l'entité physique. Ce cas correspond à la situation où l'entité est hors de portée d'action des deux acteurs (cas bas droite de la figure 10). L'entité ne peut servir de ressource d'interaction, mais elle est une source d'information partagée entre les deux acteurs. C'est le cas, par exemple, d'un thermomètre que les deux acteurs observent pour déterminer la température d'une pièce.

C) L'acteur humain peut observer et l'acteur système peut agir sur l'entité physique. Alors l'entité physique peut servir de ressource d'interaction directe en tant que dispositif de sortie (cas haut droite de la figure 10).

D) L'acteur humain peut agir sur l'entité et l'acteur système peut l'observer. Alors l'entité physique peut servir de ressource d'interaction directe en tant que dispositif d'entrée (cas bas gauche de la figure 10).

Poussons le raisonnement plus avant : chacun des acteurs peut agir sur l'entité physique, l'observer ou faire les deux. Alors, comme le montre la figure 11, l'utilisation d'une entité physique peut se faire selon 9 configurations possibles. Parmi ces configurations, quatre correspondent aux situations types que nous venons de présenter. Les cinq autres sont construites par produit cartésien entre action et observation. Dans les expressions qui suivent, le premier terme du produit correspond à l'acteur humain et le second à l'acteur système. On obtient :

- action x action = puits d'actions (cas A : pas d'intérêt interactionnel),
- observation x observation = source d'information partagée (cas B : pas d'intérêt interactionnel),
- observation x action = dispositif de sortie (cas C),
- action x observation = dispositif d'entrée (cas D),

Pour les cinq autres cas, appliquons la distributivité du produit :

- action x (action + observation) = action x action + action x observation = puits d'action + dispositif d'entrée = dispositif d'entrée (ligne du haut, centre, de la figure 11),
- (action + observation) x action = action x action + observation x action = puits d'action + dispositif de sortie = dispositif de sortie (ligne du centre, gauche, de la figure 11),
- (action + observation) x observation = action x observation + observation x observation = dispositif d'entrée + source d'information partagée = dispositif d'entrée (ligne du centre, droite, de la figure 11),
- observation x (action + observation) = observation x action + observation x observation = dispositif de sortie + source d'information partagée = dispositif de sortie (ligne du bas, centre, de la figure 11),
- (action + observation) x (action + observation) = action x action + action x observation + observation x action + observation x observation = puits d'action + dispositif d'entrée + dispositif de sortie + source d'information = dispositif d'entrée + dispositif de sortie (la

case au centre).

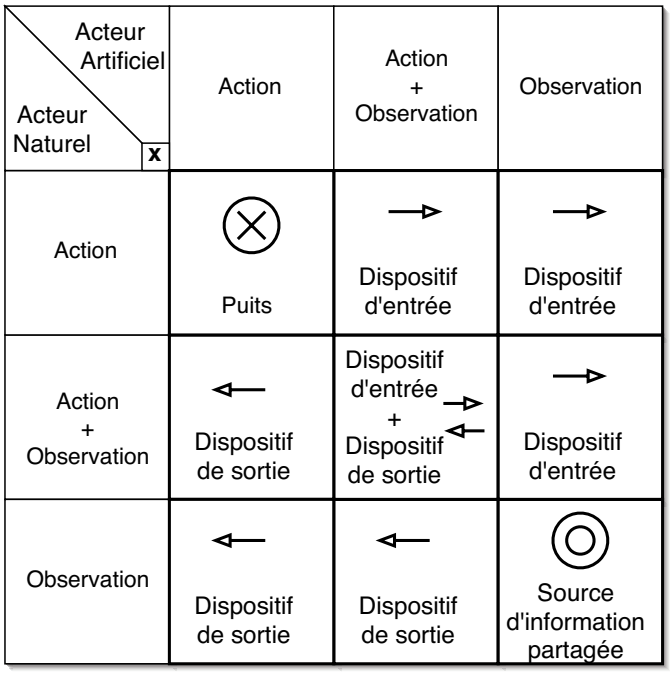

Figure 11. En agissant sur et/ou en observation une entité physique commune, une acteur naturel et un acteur artificiel confère suivant les cas la propriété de ressource d'interaction à cette entité.

Ainsi, une entité physique peut être utilisée comme ressource d'interaction directe dans 7 cas sur 9. Dans 3 cas, elle peut être utilisée comme dispositif d'entrée. Dans 3 cas, elle peut être utilisée comme dispositif de sortie. Mais aussi notre analyse systématique montre qu'une entité physique peut servir à la fois de dispositif d'entrée et de dispositif de sortie. C'est le cas des briques du Senseboard [Jacob et al 2002] : chaque brique, identifiée par RFID (Radio Frequency Identifier), peut être placée par l'utilisateur sur les cases fixes d'un tableau grillagé à la manière d'une feuille de tableur (dispositif d'entrée). En plus, une unité d'information (celle qui correspond à la case) est projetée sur la brique (dispositif de sortie).

II.3.3. RÔLES D'INTERACTION

Une entité physique impliquée dans un canal d'interaction joue de facto un rôle d'interaction. Si l'on considère les ressources d'interaction initiales, terminales et directes, ce rôle peut être celui de dispositif d'entrée, de dispositif de sortie, ou de dispositif d'entrée et de sortie. Par définition, ces dispositifs sont le lieu d'action et/ou d'observation. Action et observation mettent en jeu, de la part de l'acteur humain, des modalités sensori-motrices (vision, olfaction, etc.).

Dans le cadre de cette étude, nous avons choisi de nous limiter aux modalités visuelle et haptique (action des membres). Dès lors, nous restreignons nos investigations aux entités physiques :

- pouvant jouer le rôle de dispositifs de sortie impliquant, pour l'acteur humain, des observations visuelles : c'est le rôle de surface.
- pouvant jouer le rôle de dispositifs d'entrée impliquant,pour l'acteur humain, des actions au moyen de ses membres : c'est le rôle d'instrument.

La figure 12 explicite la notion de rôle et les relations entre dispositif d'entrée et instrument et entre dispositif de sortie et surface. Nous avons pour cela appliqué le patron de rôle UML [Baümer et al 1997]. Les deux sections qui suivent développent les concepts de surface et d'instrument.

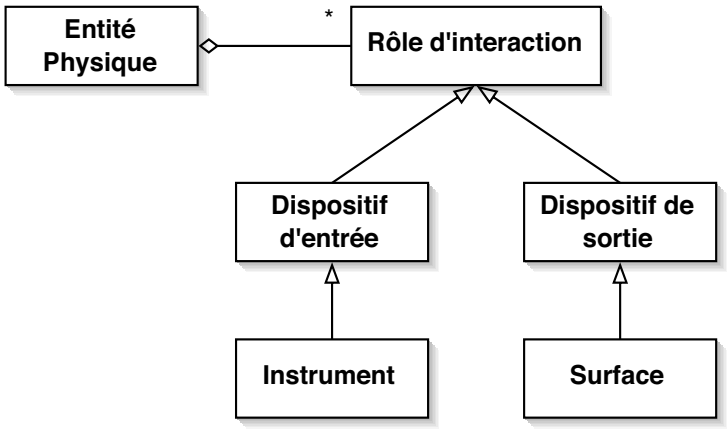

Figure 12. Une entité physique, ressource d'interaction, joue un rôle d'interaction.

II.4. Rôle de surface

Après une définition formelle du rôle de surface (paragraphe 4.1), nous complétons les attributs et les propriétés des entités physiques présentés dans la section 2 en considérant de plus près leur rôle de surface (section 4.2). Nous illustrons ce rôle avec les exemples les plus représentatifs de l'état de l'art.

Définitions :

Le rôle de *surface* est un rôle de dispositif de sortie impliquant, de la part de l'acteur humain, la modalité visuelle. Le rôle de dispositif de sortie, on le rappelle, est tenu par une entité physique intervenant dans

un canal d'interaction de type OA (où O est une observation effectuée par l'acteur humain et où A est effectuée par l'acteur système).

Par commodité d'écriture, on dira qu'une entité physique *est une surface si elle peut jouer le rôle de surface*. Le rôle de surface confère à l'entité la capacité de montrer l'état du système de manière graphique. Dans ce rôle, l'acteur système est en position d'action (pour exprimer l'état du système) et l'acteur humain en position d'observation (pour évaluer cet état).

Si, en outre, l'acteur humain peut agir sur une surface (avec ses effecteurs ou par le biais d'un instrument) et si l'acteur système peut observer ces actions, alors l'entité physique, qui tient à la fois le rôle de surface et de dispositif d'entrée est une *surface interactive*. L'écran usuel de nos stations de travail répond à la définition (action humaine par le biais d'une souris ou d'un clavier). Le Bureau Digital [Wellner et al 1993] (figure 13) et le Tableau Magique [Bérard 1999] (figure 13) aussi (action humaine au moyen des effecteurs doigts).

Figure 13. Sélection au doigt sur le Bureau Digital (extrait d'une vidéo non publiée de Xerox EuroPARC).

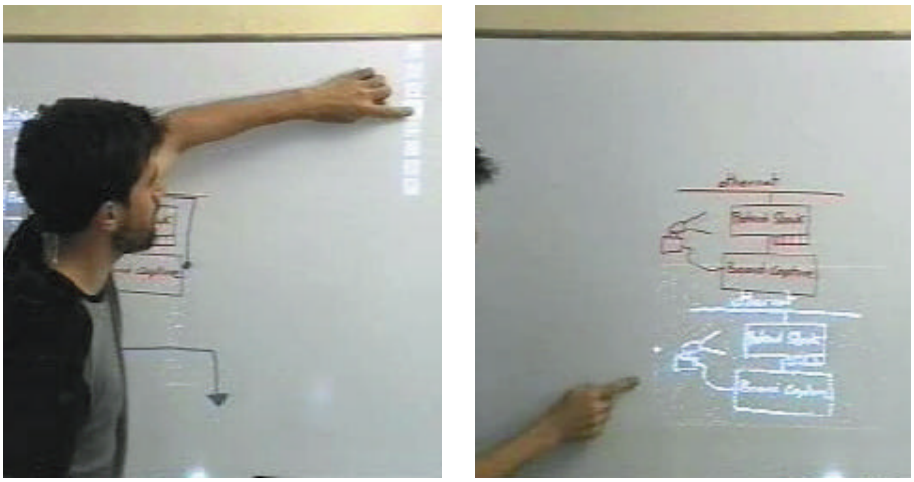

Figure 14. Sélection avec le doigt sur le tableau magique.

Puisque le rôle de surface est véhiculé par une entité physique, complétons l'analyse de la section 2 avec les attributs et propriétés qui orientent l'exploitation de cette entité comme surface. Des exemples d'utilisation tirés de l'état de l'art sont ensuite présentés.

II.4.1. Attributs et propriétés d'une surface

Au-delà des attributs de la section 2, nous proposons les éléments suivants :

- *Usage social* : indique si la surface est utilisable à des fins privées, semi-privées ou publiques.
- *Niveau de partage* : définit si cette surface est utilisable par plusieurs acteurs naturels en même temps ou non, et si oui combien.
- *Actions permises* : écrire de l'information numérique, écrire de l'information physique, sélectionner de l'information numérique et/ou physique, etc. La valeur de cet attribut dépend des capteurs, effecteurs et autres entités physiques utilisées comme instruments.

Selon les besoins des modèles système, les attributs ci-dessus seront représentés de manière numérique ou symbolique. Quelle que soit l'option choisie, ils permettent de déduire, pour une surface donnée, ses propriétés. Celles-ci, on le rappelle, servent le processus de conception, de développement et d'évaluation. En voici quelques exemples :

- *Verticalité/Horizontalité* : indique l'orientation de la surface. Nous verrons ultérieurement son impact sur le rendu et la nature des interacteurs produits sur la surface.
- *Discernabilité* : définit la facilité d'observer le contenu informationnel de la surface.
- *Inscriptibilité* : définit si, de manière générale, il est possible d'écrire sur cette surface. Par exemple, certains murs sont recou-

verts d'une couche de peinture spéciale empêchant toute inscription.

- *Accessibilité* : indique si la surface est atteignable par des effecteurs d'acteur. Trop haute, pour un acteur donné, elle ne pourra être utilisée comme surface d'action, car non accessible aux effecteurs de l'acteur.

II.4.2. Exemples d'utilisation de surface

Les exemples présentés ici illustrent le rôle de surface et son usage en fonction des attributs et des propriétés de l'entité physique support.

Les surfaces interactives sont le plus souvent rigides et opaques. Il en va ainsi pour le Bureau Digital [Wellner et al 1993], les tables et tableaux augmentés de iRoom [Johanson et al 2002b] et de i-LAND [Streitz et al 1999] (figure 15), de même les surfaces augmentées de [Rekimoto et Saitoh 1999] ou le Tableau et la Table Magiques [Bérard 2003].

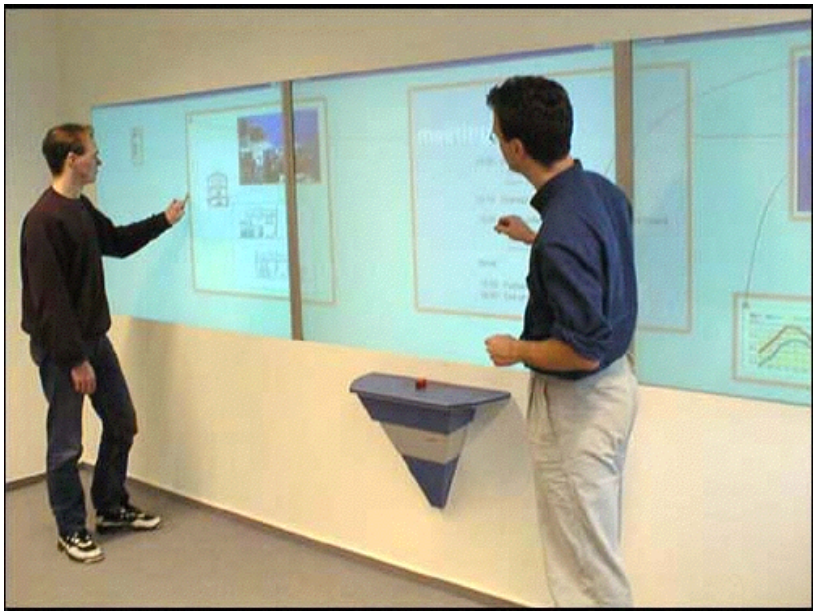

Figure 15. Le Dynawall du projet iLand.

On commence aussi à explorer les solides déformables : le papier électronique développé initialement au MIT et à Xerox PARC, de même que les tissus augmentés du studio créatif de France Télécom R&D [Deflin 2001] ou les fibres électroniques du projet FiCom [FiCom] qui offrent des surfaces pliables et enroulables. L'Illuminating Clay est aussi un exemple original [Piper et al 2002] : l'utilisateur donne forme à de l'argile supposée représenter un modèle 3D d'un terrain. La surface est observée par un capteur de type scanner et un projecteur vidéo affiche sur la surface des informations pertinentes pour l'utilisateur (forme des flux aériens, courbes de niveau, etc.).

Les surfaces transparentes et les surfaces fluides offrent des perspectives d'interaction originale : le Clearboard est le tout premier exemple d'utilisation de surfaces transparentes pour la communication interpersonnelle médiatisée : deux utilisateurs distants ont l'impression de se voir à travers une vitre qui leur sert aussi de surface de dessin [Ishii 1990]. Le rideau d'eau de [Koleva et al 2000] sur lequel des images sont projetées, sert de passage (au sens propre) entre les mondes réels et virtuels.

L'hétérogénéité d'une surface peut favoriser le partitionnement de la surface en zones d'actions et d'observation. Par exemple, l'hétérogénéité de la table GLOSS peut inciter l'utilisateur à exploiter la partie en bois comme surface d'entrepôt (papier, stylo, tasse à café), le tableau sert de surface pour écrire et la surface éclairée du disque de surface d'échange (action et observation) avec le système. Encore faut-il vérifier ces hypothèses d'affordance [Ref Gibson et Norman, papier recent papru ds Interaction] par l'évaluation expérimentale.

Inscriptible, la surface peut recevoir des informations éventuellement effaçables. Ces deux propriétés permettent de traduire qu'il est socialement incorrect de produire des inscriptions à l'encre physique sur un mur public. Mais, il sera admis d'afficher sur un mur public des inscriptions numériques, à la manière des publicistes, des spectacles son et lumière ou encore des murs animés par le mouvement des passants et spectateurs (Maynes-Aminzade et al., 2002).

Nous avons analysé les surfaces. La section qui suit traite des instruments.

II.5. Rôle d'Instrument

Comme pour les surfaces, nous allons successivement proposer une définition, puis étendre les attributs et propriétés d'une entité physique au regard de son rôle d'instrument, et illustrer la discussion avec quelques exemples de l'état de l'art.

Définitions :

Le *rôle d'instrument* est un rôle de dispositif d'entrée impliquant, de la part de l'acteur humain, la modalité haptique. Le rôle de dispositif d'entrée, on le rappelle, est tenu par une entité physique intervenant dans un canal d'interaction de type AO (où A est une action réalisée par l'acteur humain et où O est réalisée par l'acteur système).

Par commodité d'écriture, on dira qu'une *entité physique est un instrument si elle peut jouer le rôle d'instrument*. Le rôle d'instrument confère à l'entité la capacité de faciliter la modification de l'état du système. Dans ce rôle, l'acteur humain est en position d'action (qui agit pour modifier l'état du système) et l'acteur système est en position d'observation de l'instrument (pour interpréter et réagir à ce changement d'état). Dans notre recherche, nous privilégions les instruments qui permettent de modifier le contenu informationnel d'entités physiques jouant le rôle de surfaces.

En Interaction Homme-Machine usuelle, le clavier et la souris sont les instruments privilégiés. Avec les interfaces saisissables (Graspable User Interfaces), tout objet physique, sous réserve qu'il tienne dans la main et qu'il soit observé par le système, peut aussi servir d'instrument (Fitzmaurice et al., 1995). Mais, comme le signifie notre schéma de la figure 2.2, un effecteur humain est une entité physique. Dès lors, du point de vue système, un doigt est un instrument. Aujourd'hui, disponibles sur le marché, les vitrines de magasin s'animent d'une simple pitchenette effectuée avec le doigt [I-vibration].

II.5.1. Attributs et propriétés

En plus des attributs et propriétés présentés dans la section 2, nous proposons les éléments suivants :

- *Actions possibles* : pointer, sélectionner, saisir du texte, déplacer, etc.
- *Précision* : mesure la précision avec laquelle l'acteur humain peut effectuer les actions possibles au moyen de l'instrument. Pour une action de pointage, on sait que le laser est peu précis (Myers et al., 2002).
- *Résolution* : désigne la plus petite variation perceptible de la grandeur à mesurer dans des conditions de mesure données. Par exemple, pour une action de pointage sur une surface, la grandeur en question est les coordonnées d'un point sur une surface. La résolution de l'instrument mesure la distance minimum que l'utilisateur doit imprimer à l'instrument pour que son déplacement soit perceptible.
- *Stabilité* : Un instrument est stable si, en l'absence d'action de la part de l'utilisateur, la grandeur mesurée ne varie pas. Si la souris est un instrument stable, il n'en va pas de même pour les instruments captés par des systèmes de vision par ordinateur. Ces dispositifs sont en permanence soumis à des perturbations qui varient au cours du temps et qui ont pour conséquence de faire osciller les mesures de position d'une entité immobile : perturbation du champ magnétique ou du capteur visuel des caméras.

Parmi les propriétés d'un instrument, nous notons :

- *Adéquation* : dénote la capacité d'un instrument à faciliter les

actions de l'acteur humain. L'adéquation synthétise les degrés d'intégration et de compatibilité introduits par Beaudoin-Lafon pour caractériser l'interaction instrumentale [Beaudoin-Lafon 2000]. La mesure du degré d'intégration se définit pour une tâche donnée, comme le ratio entre le degré de liberté de l'interacteur (1D pour une barre de défilement) et le degré de liberté de l'instrument (par exemple 2D pour la souris). Le degré de compatibilité mesure la similarité entre les actions de l'acteur humain sur l'instrument et la répercussion de cette action sur un interacteur. Cette propriété d'adéquation peut être affinée en critères qui dépendent de la nature des actions permises. Par exemple, la stabilité et la précision sont importantes pour un instrument utilisé pour des actions de pointage.

- *Utilisable à distance* : définit si cet instrument peut être utilisé pour agir sur une surface sans contact avec elle. Cette propriété est vraie pour la souris et le pointeur laser.
- *Maniabilité* : Les attributs de forme, de taille, de poids et de texture de l'entité physique ont un impact sur la maniabilité de l'instrument.
- *Re-assignabilité* : cette propriété renseigne sur le fait que l'instrument peut servir à une autre fonction que celle pour laquelle il a été conçu.

II.5.2. Exemples d'utilisation d'instruments

En raison de leur forme, de leur poids et de leur taille, les stylos du SmartBoard [SmartBoard] ont l'aspect de feutres pour tableau blanc. En vérité, il s'agit de morceaux de plastiques peints d'une couleur. Cette couleur externe définit la couleur de l'encre que le feutre est censé déposer sur le tableau. Les actions possibles sont : écrire, dessiner, pointer, déplacer, etc. Lorsque l'utilisateur ne se sert pas d'un feutre, il doit le déposer dans l'encoche de même couleur prévue à cet effet. Un capteur de contact situé sous l'encoche permet au système de déterminer la présence du stylo. L'absence de stylo dans l'encoche de couleur X permet au système d'identifier X comme couleur courante. Mais si un stylo de couleur rouge est retiré de l'encoche verte, l'encre déposée sera verte ! Si deux encoches sont vides, la couleur de l'encre est indécidable.

Ce problème est renforcé par le fait que la localisation du feutre sur le SmartBoard se fait par l'intermédiaire d'un capteur de pression. Si l'acteur humain imprime plusieurs points de contact sur le tableau, le système croit qu'un seul stylo agit sur la surface au barycentre de ces points. Si le feutre est utilisé lentement alors il est précis. Par contre, si l'utilisateur écrit rapidement, l'échantillonnage temporel des données de pression sur le tableau n'est pas suffisant pour numériser toutes les positions du feutre, ce qui a pour effet de produire un texte incomplet, voire illisible.

Dans les systèmes mettant en jeu un algorithme de vision par ordinateur pour suivre une entité physique, il n'est pas rare de trouver des

problèmes d'instabilité statique, de précision et de résolution spatiale. Par exemple, si l'utilisateur utilise son doigt pour sélectionner et déplacer des interacteurs sur un tableau [Hardenberg et Bérard 2001], il faut que le système utilise un filtre passe bas pour limiter l'instabilité statique et les tremblements créés par l'algorithme de suivi du doigt. Ce filtre augmente la précision de l'instrument, mais limite la résolution spatiale. Dans ce cas, les interacteurs doivent être suffisamment gros pour être atteignables et suffisamment espacés pour éviter les sélections malencontreuses. On retrouve ces problèmes lors de l'utilisation d'un pointeur laser pour interagir à distance [Olsen et Nielsen 2001].

Quand l'instrument est non rigide, sa forme peut être exploitée pour définir la sémantique de l'action qui a lieu sur lui. À l'inverse, être rigide est une qualité souhaitée quand l'instrument est utilisé pour des actions ergotiques [Cadoz 1994] (des actions pour lesquelles de l'énergie est nécessaire et transférée pour transformer l'état d'une autre entité).

II.6. Synthèse

L'ontologie présentée dans ce chapitre identifie les concepts clés de l'interaction dans les nouveaux espaces interactifs en vue d'exprimer des requis pour la conception de logiciel de base en IHM et d'évaluer et comparer les solutions de l'état de l'art. Nos constats sur la nature des espaces interactifs nous ont conduits aux principes suivants :

- placer l'entité physique au centre de l'analyse,
- mettre en symétrie les acteurs humain et système,
- introduire la notion de rôle d'interaction de façon à traduire l'usage opportuniste d'une entité physique selon le point de vue et les attributs et propriétés des entités et acteurs en présence.

Une entité physique joue un rôle d'interaction si elle est impliquée dans un canal d'interaction, c'est-à-dire si elle fait l'objet d'une action A ou d'une observation O dans une chaîne A*O. On dit alors qu'elle sert de ressource d'interaction. Ce rôle d'interaction est un rôle de:

- dispositif de sortie si la ressource d'interaction est un lieu d'action pour l'acteur système et d'observation pour l'acteur humain ; on parle de rôle de surface si la modalité humaine mise en jeu est visuelle ;
- dispositif d'entrée si la ressource d'interaction est un lieu d'action pour l'acteur humain et d'observation pour l'acteur système ; on parle de rôle d'instrument si la modalité humaine mise en jeu est

haptique.

Par commodité, on dira qu'une entité physique jouant le rôle de surface est une surface. De même, une entité physique jouant le rôle d'instrument est un instrument.

En lieu et place de notre ontologie, nous aurions pu faire appel aux cadres conceptuels visant des finalités semblables aux nôtres. Citons « The Input Device Model » (IDM) [Card et al 1990] et Single Display Groupware (SDG) [Streitz et al 1999]. Comme son nom l'indique, IDM n'est concerné que par les dispositifs d'entrée conçus en tant que tel. IDM ignore les surfaces. SDG couvre, quant à lui, la capacité pour une unique surface d'être manipulée par la biais de plusieurs instruments. IDM comme SDG s'inscrivent dans l'esprit d'une interaction usuelle centralisée. Notre ontologie vise le multi-surface et le multi-instrument. Ni IDM, ni SDG n'entrevoient le cas où n'importe quelle entité de ce monde prend ou perd soudainement son rôle d'interaction. Et pour que cela soit possible, l'expression explicite d'attributs, de propriétés et de relations entre ressources d'interaction devient incontournable. C'est ce que notre ontologie fait. Au chapitre suivant, nous analysons le problème des relations spatiales.

Chapitre III *Relations spatiales*

Comme nous l'indiquions dans notre cadre conceptuel, les entités physiques d'un espace interactif peuvent entretenir des relations spatiales. Le schéma UML de la figure 16, complément de la figure 5, représente, sous la forme d'une classe associative, les relations spatiales entre entités physiques.

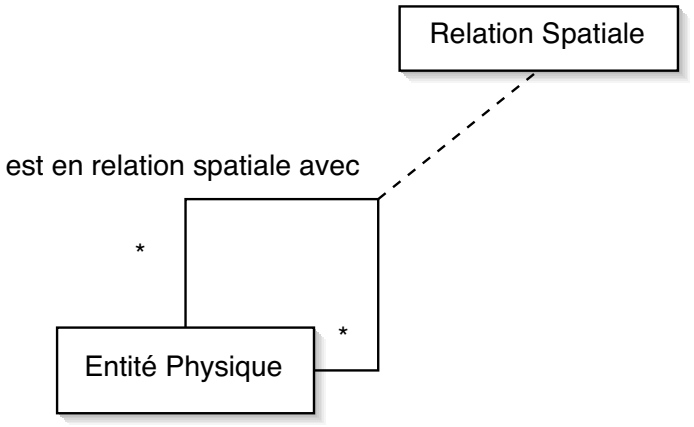

Figure 16. Diagramme UML modélisant les relations spatiales entre entités physiques.

Que des entités entretiennent des liens de dépendance spatiale, est une évidence pour l'acteur humain. Pour l'acteur système, la capacité de localiser des entités physiques, de les suivre (si ces entités sont mobiles) et de les relier dans l'espace est un problème complexe. Bien qu'aujourd'hui la localisation soit un sujet actif de recherche, on ne dispose pas encore de techniques fiables, précises et générales. Les solutions actuelles dépendent étroitement du domaine applicatif dont les exigences de précision, de latence, et de niveau d'abstraction (du numérique au symbolique) sont spécifiques. En effet, localiser un doigt

sur une surface et localiser un individu dans une ville ou chez lui, posent des requis bien différents.

Dans le cadre de cette étude, nous sommes intéressés par la localisation de toute entité physique, et notamment lorsqu'elle intervient comme ressource d'interaction ou comme acteur humain. Par exemple, l'espace FAME doit connaître les positions relatives des surfaces de façon à ce que l'effet d'animation graphique qui exprime la migration d'information de la table vers les murs suive un chemin cohérent avec la géométrie de l'espace. Au chapitre suivant, nous verrons comment la proximité sert de technique d'interaction pour coupler des surfaces. De même, il importe que le système détermine les relations spatiales entre les ressources d'interaction et l'utilisateur. Par exemple, l'information projetée sur une table doit être orientée convenablement vers l'utilisateur en sorte qu'il puisse en lire la teneur sans se tordre le cou !

Dans ce chapitre, nous présentons deux modèles complémentaires représentatifs de l'état de l'art en matière de système de localisation. Nous les avons retenus pour leurs concepts et principes structurants. Puis, au regard de ces concepts et principes, nous analysons des systèmes de localisation appliqués à notre sujet : la mise en relation de ressources d'interaction et d'utilisateurs.

III.1. Modélisation de l'espace physique

De la littérature, nous considérons deux modèles aux propriétés complémentaires : les travaux de Flury et Privat [Flury et Privat 2003] qui visent l'intégration de différents modèles de localisation au sein d'un patron d'architecture en couches et le système hybride de localisation du projet Aura [Jiang et Steenkiste 2002] qui s'appuie sur le concept d'identifiant de localisation.

III.1.1. PATRON D'ARCHITECTURE EN COUCHES

Comme le montre la figure 17, le modèle de Flury et Privat comprend quatre couches de transformation successives, chaque couche définissant un niveau d'abstraction. On relève les niveaux de :

- Localisation discrète et identification. De manière analogue à la couche physique des protocoles réseaux, les techniques de localisation et d'identification de cette couche dépendent du matériel sous-jacent, par exemple, les tags RFID ou la vision par ordinateur.
- Localisation métrique et géométrique. Ici, les informations de localisation sont définies sur la base de positions relatives ou absolues dans un espace de coordonnées métriques. Les informations du niveau inférieur, qui proviennent des capteurs, sont toujours relatives au capteur. À ce niveau d'abstraction, les positions sont rendues

absolues en tenant compte de la position du capteur dans un espace de coordonnées absolues. Les informations relatives correspondent au concept de distance dans un espace métrique, alors qu'un modèle d'espace affine permet de définir une coordonnée de référence ajoutant ainsi des informations de translation. Un modèle euclidien permet de définir les informations de localisation et d'orientation sous forme d'une matrice de rotation et de translation dans un espace de coordonnées de référence orthonormées. Cette couche définit un espace continu de localisation numérique. Nous en verrons l'utilisation dans le chapitre 5 sur la mise en correspondance des espaces numérique et physique.

- Localisation structurelle et relationnelle. À ce niveau d'abstraction, les entités physiques sont liées sur la base de leurs positions relatives pour définir des relations structurelles telles que "à droite de", "à l'intérieur de". Le modèle de localisation correspondant à ce niveau d'abstraction peut prendre la forme de graphes où les nœuds sont des entités physiques et les arcs les relations spatiales entre ces entités. On peut parler aussi de couche hiérarchique et définir le concept de chemin.
- Localisation sémantique. Ici, les informations de localisation sont définies en relation avec le monde de l'application, par exemple, les notions de région, de rue, de bâtiment pour localiser des entités dans une ville. En outre, plusieurs symboles peuvent désigner une même localisation : Grenoble, c'est aussi la préfecture de l'Isère et la capitale des Alpes. Dans cette couche, les informations de localisation peuvent être très abstraites comme "être proche de" ou encore "à moins de 10mn en voiture".

Sur le plan technique, toutes les couches du modèle sont parcourables du bas vers le haut, mais aussi du haut vers le bas lorsque, par exemple, le niveau sémantique exprime des requis de comportement à l'adresse des couches basses.

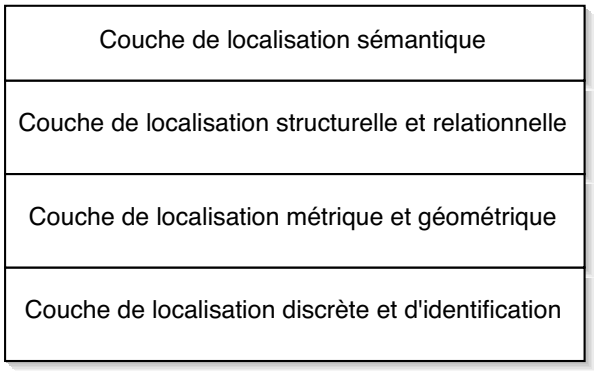

Figure 17. Les différents niveaux d'abstractions du modèle de Flury et Privat.

Ce modèle permet de considérer aussi bien les entités servant de lieu (dits locus) que les entités peuplant ces lieux (les locants). Partant de là, le modèle conduit naturellement à la mise en œuvre de services capables de répondre à des questions directes ou indirectes :

- Les questions directes. Les questions directes, comme "où puis-je trouver cela?", prennent un "locant"en paramètre d'entrée et fournissent en retour un "locus". La question et la réponse peuvent relever d'une sémantique à haut niveau d'abstraction comme "où puis-je trouver un distributeur de monnaie? " conduisant à la réponse : "au bout de la rue à gauche".
- Les questions indirectes. Une question indirecte, du type "qu'y a-t-il par ici?", prend en entrée un "locus" et l'auteur de la question s'attend à obtenir une réponse de type "locant".

Le modèle de Flury et Privat offre un cadre fonctionnel structurant couvrant un large éventail de types de localisation. Comme Location Stack [Hightower et al 2002], il introduit de manière explicite des niveaux d'abstraction qui, s'ils s'avèrent convenir à l'usage, ouvrent la voie à des solutions normalisées. Toutefois, on doit noter que ces modèles n'indiquent pas comment composer en une structure unique les informations de haut niveau d'abstraction avec les informations de bas niveau d'abstraction. Le modèle hybride de localisation du projet Aura, avec sa notion d'identifiant de localisation, apporte des éléments de réponse à ce problème.

III.1.2. IDENTIFIANT DE LOCALISATION

Aura utilise un modèle de localisation hybride qui combine les principes des modèles hiérarchiques et des modèles à base de coordonnées. Dans un modèle hiérarchique, la localisation des entités physiques est définie de manière symbolique, descriptive ou topologique. Cette description correspond aux deux plus hautes couches du modèle de Flury et Privat. Dans un modèle à base de coordonnées, la localisation des entités physiques est définie de manière géométrique ou métrique.

Dans le modèle hybride d'Aura, une localisation s'appelle "identifiant de localisation" ou "Aura Location Identifier" (ALI). Un ALI inclut à la fois les niveaux "localisation structurelle et relationnelle" et "localisation métrique et géométrique". Par exemple, la localisation ALI "ali://cmu/wean-hall/floor3#{1, 0), (-1.5, 0.5), (0, 3), (2, 3.5), (3, 1.5)-(1,5)}" désigne une localisation correspondant au troisième étage du bâtiment Wean Hall à CMU (Carnegie Mellon University). Cette information hiérarchique, analogue à une adresse web, est représentée par la chaîne de caractères qui précède le séparateur #. Ce séparateur est suivi d'une séquence de points qui spécifie un polygone dans un plan parallèle au plan X-Y défini dans l'espace de coordonnées du lieu symbolique (dans notre exemple, le 3ème étage de Wean Hall à CMU). Le dernier couple (1, 5) dénote la hauteur de la localisation.

La structure d'un identifiant ALI répond à la grammaire BNF suivante :

<ALI> ::= [ali://] <Path> ["#" <Position>]
<Position> ::= <Pt-3D> | <Area> ["-" <Height>]
<Path> ::= <Space> {"/" <Space> }
<Area> ::= "{" <Pt-2D> "," <Pt-2D> "," <Pt-2D> {"," <Pt-2D> }"}"
<Height> ::= "(" <Float> "," <Float> ")"
<Pt-3D> ::= "(" <Float> "," <Float> "," <Float> ")"
<Pt-2D> ::= "(" <Float> "," <Float> ")"
<Space> ::= <Char> {<Char>}
<Float> ::= ["+" | "-"] <Digit> {<Digit>} ["." <Digit> {<Digit>}]
<Char> ::= <Alphanum> | "-" | "_"
<Alphanum> ::= <Alpha> | <Digit>
<Alpha> ::= "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z" | "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
<Digit> ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

Le modèle définit des opérateurs sur les ALI, par exemple l'opérateur de distance.

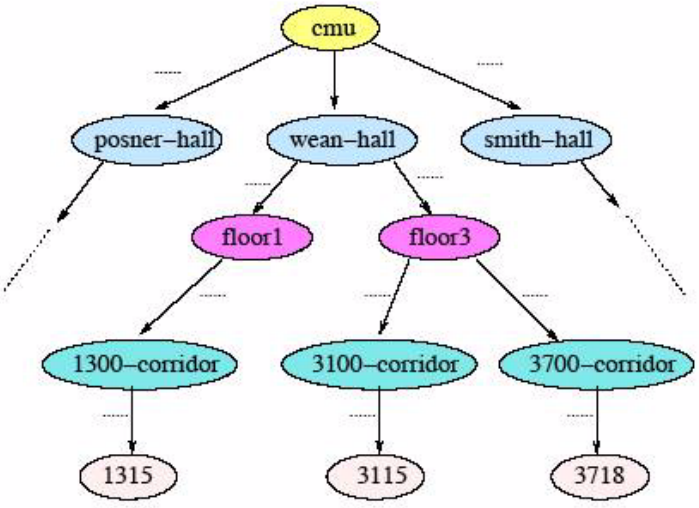

Figure 18. Modèlisation hiérarchique du campus de Carnegie Mellon dans Aura.

Toutes les informations de localisation sont centralisées dans une base de données interrogeable à distance. Dans sa version actuelle, cette base contient les informations de localisation du campus de Carnegie Mellon. Comme le montre la figure 18, le campus (un locus) est structuré de manière hiérarchique en bâtiments qui, à leur tour, comprennent des étages, chaque étage comportant plusieurs couloirs qui

donnent accès à des bureaux. Comme le montre la figure 19, chaque nœud de l'arbre (ou espace) définit un espace géométrique 3D dans lequel se situent les espaces géométriques des nœuds fils (ou sous-espaces).

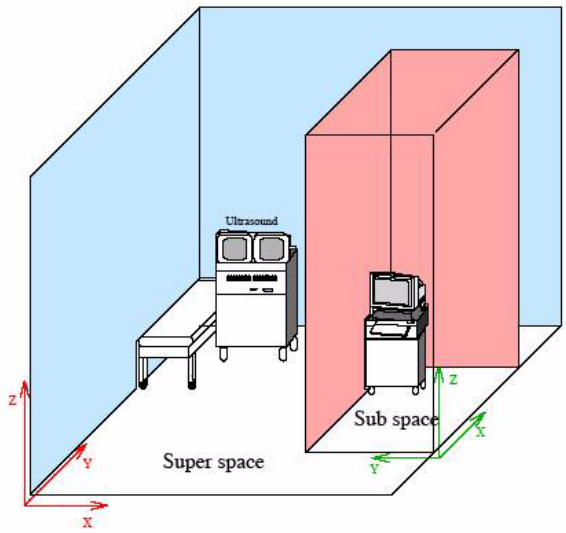

Figure 19. Modélisation géométrique. Un sous-espace possède son propre espace géométrique mais localisé dans l'espace de référence de l'espace parent.

Si le modèle hybride d'Aura permet de raisonner dans l'espace géométrique comme dans l'espace symbolique et de passer aisément de l'un à l'autre, il suppose implicitement que les informations de localisation changent peu. Cette solution passe difficilement à l'échelle lorsque les entités physiques sont mobiles comme les acteurs humains et leur téléphone portable, ou alors il faut se contenter d'une localisation à gros grain, à l'échelle d'une pièce par exemple.

Les deux modèles ici présentés montrent la nécessité d'une cohabitation harmonieuse des représentations de localisation de nature symbolique, structurelle et géométrique. Dans les systèmes où la mobilité est un facteur clé, le problème du passage à l'échelle n'est pas encore résolu. Dans ce qui suit, nous nous limitons à la localisation de ressources d'interaction en milieu fermé où le facteur d'échelle n'est pas pertinent, mais où la précision peut l'être.

III.2. Localisation d'entités physiques en milieu fermé

III.2.1. NETWORKED SURFACES

Networked Surfaces [Hoffmann et Scott] vise l'alimentation en énergie, la connexion au réseau et la localisation de dispositifs mobiles comme les assistants personnels et les téléphones portables. Ce système s'appuie sur l'hypothèse que, pour se servir de ces objets, les utilisateurs les posent sur une table.

Dans Networked Surfaces, la table est équipée de tuiles métalliques conductrices rectangulaires de 12,2 sur 1,7 cm. La figure 20 en montre l'agencement où 3mm séparent chaque bord de tuile. Chaque entité physique mobile est équipée au dos de tampons circulaires de taille inférieure à 3mm disposés en cercle. Chaque tuile, comme chaque cercle de tampons, est gérée par un contrôleur dédié. Lorsqu'une entité physique est posée sur la table, chaque tuile fournit à son contrôleur la liste des tampons en contact (voir l'illustration de la figure 21). Un contrôleur central fusionne les informations de contact reçues de tous les contrôleurs de tuile pour fournir une information de localisation sur le tapis avec une précision de 2 cm pour les coordonnées dans le plan de la table et de 5 degrés pour l'orientation. Il est possible d'améliorer la précision en augmentant le nombre de tampons par entité et en diminuant la taille des tuiles.

Figure 20. Networked Surfaces en action.

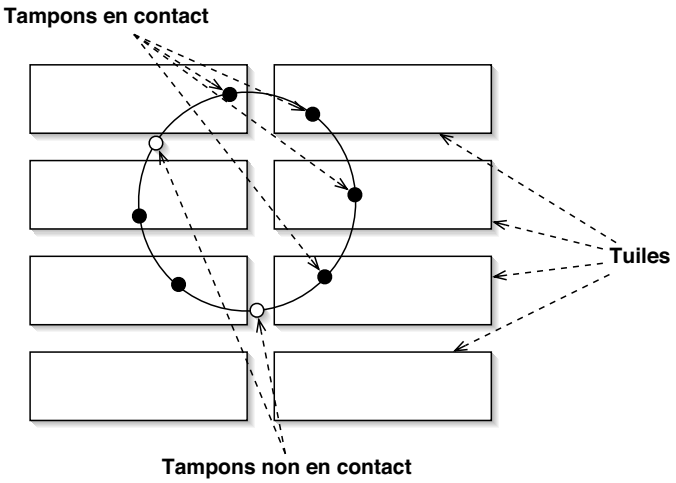

Figure 21. Agencement des tuiles de la table et relations avec les tampons disposés sur les objets mobiles.

"Networked Surfaces" illustre la transformation d'informations de localisation de type structurel (le tampon A est sur la tuile B) en une information de localisation de type géométrique (l'entité Z se situe en X,Y selon une orientation O dans le repère du tapis de tuiles). Comme les SmartIts [Smart-Its] et bien d'autres [Richardson et al 2003] notamment, il utilise comme couche basse de localisation une technique matérielle de type contact. L'exemple qui suit s'appuie sur une technique matérielle différente : la vision par ordinateur.

III.2.2. EASYLIVING

EasyLiving [Brumitt et Shafer 2000] [Brumitt et al 2002] concerne l'étude de nouveaux services informatiques pour le domicile, par exemple, comment "contacter quelqu'un chez lui pour lui rappeler un rendez-vous important". Pour cet exemple, le problème revient au choix du dispositif d'interaction le mieux adapté pour cette tâche parmi les écrans, le téléphone fixe ou portable, haut-parleurs, ou assistant personnel.

Une première approche à ce problème consisterait à utiliser les préférences prédéterminées de l'utilisateur : clignotement de l'écran, sonnerie du téléphone fixe, puis du portable, enfin message sur l'assistant personnel. Mais l'écran est inopérant si l'utilisateur ne peut le voir ; il en va de même pour le téléphone si l'utilisateur ne peut pas l'entendre. L'approche retenue dans EasyLiving est de fonder le choix du dispositif sur la localisation de l'utilisateur par rapport aux entités physiques utilisables comme ressources d'interaction.

L'espace EasyLiving identifie et suit l'utilisateur dans un intérieur au moyen d'un système de vision par ordinateur. L'intérieur, réduit à une

pièce, est équipé de deux grands écrans de projection et d'un ordinateur de bureau usuel. Lorsqu'une personne rentre dans la pièce, elle est automatiquement localisée et se voit attribuée un identifiant. Lorsqu'elle se logue sur l'ordinateur, l'identifiant devient le login. Elle peut alors consulter son courrier électronique. Si la personne vient s'asseoir en face de l'un des grands écrans, EasyLiving fait migrer l'interface graphique de l'ordinateur vers le grand écran.

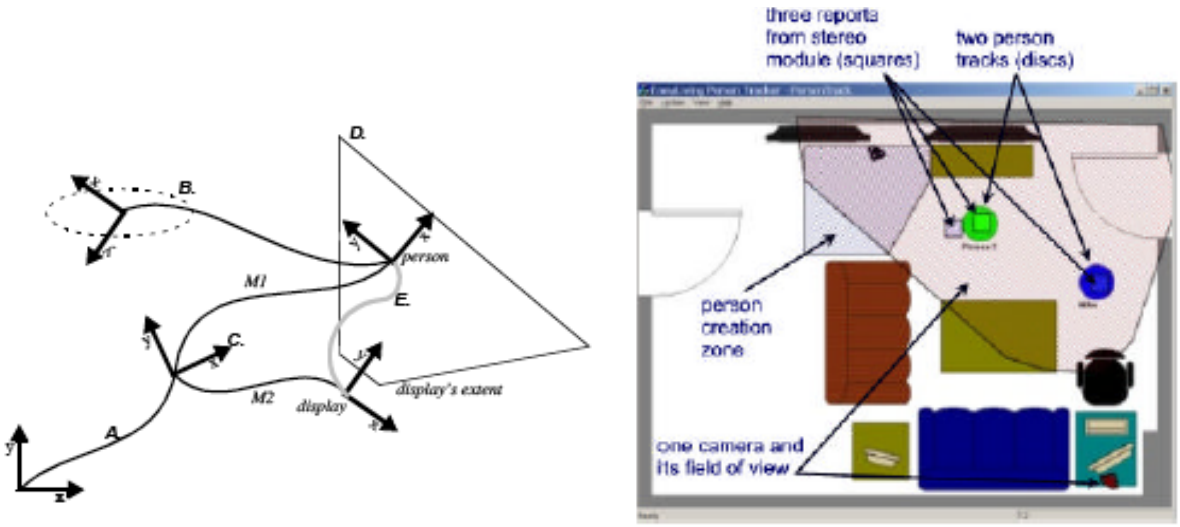

Figure 22. Le modèle de localisation dans EasyLiving.

Le système de localisation d'Easyliving inclut un modèle géométrique 3D de la pièce qui sert de référence à la localisation des individus (voir figure 22). Mais chaque entité physique suivie (les utilisateurs) a un repère qui lui est propre (figure 22 à gauche). Ce faisant, le système maintient la position des ressources d'interaction par rapport à l'utilisateur (figure 22 à droite) Ce modèle géométrique, largement utilisé en vision par ordinateur, correspond à la couche de localisation géométrique de Flury et Privat. Et l'identification des utilisateurs correspond à la couche localisation discrète et identification au même titre que le calibrage des caméras.

Si EasyLiving connaît les relations spatiales entre un utilisateur et les ressources d'interaction, il ne couvre pas les relations spatiales entre ressources d'interaction. Avec PointRight, on assiste à la situation inverse.

III.2.3. PointRight

PointRight [Johanson et al 2002a] est un sous-système du «roomware» iRoom [Johanson et al 2002b]. PointRight permet l'utilisation d'une souris et de son pointeur pour agir sur le contenu de n'importe quelle surface d'une pièce. Lorsque le pointeur sort d'une surface, il pénètre

sur la surface qui lui est contiguë. Pour cela, PointRight modélise les relations spatiales entre les surfaces dans un plan (se référer à la figure 23).

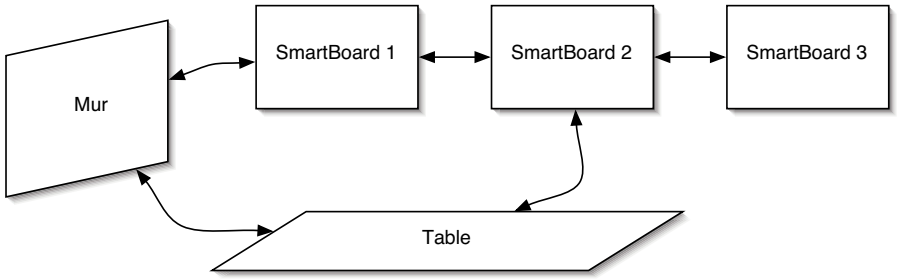

Figure 23. Configuration spatiale des surfaces interactives dans PointRight. Les relations spatiales sont représentées par des flèches entre des points de connexion.

PointRight ne découvre pas les relations spatiales, mais les lit dans un fichier de configuration. La figure 24 en donne un exemple simplifié. Ce fichier comprend une liste d'écrans, de machines et de connexions. Une connexion exprime les relations spatiales entre deux écrans et de là, les transitions valides du pointeur entre écrans. Elle est représentée par la spécification du côté (gauche, haut, droit, bas) et du segment de ce côté (exprimé en millimètre) par lequel le pointeur peut transiter. Un même écran peut faire l'objet de plusieurs connexions à condition que les segments de connexion ne se superposent pas. Le modèle de localisation est donc de type géométrique.


```
[screens]
iw-smartboard1 = 146.3 x 109.7
iw-smartboard2 = 146.3 x 109.7
iw-smartboard3 = 146.3 x 109.7

[machines]
iw-smartboard1 = 1024 x 768
iw-smartboard2 = 1024 x 768
iw-smartboard3 = 1024 x 768

[edges]
iw-smartboard2:left:0.0:109.7 = iw-smartboard1:right:0.0:109.7
iw-smartboard2:right:0.0:109.7 = iw-smartboard3:left:0.0:109.7
```

Figure 24. Exemple simplifié de fichier de configuration dans PointRight.

Dans l'exemple de la figure 24, les écrans sont des SmartBoard (tableau électronique) modélisés par leurs caractéristiques physiques (taille en cm) et numériques (nombre de pixels). Il existe une connexion entre le tableau 2 et le tableau 1 et une connexion entre le tableau 2 et le tableau 3. Ces tableaux sont de même taille (146.3 x 109.7 cm), et leur bord gauche-droit respectivement, sont alignés horizontalement et toute la longueur des bords sert de lieu de transition au pointeur (0.0 :109.7). Ici, les informations de localisation sont de type géométrique et sont traduites en relation structurelle du type gauche-droite, haut-bas.

Dans PointRight, l'espace physique est restreint aux surfaces d'interaction sans pour autant refléter les relations spatiales réelles de ces surfaces. En effet, une connexion indique seulement un cheminement autorisé pour les pointeurs souris. Dans l'exemple, une connexion spécifie que lorsque le pointeur sort par la gauche du tableau 2 il doit rentrer sur le tableau 1 par sa droite. La connexion ne traduit pas le fait qu'entre ces deux tableaux, il puisse y avoir un espace. Contrairement à EasyLiving qui calcule des relations dans l'espace physique réel, PointRight utilise des relations entre des espaces numériques. En outre, ces relations ne sont pas calculées, mais prédéfinies dans un fichier de configuration. Nous présenterons en détail au chapitre 5 le problème des relations entre espaces numérique et physique.

III.3. Synthèse

Cette brève revue de l'état de l'art montre que la localisation et la gestion des relations spatiales restent un problème ouvert. Si l'on commence à disposer de modèles structurants comme le Location Stack ou le modèle de Flury et Privat, la couche basse de perception, proche du matériel, constitue le véritable verrou technique :

- Absence de fiabilité des capteurs et hétérogénéité,
- Passage à l'échelle du nombre de capteurs non démontré,
- Perception multimodale balbutiante, même si le thème de la fusion multicapteur est étudié depuis longtemps en robotique,
- Instrumentation de l'espace physique : où placer les capteurs sans faire intrusion,
- Comment éviter d'instrumenter l'utilisateur,
- mais aussi le problème d'éthique avec la protection de l'espace privé.

Dans cette thèse, nous faisons l'hypothèse que ces problèmes seront, à terme, résolus. Nous sommes utilisateur des solutions techniques en matière de localisation et de calcul de relations spatiales que nous

exploitons de manière explicite dans notre logiciel de base pour IHM en milieu ubiquiste. En particulier, nous exploitons la proximité comme technique d'expression du couplage d'entités pouvant jouer le rôle de ressources d'interaction. Cet aspect important de l'interaction dans des espaces interactifs est traité au chapitre suivant.

Chapitre IV *Couplage*

Dans la vision de l'informatique ubiquiste introduite au chapitre 1, nous avons vu que l'utilisateur devenait créateur d'espaces interactifs. L'utilisateur peut y prêter, emprunter et assembler des entités physiques au gré des opportunités. Pour mémoire, reprenons l'illustration du chapitre 1 (voir figure 25). Prêt, emprunt et assemblage s'appuient sur le couplage d'entités physiques que le système autorise. Le schéma UML de la figure 26, extrait de la figure 5, représente le couplage entre entités physiques sous la forme d'une classe associative.

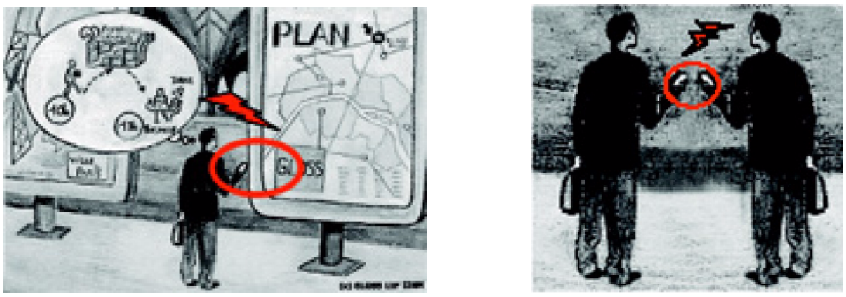

Figure 25. À droite, deux amis assemblent leur PDA. À gauche, le voyageur à proximité d'un mur actif est automatiquement relié aux services du mur interactif.

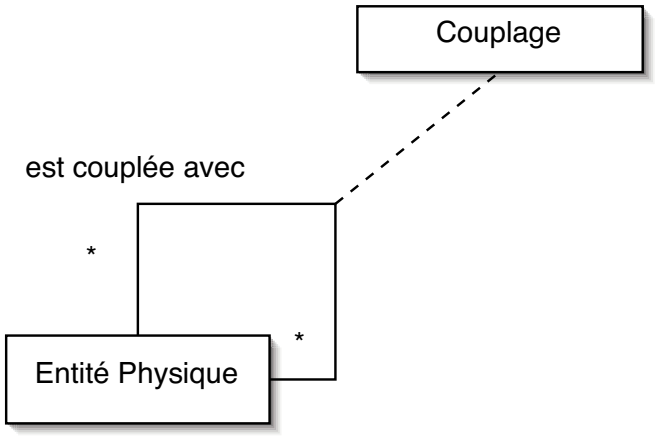

Figure 26. Diagramme UML modélisant le couplage entre entités physiques.

Dans ce chapitre, nous explorons de manière rationnelle la notion de couplage étudiée jusqu'ici de manière exploratoire. Dans la section qui suit, nous en proposons une définition, puis nous illustrons cette notion en analysant l'état de l'art aussi bien des IHM conventionnelles pour lesquelles il existe déjà des formes de couplage, que pour les IHM relevant de l'informatique ubiquiste (sections 2 et 3). Si le couplage est une dimension importante de l'informatique ubiquiste, il convient de s'interroger sur sa mise en œuvre technique mais aussi sur son appropriation par l'utilisateur. En particulier, l'utilisateur comprend-il qu'un couplage est possible, peut-il en prédire les effets, sait-il le réaliser, et peut-il en évaluer l'état ? Ces questions indiquent que le couplage n'est pas un phénomène instantané, mais qu'il a des états et que ces états doivent être observables et contrôlables de manière souple et robuste par l'utilisateur. En sorte de répondre rationnellement à ces questions, nous proposons en Section 4 et conjointement avec [Barralon et al 2004], un cycle de vie du couplage. Nous sommes alors en mesure d'analyser les propriétés d'interaction en relation avec le couplage (section 5),

Notre étude est centrée sur les entités physiques jouant le rôle de ressources d'interaction. Par commodité d'écriture, nous parlerons de couplage de ressources d'interaction.

IV.1. Définition

Définition :
Le couplage de deux ressources d'interaction est l'action de lier ces

ressources de manière à ce qu'elles opèrent conjointement pour fournir une nouvelle fonction.

De manière plus formelle, soient :

- R, un ensemble fini non vide de ressources d'interaction,
- F, un ensemble fini non vide de fonctions, alors,
- l'ensemble C des couplages possibles entre les ressources de R s'écrit F : C : RxR -> F.

Nous illustrons maintenant le couplage de ressources d'interaction dans les IHM conventionnelles avec leurs biens fondés et leurs limitations.

IV.2. Couplage et IHM conventionnelles

Dans les IHM conventionnelles, les écrans jouent le rôle de surface tandis que le clavier et les dispositifs de pointage comme la souris, jouent celui d'instrument. Nous allons analyser successivement le couplage d'écrans, le couplage clavier-souris, puis le couplage hybride surface-instrument.

IV.2.1. COUPLAGE DE SURFACES

Les cartes graphiques des stations de travail permettent la connexion simultanée de plusieurs écrans à une même station de travail. Les systèmes de fenêtrage ont à charge de projeter l'espace logique d'affichage sur la configuration physique d'écrans. Si le système sait détecter la présence des écrans, il est incapable d'en déterminer la configuration spatiale. Comme le montre la figure 27, ce travail revient à l'utilisateur qui doit spécifier que tel écran est à droite, ou à gauche, de l'écran principal.

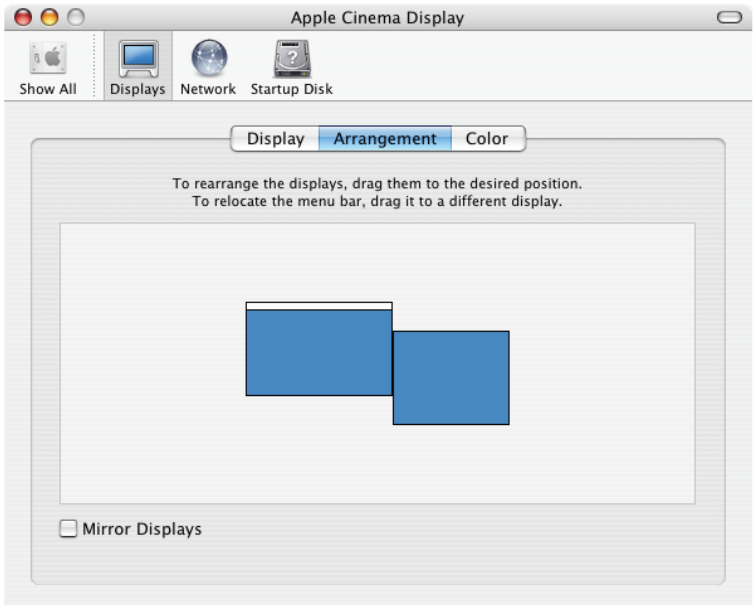

Figure 27. Spécification des relations spatiales dans le couplage d'écrans sur Mac OS X.

Le couplage entre écrans est effectif lorsque les deux conditions suivantes sont remplies : connexion physique filaire effectuée, disposition spatiale des écrans spécifiée. En retour, l'utilisateur obtient la fonction "extension de la surface d'affichage". Si l'utilisateur branche un projecteur vidéo, il définit une fonction de couplage en mode miroir, redirection, ou extension.

IV.2.2. COUPLAGE D'INSTRUMENTS

Dans les IHM WIMP actuelles, l'instrument de pointage (souris) et l'instrument de saisie de caractères (clavier) sont couplés dès lors qu'ils sont connectés au calculateur. La fonction résultant de ce couplage est immuable : l'instrument de pointage détermine le "focus" de l'instrument de saisie. Il lui confère la capacité de multiplexage spatial.

Notons que les touches "flèche" et "tabulation" du clavier permettent aussi de modifier le point d'insertion, mais il s'agit là d'un couplage hybride du clavier avec une surface.

IV.2.3. COUPLAGE HYBRIDE INSTRUMENT-SURFACE

Le couplage entre un instrument de pointage et une surface est effectif dès que l'instrument est physiquement connecté au calculateur et qu'il est déplacé. Dans le cas d'un écran tactile, le couplage a lieu par contact entre la surface et un instrument de forme pointue (qui peut être un doigt).

Le couplage instrument de pointage-surface offre la fonction "désignation d'un point de la surface". Il se traduit par la présence d'un curseur sur la surface dont les déplacements sont asservis aux mouvements de

l'instrument de pointage dans les limites de la surface : le curseur ne sort jamais de l'écran. En conséquence, les actions de l'utilisateur sont nécessairement confinées dans la surface. Notons que si plusieurs instruments de pointage sont connectés à un même ordinateur, un seul, à un instant donné, est couplé à l'écran. Il en va de même pour les écrans tactiles.

Un instrument de saisie de caractères est couplé à l'écran par transitivité de son couplage avec la souris : couplage écran-souris + couplage souris-clavier. Puisque la souris ne peut désigner que des points de l'écran et que la surface d'action du clavier est réduite aux points atteignables par la souris, il est impossible de saisir du texte dans un interacteur hors du bureau. Cette restriction apparente relève d'un bon principe : celui de rendre observables les effets des actions de l'utilisateur.

IV.2.4. SYNTHÈSE

En résumé, notre analyse des IHM conventionnelles montre l'existence de couplages dont les effets sont prédéfinis. Certains de ces effets sont directement observables comme l'asservissement du curseur aux mouvements de la souris qui traduit le couplage écran-souris. Inversement, si plusieurs instruments de pointage sont connectés à l'ordinateur, un seul, à un instant donné est couplé à l'écran. Cette application surjective peut, en situation d'interaction multi-utilisateur par exemple, paraître quelque peu restrictive. Le couplage clavier-souris est, quant à lui, immuable, et n'est pas directement observable. La configuration des écrans introduit une tâche articulatoire dont la résolution exige de découvrir sur le bureau l'interacteur donnant accès au formulaire de configuration. Enfin, les actions utilisateur, confinées dans l'écran, ne permettent pas d'étendre l'espace d'action au-delà du visible. Qu'en est-il pour les IHM non conventionnelles ?

IV.3. Couplage et IHM non conventionnelles

Une caractéristique que partagent les supports matériels de ces nouvelles IHM est l'absence de fil comme le remarque [Edwards et Grinter 2001]. Dans ces conditions, le couplage doit s'appuyer sur de nouvelles métaphores. Dans les exemples qui suivent, les gestes synchronisés et la proximité (relation spatiale particulière) servent systématiquement de fondement à ces métaphores, qu'il s'agisse de couplage de surfaces, d'instruments, ou de couplage hybride.

IV.3.1. COUPLAGE DE SURFACES

Gestes synchronisés

Avec la technique des gestes synchronisés de Hinckley (synchronous gesture), l'utilisateur peut construire dans le plan une mosaïque de surfaces [Hinckley 2003] (voir figure 28). Chaque surface est une tablette équipée d'accéléromètres qui détectent les collisions de tablettes (cas a de la figure) ainsi que le bord d'impact. Une paire de capteurs de pression placés sur les côtés gauche et droit déterminent si la tablette est tenue. Le couplage de deux surfaces est effectif si l'utilisateur cogne les surfaces bord à bord. L'état du couplage est rendu observable par un clip sonore métallique et deux flèches qui se font face sur chacune des surfaces pendant 2 secondes (cas b de la figure). Le découplage a lieu dès que l'une des surfaces est retirée. Le système produit alors un son mat et des flèches brisées apparaissent quelques secondes.

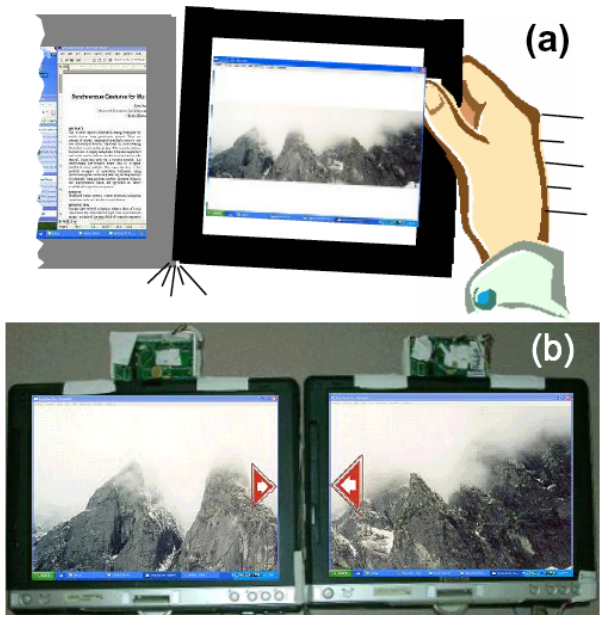

Figure 28. Couplage de surfaces par gestes synchronisés [Hinckley 2003].

Contrairement aux IHM conventionnelles, la configuration spatiale des surfaces est ici construite par le système à partir des signatures des accéléromètres. Comme pour les IHM conventionnelles, la fonction obtenue est une "extension de surface d'affichage", mais la projection de l'espace logique d'affichage sur la configuration physique dépend de la nature des gestes synchronisés. Si une surface vient heurter une seconde surface immobile (dite surface de base), alors la surface mobile emprunte la base : son espace logique d'affichage s'étale sur la base. Si les surfaces se heurtent de manière symétrique, les espaces

d'affichage sont échangés. Comment l'utilisateur peut-il prédire la différence ?

Proximité Les tablettes des ConnecTables sont équipées de capteurs et d'antennes RFID (Cf. figure 29). Le couplage a lieu lorsque l'antenne d'une tablette voit le tag de l'autre. Une seule configuration spatiale est autorisée : écrans haut contre haut. La fonction obtenue est une "extension de surface d'affichage" avec fusion des deux espaces logiques sources pour ne former qu'un.

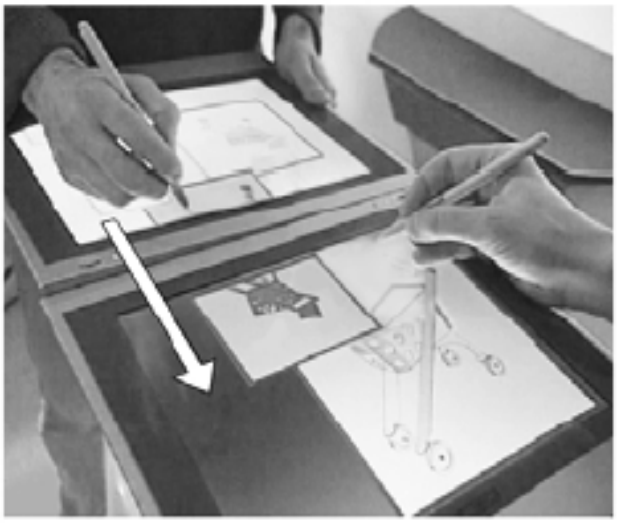

Figure 29. Le couplage de surfaces dans ConnecTables.

Les Datatiles permettent la construction d'un espace de services par assemblage de tuiles sur un plateau (figure 30). Une tuile est une petite surface dotée d'un RFID et d'un écran plat à cristaux liquides sur lequel est inscrit le nom du service auquel elle donne accès [Rekimoto et al 2001]. Un couplage entre deux tuiles a lieu si ces tuiles sont placées sur le plateau, si les services auxquels elles donnent accès sont compatibles et si l'utilisateur trace au stylo un trait de jointure entre les deux tuiles. Le tracé sert à lever les ambiguïtés car deux tuiles ont pu être placées sur le plateau par inadvertance. Il permet aussi de spécifier le sens des échanges de données entre les tuiles ainsi que le mode, discret ou continu, de ces échanges.

Figure 30. Le couplage de surfaces dans DataTiles.

Un retour visuel sur les tuiles traduit l'état du couplage et le mode de communication. La fonction obtenue est une "extension de service" dont la nature dépend du service porté par chacune des tuiles. Par exemple, en couplant une tuile donnant accès à la carte météorologique du jour avec une tuile de navigation dans le temps, on obtient la carte météorologique du jour de son choix. Le découplage a lieu si une tuile est retirée du plateau ou si un trait vertical est tracé à la jointure des deux tuiles.

IV.3.2. COUPLAGE D'INSTRUMENTS

Les interfaces bi-manuelles s'appuient nécessairement sur le couplage d'instruments [Chatty 1994]. Ce couplage est le plus souvent statique. C'est le cas des Toolglass [Bier et al 1993], du metaDesk [Ullmer et Ishii 1997], du Holowall [Matsushita et Rekimoto 1997] comme du système de navigation de Hinckley [Hinckley et al 1998]. Pour ce dernier exemple, un TouchPad et une souris TouchMouse sont couplés de manière permanente. La fonction obtenue est un "pan zoom" graphique.

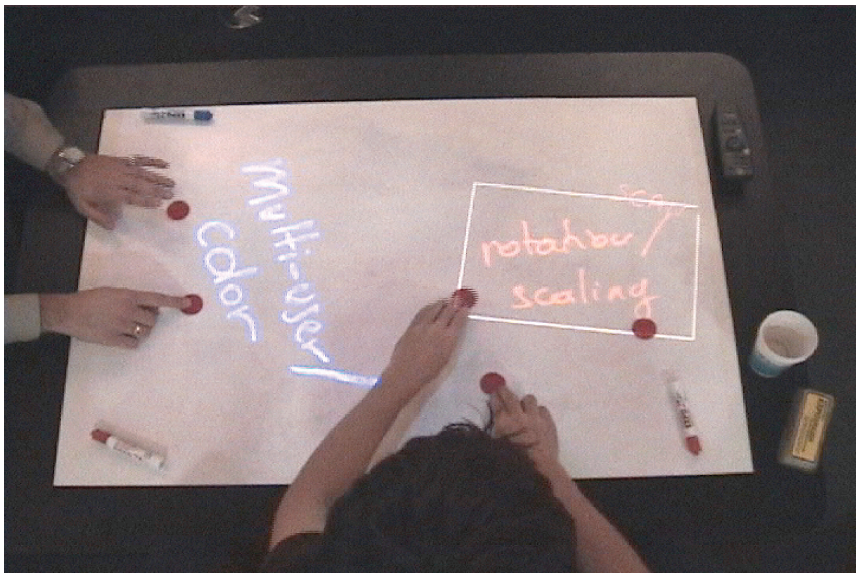

Figure 31. La Table Magique et deux paires de jetons couplés.

Avec la Table Magique, le couplage d'instruments est dynamique [Bérard 2003]. Comme le montre la figure 31, des jetons en plastique suivis par un système de vision par ordinateur, permettent de manipuler un espace d'information projeté sur la table. Le couplage de deux jetons a lieu lorsque ces jetons sont mis en contact puis éloignés l'un de l'autre (voir figure 32). On assiste ici à des gestes synchronisés qui jouent sur la proximité (contact puis éloignement) d'entités physiques. Le système montre l'existence du couplage par l'affichage d'une boîte élastique asservie aux mouvements des jetons. L'utilisateur a alors à sa disposition la fonction "sélection d'un patch" de la table délimité par la boîte élastique comme le montre le couplage de deux jetons sur la table magique afin d'obtenir la fonction sélection.

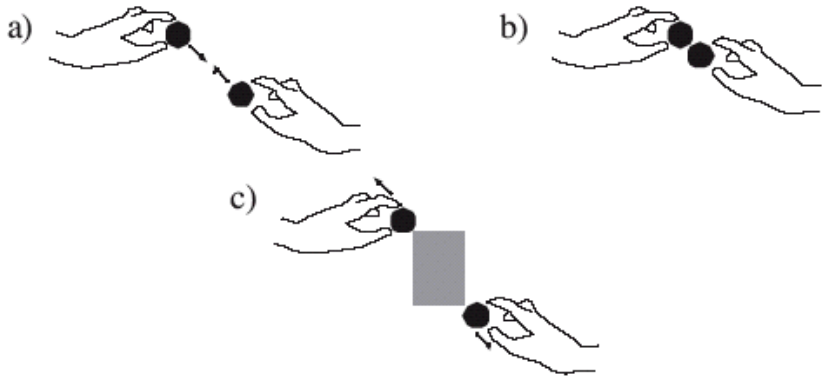

Figure 32. Couplage de deux jetons sur la table magique afin d'obtenir la fonction sélection. L'utilisateur rapproche les jetons l'un de l'autre (a) jusqu'au contact (b). La fonction de sélection est alors activée. L'utilisateur peut alors sélectionner les informations se trouvant dans un rectangle compris entre les deux jetons.

Le découplage des jetons prend fin lorsque l'un des jetons disparaît (en le retirant de la table ou en le masquant avec la main). Plusieurs couples de jetons peuvent être définis simultanément et fonctionner en parallèle.

IV.3.3. COUPLAGE HYBRIDE INSTRUMENTS-SURFACES

Dans les IHM non conventionnelles, le couplage hybride instruments-surfaces s'appuie sur le couplage préalable de dispositifs électroniques dotés de surfaces et d'instruments. On retrouve les techniques d'interaction par gestes synchronisés et par proximité.

Gestes synchronisés

Dans SyncTap, les chocs de Hinckley sont remplacés par l'appui simultané de deux touches dédiées placées chacune sur les dispositifs à relier [Rekimoto et al 2003 (1)]. La fonction qui résulte de ce couplage est une "ouverture de communication ad-hoc" entre deux entités identifiées. À partir de là, le développeur peut définir des couplages de surfaces et d'instruments.

Par exemple, Rekimoto montre comment SyncTap peut remplacer le suivi par vision par ordinateur utilisé dans les Surfaces Augmentées [Rekimoto et Saitoh 1999] pour détecter la présence de PC portables sur une table augmentée et en déduire la configuration spatiale. Avec la vision artificielle, les relations spatiales entre les écrans de PC portables sont automatiquement calculées, mais exige d'instrumenter l'environnement. Le couplage est représenté par la projection d'un halo sur la table centré sur la position du portable (voir figure 33) . Avec SyncTap, l'utilisateur doit agir de manière conventionnelle sur la souris d'un PC particulier non mobile (dit maître) : par exemple, en faisant

sortir le curseur de la souris du PC-maître par la gauche, il indique la présence d'un portable à sa gauche.

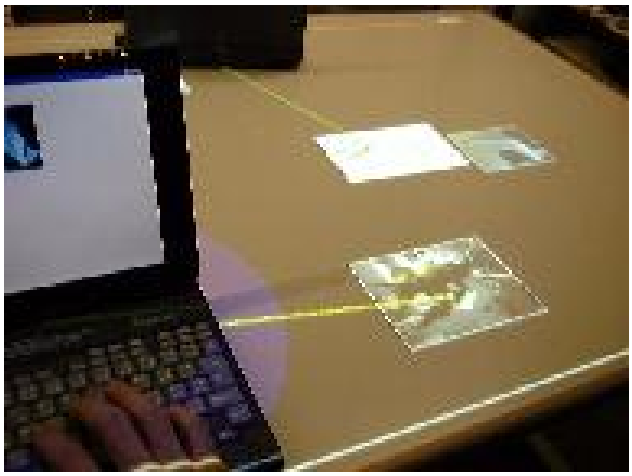

Figure 33. Représentation de l'existence d'un couplage (halo violet) et Hypercurseur dans les Surfaces Augmentées.

Une fois la configuration spatiale des surfaces connue, l'utilisateur dispose de la fonction "extension de surface d'affichage" avec fusion des espaces logiques sources. Comme le montre la figure 33, l'HyperCursor opère un lien visuel lors de la migration d'entités numériques entre les surfaces (fonction dite d'hyperdragging).

Proximité En interaction proximale, un couplage de dispositifs a lieu lorsque ces entités, équipées de capteurs RFID et d'infrarouge, sont suffisamment proches et alignées [Rekimoto et al 2003 (2)]. Ce couplage résulte de l'ouverture d'un canal de communication d'un nouveau type, le near-field channel, qui assure la sécurité, et l'authentification. Le couplage offre, comme précédemment la fonction de "communication ad-hoc" sur laquelle il est possible de définir de nouvelles fonctions de couplage.

Le couplage de SmartIts [Holmquist et al 2001] s'appuie à la fois sur la proximité et les gestes synchronisés. Le couplage a lieu si deux SmartIts (par exemple un porte-monnaie et une clef de voiture augmentés) sont placés l'un contre l'autre et secoués ensemble. La fonction obtenue peut être un "avertissement" dès que les entités couplées s'éloignent trop l'une de l'autre. En l'occurrence, ne pas oublier un porte-monnaie Smart-It lorsque l'on part sans les clefs de la voiture.

IV.3.4. SYNTHÈSE Par couplage opportuniste de ressources d'interaction, l'utilisateur, nous l'avons vu, obtient de nouvelles fonctions. L'idée est séduisante à condition que sa mise en œuvre respecte les principes fondamentaux de l'IHM. Avec les réseaux sans fil, nous perdons la physicalité pré-

gnante des connexions filaires. En compensation, les métaphores auxquelles nos exemples de l'état de l'art ont recours s'appuient sur des notions familières : la proximité et la simultanéité de gestes sur des entités.

Toutefois, on s'est encore peu interrogé sur l'affordance perçue (au sens de Norman [Norman 1999]) des entités en question. Autrement dit, de quels indices l'utilisateur dispose-t-il pour déterminer a priori que deux entités "vont ensemble" techniquement, sémantiquement et socialement ? Et comment peut-il prédire la fonction qu'il obtiendra par une action de couplage ? Pour répondre de manière fondée à ces questions, nous avons étudié et formalisé le cycle de vie d'un couplage de ressources d'interaction. Pour mieux comprendre ce problème, Barralon propose un cycle de vie du couplage de ressources d'interaction modélisé par un automate [Barralon et al 2004] que nous reprenons ici.

IV.4. Cycle de vie du couplage de ressources d'interaction

Nous nous plaçons dans le cadre de couplages dynamiques avec l'hypothèse suivante : l'utilisateur doit disposer à tout instant des "indices nécessaires et suffisants" pour juger de l'existence ou de la potentialité de couplage entre deux ressources d'interaction. En réponse à ce questionnement, nous disons que le couplage passe par une suite d'états observables et contrôlables par l'utilisateur. Chaque état se définit au moyen de critères. Nous posons ces critères avant de présenter les conditions de transition entre les états, puis le cycle de vie proprement dit d'un couplage.

IV.4.1. UN ÉTAT : TROIS CRITÈRES DE VÉRITÉ

Soient :

- R l'ensemble des ressources d'interaction,
- F l'ensemble des fonctions de couplage et
- C l'ensemble des couplages.

Soient :

- r1 et r2, deux ressources d'interaction éléments de R,
- f inclu dans F l'ensemble des fonctions obtenues par le couplage c

élément de C de r1 avec r2, noté (r1, c, r2).

Un état de (r1, c, r2), est défini par l'ensemble des trois critères de vérité suivants :

- r1 est (couplée /non couplée) avec r2 pour c,
- r1 est (verrouillée /non verrouillée) pour r2 et pour c,
- r1 est (couplable /non couplable) à r2 pour c, ou bien r1 est (découplable/non découplable) de r2 pour c.

Ou, si l'on utilise les prédicats équivalents :

- Couplée (r1, c, r2) est vrai si et seulement si F est non vide. Si F est vide, Couplée (r1, c, r2) est faux et NonCouplée (r1, c, r2) est vrai.
- Verrouillée (r1, c, r2) est vrai si l'état de r1 ne permet pas de changer l'état de (r1, c, r2). Ce prédicat traduit l'indisponibilité sociale de r1 pour entrer ou sortir du couplage c avec r2. Par exemple, un utilisateur ne veut pas joindre à l'écran public, l'écran privé de son PDA utilisé à des fins personnelles. L'état de (r1, c, r2) est contraint au statu quo jusqu'à ce que Verrouillée (r1, c, r2) devienne faux ou, de manière duale NonVerrouillée (r1, c, r2) vaut vrai.
- Couplable (r1, c, r2) est une expression de prédicats autres que Couplée (r1, c, r2) et Verrouillée (r1, c, r2). Elle regroupe toutes les conditions nécessaires à la réalisation de (r1, c, r2), autres que les conditions sur l'existence de F ou sur le verrouillage de r1 vis-à-vis de r2 dans c. Par exemple, la compatibilité de forme et de rôle, peuvent être modélisées par Couplable. De manière symétrique, Découplable exprime toutes les conditions, autres que les conditions sur l'existence de F ou sur le verrouillage, nécessaires à la réalisation de la destruction de (r1, c, r2).

IV.4.2. TRANSITIONS

Les transitions entre états correspondent à des événements dont la prise en compte par le système modifie la valeur de l'un des critères. Ces événements traduisent soit la demande de couplage/découplage (c'est-à-dire la demande de lancement ou de suspension des fonctions de f), soit le verrouillage/déverrouillage de ressource, soit leur couplabilité/découplabilité .

IV.4.3. CYCLE DE VIE

L'automate de la figure 34 modélise le cycle de vie de (r1, c, r2). Un automate similaire modélise celui de (r2, c, r1).

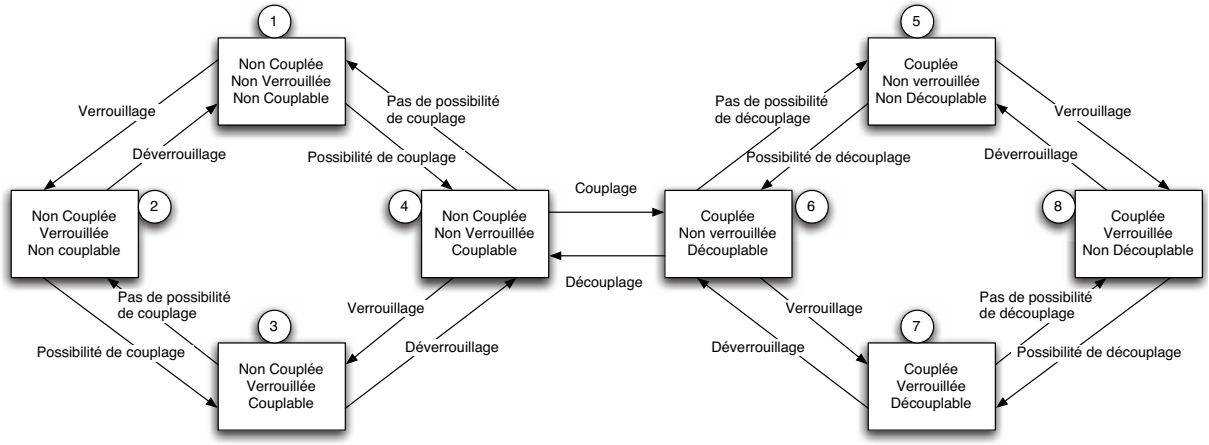

Figure 34. Automate du cycle de vie du couplage (r1 c r2). Par souci de clarté du schéma, la fermeture transitive entre états n'est pas représentée, mais les états 1 et 3, 2 et 4, 5 et 7, 6 et 8 sont évidemment reliés.

Le cycle de vie comprend deux sous-automates, l'un formé par les états 1, 2, 3, 4 pour lesquels Couplée (r1, c, r2) est vrai, l'autre par les états 5, 6, 7, 8 pour lesquels Couplée (r1, c, r2) est faux, les états 4 et 6 servant de passerelle entre les deux sous-automates. L'état 4 correspond à la situation où toutes les conditions sont réunies pour que (r1 c r2) soit réalisé. Il ne manque qu'un événement de demande de couplage pour entrer dans l'état 6.

À titre d'illustration, déroulons l'automate pour le cas de deux jetons j1 et j2 de la Table Magique. A l'initialisation du système, (j1, c, j2) est dans l'état 4 : j1 n'est pas couplé à j2, il n'est pas verrouillé vis-à-vis de j2, il est coupable à j2 puisqu'il est sur la table, qu'il est de même couleur, de même forme et de même taille que j2 (le suivi des jetons s'appuie sur un système de vision par ordinateur fondé sur un modèle de couleur). La mise en contact de j1 et j2 signifie, pour le système, une demande de couplage. (j1, c, j2) entre dans l'état 6 et la fonction "sélectionner un patch" est disponible.

Si l'utilisateur sélectionne un patch, puis masque j2 de la main, (j1, c, j2) revient en 4. Mais comme j1 est maintenant attaché à un patch (couplage instrument-surface), j1 est verrouillé pour (j1, c, j2), et (j1, c, j2) passe de l'état 4 à l'état 3. (j1, c, j2) reviendra à l'état 4 dès que j1 sera détaché de la surface, c'est-à-dire dès qu'il sera masqué par la main. S'il est verrouillé pour (j1, c, j2), j1 n'est pas verrouillé pour (j1, c', j2) où c' donne accès aux fonctions de rotation, de zoom et de destruction de patch. (j1, c', j2) est dans l'état 4. Il passe en 6 dès que l'utilisateur pose j2 à l'intérieur du patch auquel j1 est attaché.

L'analyse de ces quelques exemples de couplage montre combien cette opération, simple de principe, soulève de nombreux problèmes potentiels d'utilisabilité. Nous les analysons plus avant dans la section qui suit.

IV.5. Couplage et propriétés d'interaction

Nous reprenons ci-dessous l'analyse de l'utilisabilité du couplage présentée dans [Barralon et al 2004], selon les deux grandes dimensions, souplesse et robustesse de l'interaction introduites dans [Gram et Cockton 1996].

IV.5.1. SOUPLESSE DE L'INTERACTION

La souplesse de l'interaction couvre une dizaine de propriétés. Nous avons retenu les plus pertinents d'entre eux : l'atteignabilité, la non-préemption, l'interacation multifilaire, l'adaptabilité et la migrabilité de tâche.

Atteignabilité

L'atteignabilité dénote la "capacité du système à permettre à l'utilisateur de naviguer dans l'ensemble des états observables du système". Appliquée à notre problème, l'atteignabilité signifie que l'utilisateur doit pouvoir atteindre les états (couplé, verrouillé, couplable) et leur inverse (découplé, déverrouillé, découplable). Il convient donc que l'utilisateur dispose de techniques d'interaction pour atteindre chacun de ces états. Par exemple, pour le verrouillage, Hinckley [Hinckley 2003] propose de masquer avec la main la tablette dont on ne veut pas partager le contenu.

Non-préemption

Le prochain but souhaité par l'utilisateur est directement atteignable. Pour notre problème, on retiendra que les états souhaités par l'utilisateur doivent être accessibles à tout instant et ceci par des trajectoires d'interaction optimisées. C'est le cas des ConnecTables [Tandler et al 2001] qui peuvent toujours être couplées ou découplées en un seul geste.

Interaction multifilaire

L'interaction multifilaire dénote la "capacité du système à gérer de front la réalisation de plusieurs tâches". Dans le contexte de notre étude, cette propriété traduit la capacité du système, comme la Table Magique, à traiter plusieurs couplages simultanément ; ou bien, comme cela peut se produire en interaction proximale, plusieurs utilisateurs demandent en même temps le couplage d'entités avec une même entité cible. Soit le système gère l'accès concurrent et l'exprime, soit les conflits sont résolus de manière sociale.

Adaptabilité L'adaptabilité dénote la "personnalisation du système sur intervention explicite de l'utilisateur". Aucun exemple de l'état de l'art sur le couplage de ressources d'interaction, n'illustre cette propriété. Couplage/Découplage, Verrouillage/Déverrouillage etc., sont conventionnels et immuables.

Migrabilité de tâche La migrabilité de tâche dénote la "capacité de délégation dynamique de tâches entre le système et l'utilisateur ou entre utilisateurs. C'est un changement dynamique de l'acteur(s) responsable(s) de l'accomplissement de la tâche". Au regard du couplage, cette propriété s'interprète comme la possibilité de déléguer une partie du couplage au système. Par exemple, les claviers et souris d'une même plate-forme sont automatiquement couplés dès le lancement du système.

IV.5.2. ROBUSTESSE DE L'INTERACTION

Les propriétés liées à la robustesse incluent l'observabilité, l'honnêteté, la curabilité et la prévisibilité.

Observabilité L'observabilité dénote la "capacité du système à rendre perceptible l'état pertinent du système". Cette propriété, appliquée à l'automate de couplage, impose de rendre observable l'ensemble des états de l'automate. En particulier, nous avons vu dans l'analyse de l'état de l'art que les états « couplable » et « non couplable » ne sont généralement pas observables.

Honnêteté L'honnêteté dénote la "capacité du système à rendre observable l'état du système sous une forme conforme à cet état et qui engendre une interprétation correcte de la part de l'utilisateur". Autrement dit, l'état du couplage doit non seulement être observable à tout instant, mais ce rendu doit être compris correctement par l'utilisateur. Nous verrons par exemple avec I-AM (notre solution technique) où l'utilisateur dispose de plusieurs souris et claviers, le problème de représenter le couplage des claviers-souris qui peut varier au cours du temps.

Curabilité La curabilité dénote la "capacité pour l'utilisateur de corriger une situation non désirée". Dans l'automate du couplage, cette propriété se traduit par la possibilité de découpler et de déverrouiller à tout instant. La curabilité impose une vigilance toute particulière sur les conséquences du découplage. Par exemple, considérons deux surfaces S1 et S2 gérée par deux machines distinctes. Supposons que S1 soit couplée à des instruments et à S2. Si l'on découple S1 de S2, doit-on pour autant "rapatrier" les instruments de S1 vers la plate-forme de S2, c.-à-d. les recoupler à S2 ou doit-on les perdre et les laisser poursuivre leur vie avec S1?

Prévisibilité La prévisibilité dénote la “capacité pour l'utilisateur de prévoir, pour un état donné, l'effet d'une action”. Ici, l’utilisateur doit anticiper les conséquences d’un couplage ou d’un découplage. Nous avons vu chez Hinckley que, selon le geste, le couplage de deux tablettes ne fournit pas le même résultat et que ce résultat, pour un utilisateur néophyte, n’était pas prévisible. Ou encore, prenons l’exemple de deux souris P1 etP2 couplées entre elles, et de deux claviers C1 et C2 couplés entre eux. Que se passe-t-il si l’utilisateur couple P1 à C1 ? Le couplage est-il transitif ? Si tel est le cas, P1 devient également couplé à C2 ; de même, P2 est couplé à C1 et C2. Ou encore, si P1 est couplée à une surface S1, et P2 est couplée à une surface S2. Quels sont les effets de bord pour les souris suite au couplage de S1 et S2 ? Pour l’utilisateur, ces conséquences sont-elles prévisibles?

IV.5.3. SYNTHÈSE

L’analyse des propriétés de souplesse et de robustesse de l’interaction appliquées à l’automate du couplage de ressources d’interaction, montre, avant toute implémentation, qu’il s’agit d’un problème plus complexe qu’il n’y paraît. Il montre aussi que l’état de l’art s’est peu préoccupé du problème ou s’y est attaché de manière plutôt exploratoire que rationnelle. Il va de soi que la satisfaction de toutes ces propriétés d’utilisabilité n’est pas systématiquement nécessaire. Pour notre solution technique que nous présentons au chapitre 7, nous avons placé la priorité d’une part, sur l’atteignabilité et la non-préemption en faveur de la souplesse d’interaction, d’autre part sur l’observabilité, l’honnêteté, et la curabilité pour la robustesse.

IV.6. Synthèse

Dans ce chapitre, nous avons étudié le couplage de ressources d’interaction. Dans les IHM conventionnelles, ce problème est ignoré, puisque les ressources d’interaction sont stables par leurs rôles, leurs types et leur nombre. Il suffit que les utilisateurs apprennent et découvrent les conventions. Cette confortable stabilité ne tient plus si, dans la vision informatique ambiante, l’utilisateur bâtit à façon son espace d’interaction à la faveur des besoins et des ressources disponibles. Dans ces conditions, le couplage devient un problème qu’il convient d’étudier avec attention.

En réponse à ces nouvelles exigences, nous avons proposé un cycle de vie du couplage dont chaque état est une combinaison de trois critères : existence de couplage, verrouillage et couplabilité. À partir de là, il convient de permettre à l’utilisateur de naviguer dans ces états en respectant au mieux les règles de souplesse et de robustesse de l’interaction. Nous avons vu que certains types de relations spatiales comme la proximité constituent un critère de premier choix pour décider ou non

du couplage de ressources d'interaction et, réciproquement, de leur découplage. Nous avons également vu qu'il peut y avoir d'autres propriétés incluses sous le terme de couplabilité.

Nous avons également vu que le couplage de surfaces pouvait couvrir plusieurs sémantiques. À chaque sémantique correspond une mise en correspondance spécifique entre les espaces numériques et l'espace physique formé par les surfaces couplées. Par exemple, dans les ConnectTables, le couplage de deux surfaces conduit à une fusion d'espaces numériques tandis que chez Hinckley, les espaces numériques peuvent être échangés. La mise en correspondance entre les espaces numériques et physiques se traduit par une fonction de projection. Cet aspect est traité au chapitre suivant.

Chapitre V *Projection de l'espace logique*

L'ontologie du chapitre 2 indique que les acteurs peuvent changer le statut des entités physiques de l'environnement en leur conférant le rôle de ressource d'interaction, et notamment celui de surface. Le rôle de surface, on le rappelle, est tenu par une entité physique intervenant dans un canal d'interaction de type OA, où O est une observation effectuée par l'acteur humain au moyen de la modalité visuelle, et où A est une action effectuée par l'acteur système par le biais d'un effecteur (par exemple, un projecteur vidéo) pour afficher des pixels. Techniquement, cette action système se traduit par une fonction de projection d'un espace numérique (ou logique) sur un support physique. Dans le cadre de cette étude, ce support, lieu de fusion des mondes physique et numérique, correspond aux surfaces d'entités du monde physique. Le monde numérique, quant à lui, est peuplé d'interacteurs. La projection d'espaces numériques consiste donc à projeter des interacteurs sur une surface, et plus généralement sur un ensemble de surfaces.

La projection d'espace numérique n'est pas une notion nouvelle : elle est à la base des IHM graphiques conventionnelles où l'écran sert de surface de projection. Dans ce chapitre, nous analysons plus avant le problème de la projection dans l'esprit des IHM migrables et distribuées. Le diagramme UML de la figure 35 en synthétise la modélisation. On notera que la fonction de projection est représentée par une classe associative entre les classes Interacteur et Projection. Ce faisant, nous exprimons la possibilité de contrôler finement le rendu d'un espace numérique, non pas, par l'application de la même fonction à tous les interacteurs de cet espace, mais par l'application d'une fonction, spécifique à chaque interacteur. L'analyse de l'état de l'art en matière de projection montrera l'intérêt de ce choix.

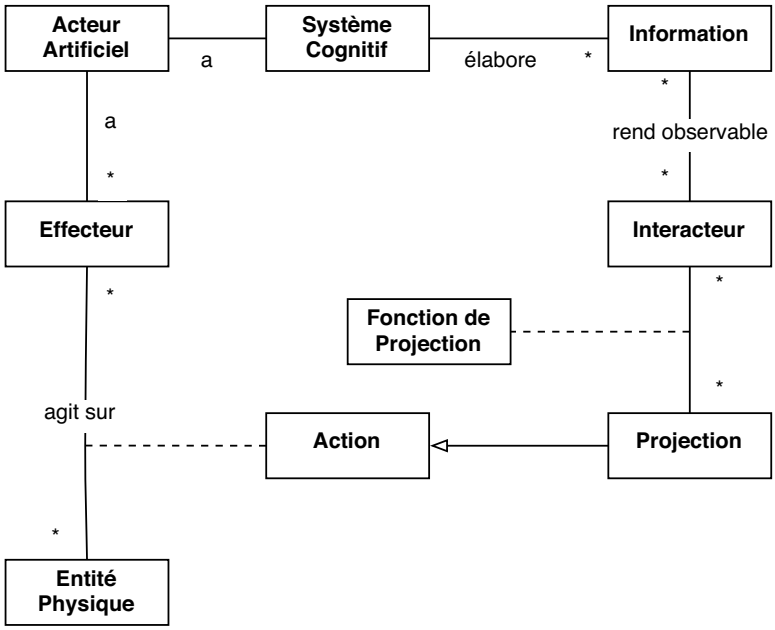

Figure 35. Modélisation UML de la projection d'interacteur sur une entité physique.

Après une définition formelle de la projection, nous analysons les différentes classes d'interfaces graphiques centralisées sur une surface que nous illustrons par des exemples représentatifs de l'état de l'art. Puis, en nous appuyant sur les concepts de relation spatiale et de couplage des chapitres précédents, nous étudions les interfaces graphiques migrables et distribuables sur plusieurs surfaces.

V.1. Définition

Dans le contexte de l'interaction multisurface, la fonction de projection d'un interacteur I projette le système de coordonnées de I défini sur une échelle de pixels logiques dans le système de coordonnées défini en millimètres, de la surface physique cible. Cette fonction s'exprime sous la forme d'une transformation affine définie par la matrice 3*3 de la figure 36.

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} Zx*\cos(theta) & Sx*\sin(theta) \\ -Sy*\sin(theta) & Zy*\cos(theta) \end{bmatrix} * \begin{bmatrix} X \\ Y \end{bmatrix} + \begin{bmatrix} Tx \\ Ty \end{bmatrix}$$

ou encore

$$\begin{bmatrix} X' \\ Y' \\ 1 \end{bmatrix} = \begin{bmatrix} Zx*\cos(theta) & Sx*\sin(theta) & Tx \\ -Sy*\sin(theta) & Zy*\cos(theta) & Ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$$

Figure 36. La fonction de projection exprimée sous forme de matrice.

Le paramètre Tx de la matrice exprime une translation selon l'axe des x alors que le paramètre Ty définit une translation sur l'axe des y. Les paramètres Zx et Zy définissent le coefficient de zoom, respectivement, sur l'axe des x et des y. Les paramètres Sx et Sy définissent le coefficient de glissement (« shear » en anglais) sur l'axe des x et des y, respectivement. Le coefficient de glissement permet de créer des vues en perspective. Enfin, le coefficient thêta définit la rotation appliquée à l'interacteur sur l'axe des z. La fonction de projection peut aussi s'exprimer en coordonnées homogènes au moyen d'une matrice 4*4. Cependant, cette représentation plus complexe n'apporte pas d'avantages notables si l'on s'en tient au seul problème de modélisation.

V.2. Projection d'interacteurs sur une surface

Les paramètres de la fonction de projection autorisent différents types de projection et de là, différentes classes d'IHM graphiques. Parmi ces IHM, on distingue les interfaces conventionnelles, les interfaces zoomables, les interfaces rotatives et les interfaces avancées.

V.2.1. IHM CONVENTIONNELLES

Les concepteurs de boîtes à outils graphiques conventionnelles ont fait l'hypothèse que les utilisateurs en situation d'interaction se trouvaient face à une surface rectangulaire verticale (l'écran). L'origine du repère d'affichage est donc placée en haut et à gauche de la surface (ce qui simplifie la programmation), et le positionnement des interacteurs est spécifié dans ce repère.

Avec les boîtes à outils conventionnelles, le programmeur peut redéfinir dynamiquement la taille et la position de chaque interacteur, mais aucune homothétie n'est appliquée à la représentation de l'information véhiculée à l'intérieur de la surface de l'interacteur. Par exemple,

l'empreinte d'un bouton à l'écran peut être agrandie sans toutefois impliquer que le texte affiché sur le bouton soit agrandi. Les boîtes à outils conventionnelles ne permettent donc pas de réaliser des IHM zoomables.

Les paramètres de zoom (Zx et Zy) permettent aux systèmes d'exploitation de gérer différentes résolutions d'affichage. Toutefois, la modification de la résolution d'affichage entraîne le changement d'échelle de tous les interacteurs de l'espace numérique. En conséquence, un interacteur ne peut pas changer de taille indépendamment de ses voisins.

Avec les boîtes à outils graphiques conventionnelles, aucun paramètre de la fonction de projection ne peut être spécifié aisément par le développeur. La seule souplesse offerte est le libre positionnement de l'interacteur sur la surface. La matrice de projection est la suivante : [Zx, 1, 0][1, Zy, 0][0, 0, 1]. Le paramètre de rotation thêta est fixé à 0 et le paramètre de glissement à 1. Les paramètres Zx et Zy de zoom sont des constantes qui représentent la taille d'un pixel logique en mm sur la surface. Ces paramètres traduisent la résolution d'affichage gérée par le système d'exploitation et la taille physique de l'écran connue seulement du constructeur.

Autrement dit, un interacteur positionné sur une échelle de pixels en x=100 et y=150 par programmation a une position effective en millimètres sur l'écran de x'=100*Zx et y'=150*Zy. Cette position effective est définie par rapport au coin haut gauche de l'écran. Le concepteur n'a donc qu'une idée relative, non pas absolue, de la taille et du positionnement réels de l'interacteur sur la surface ; s'il est possible par programmation de connaître la résolution d'affichage, la taille physique de l'écran reste une inconnue. Par conséquent, le seul moyen de contrôler la taille effective des interacteurs réside dans la définition d'interfaces zoomables.

V.2.2. IHM ZOOMABLES

Avec Pad++, Bederson est l'un des pionniers de la conception d'interfaces zoomables [Bederson et Hollan 1994]. En permettant de changer dynamiquement le facteur de zoom associé à la projection d'un interacteur sur une surface, Pad++ permet la navigation dans de grands espaces d'information multi-échelle. Dans [Furnas et Bederson 1995], les auteurs montrent les différents types de navigation multi-échelle possibles ainsi que leurs implications sur les valeurs des coefficients de la matrice de projection. Un exemple de navigation est donné sur la partie droite de la figure 37. Elle montre que dans le cas des interfaces zoomables, la matrice de projection s'exprime comme suit : [Zx*zx, 1, 0][1, Zy*zy, 0][0, 0, 1] où zx et zy sont les facteurs de zoom en x et en y spécifiables pour chaque interacteur. Zx et Zy, on le rappelle, traduisent la résolution d'affichage et la taille physique de l'écran utilisé.

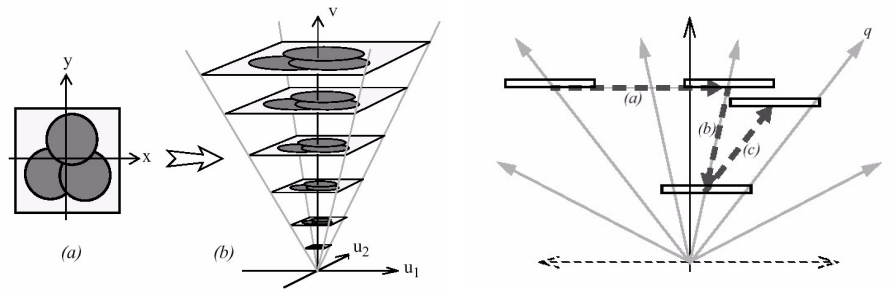

Figure 37. Navigation multi-échelles dans un espace logique. La partie de gauche illustre l'intérêt du facteur de zoom et la partie de droite donne un exemple de navigation dans un espace logique. tout d'abord avec translation seule (a) puis avec facteur d'échelle seul (b) et enfin avec les deux en même temps.

Pad++ offre en outre un type d'interface zoomable singulier où le facteur d'échelle est le même sur l'axe des x et des y (Zx = Zy). Cette contrainte entre Zx et Zy définit un changement d'échelle homogène, l'interacteur n'étant pas déformé lors de la projection. Un exemple illustrant cette situation est donné sur la partie gauche de la figure 37. De plus, ici chaque interacteur jouit de sa propre fonction de projection. Les interacteurs les plus importants peuvent donc être mis en avant.

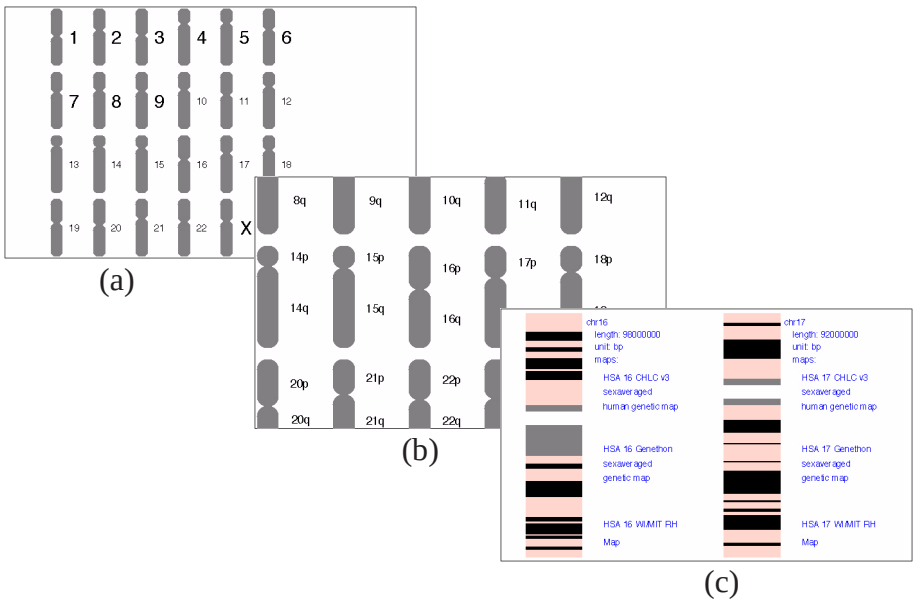

Figure 38. Modification du rendu de l'interface graphique (ici des chromozomes) suivant la valeur du facteur d'échelle (zoom sémantique). A l'échelle nominale les chromozomes sont visualisés de manière simplifiée (a). Puis quand on augmente l'échelle on distingue nettement deux parties sur chaque chromozome (b). Et si on augmente encore l'échelle on voit apparaître les composants des chromozomes.

À l'inverse, Zomit [Pook 2001] n'offre pas cette souplesse, mais apporte une autre fonctionnalité : le zoom sémantique. Celui-ci consiste à adapter la présentation de l'interacteur suivant le facteur de zoom en ajoutant ou en enlevant des détails pour avoir plus ou moins d'information sur les concepts du domaine. Un exemple est donné sur la figure 38. Cependant l'intérêt du zoom sémantique est minimisé par la présence de sauts imprévisibles lors d'une navigation multi-échelle. Il convient donc d'utiliser ce type d'interface avec précaution.

En synthèse, les paramètres de zoom permettent la création d'interfaces graphiques adaptées à l'exploration de grands espaces d'information. Toutefois, ils ne remettent pas en cause l'idée que l'utilisateur, en situation d'interaction, se trouve en face à un écran vertical. Avec l'informatique ubiquiste, cette hypothèse ne tient plus puisque toute surface, même à l'horizontale, sert potentiellement de surface de projection. Voyons comment le facteur de rotation peut être exploité à cette fin.

V.2.3. IHM ROTATIVES

Dans le cas des surfaces horizontales, comme les tables, les utilisateurs ont la possibilité de prendre place tout autour. Ainsi, une interface graphique projetée sur une table n'est lisible que par quelques-uns. En conséquence, le système doit tenir compte de la position de l'utilisateur autour de la table et utiliser cette position pour projeter l'IHM face à lui ou alors laisser à l'utilisateur la possibilité de définir l'orientation des interacteurs. Cette situation interactionnelle renforce donc, comme décrit au chapitre 3, l'intérêt pour le système de connaître les relations spatiales entre les entités physiques, et notamment entre les entités servant de surface et l'acteur naturel.

Pour les IHM rotatives, la matrice de projection d'un interacteur prend la forme suivante : [cos(thêta), sin(thêta), 0][-sin(thêta), cos(thêta),0][0,0,1] où thêta définit l'angle de rotation en radian. Notre prototype de table GloSS [Coutaz et al 2002] utilise cette fonction de projection afin de garantir au mieux l'adéquation de l'interface graphique avec les caractéristiques de la surface choisie : une table ronde. La figure 39 montre deux utilisateurs interagissant sur notre prototype de table augmentée. Le contour de la table définit une zone sensible (la partie blanche) qui permet à l'utilisateur de spécifier la rotation à appliquer à toute l'IHM.

Figure 39. Interface rotative du prototype table Gloss. En cliquant devant lui sur la zone blanche située autour de la table l'utilisateur peut orienter l'interface en face de lui

L'inconvénient de la rotation « tout ou rien » est que l'affichage ne peut convenir à tous les utilisateurs simultanément si ceux-ci sont répartis autour de la table. DiamondSpin [Shen et al 2004] offre plus de souplesse puisque cet outil permet de définir une orientation pour chaque interacteur. La figure 40 illustre un système de visualisation d'images réalisé avec DiamondSpin où les interacteurs-photos sont disposés en cercle.

Figure 40. Interface rotative du système DiamondSpin.

Toutefois, la surface utilisée doit être placée à la verticale du projecteur vidéo : le paramètre de glissement (« shear ») de la fonction de projection est une constante définie à 1. En conséquence, sans ce positionnement précis, la projection ne conserve pas la forme des interacteurs (exemple : un rectangle devient un parallélogramme). En somme, l'utilisation opportuniste de n'importe quelle surface de l'environnement requiert la prise en compte de tous les paramètres de la fonction de projection (zoom, rotation et glissement) comme on le pratique dans les IHM avancées.

V.2.4. IHM AVANCÉES

Le facteur de glissement permet de réaliser une projection perspective qui garantit une IHM non déformée sur une surface plane quelle que soit sa position par rapport à l'effecteur système.

Le EveryWhere Display Projector (EDP) [Pinharez 2001] et le Personal Display Surface (PDS) [Borkowski et al 2003] en illustrent l'intérêt. EDP est un projecteur vidéo disposé à la verticale sur lequel est fixé un miroir dans le prolongement de la zone de projection (voir figure 41). En modifiant l'orientation du miroir, le lieu de la projection est modifié. Par ce dispositif, n'importe quelle entité physique peut jouer le rôle de surface sous réserve qu'elle soit atteignable via le miroir. Pour arriver à cette utilisation opportuniste et pour que la projection soit correcte d'un point de vue humain, le système passe par une phase de calibrage. Le calibrage permet de modéliser l'environnement comme un ensemble de surfaces et,pour chacune d'elle, d'en calculer la matrice de projection.

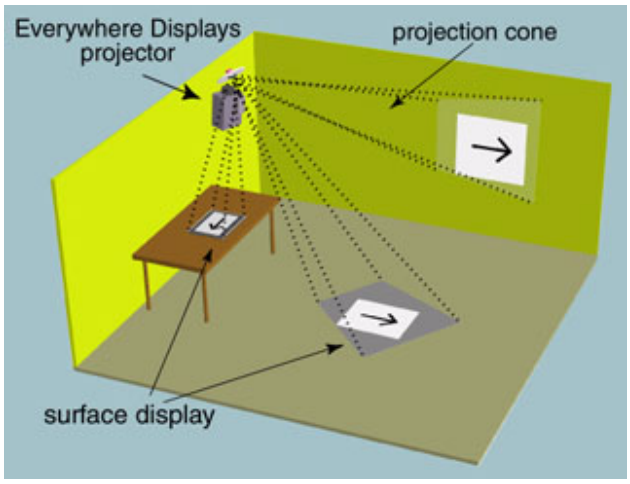

Figure 41. Principe de fonctionnement du everywhere display projector.

Dans le cas de PDS, le projecteur vidéo ne dispose pas de miroir, mais est monté sur une plate-forme motorisée rotative. Une caméra mobile permet de suivre en temps réel la localisation de la surface de projection par rapport au système de référence du projecteur et de là, de calculer la fonction de projection en sorte qu'il n'y ait pas de déformation (par exemple, la figure 42 montre l'utilisateur tenant une feuille cartonnée). Cette notion de surface personnelle mobile a été introduite par Szalavári avec le PIP (Personal Interaction Panel) [Szalavári et Gervautz 1997]. Ici, la projection se voit à travers un casque qui donne l'illusion que l'espace numérique projeté repose sur le PIP. En changeant l'orientation du PIP, le système change le contenu informationnel projeté. Le PIP sert donc à la fois de surface et d'instrument.

Figure 42. Projection d'une interface sur une surface déplaçable par un utilisateur avec le système Personal Display Surface.

Parce que le PIP, l'EDP et le PDS n'utilisent qu'un seul effecteur, la projection n'est possible, à un instant donné, que sur une seule surface à la fois. La création d'IHM multisurface nécessite l'utilisation conjointe de plusieurs effecteurs et la nécessité de mettre en correspondance plusieurs fonctions de projection. La section suivante développe ces éléments.

V.3. Projection d'espaces numériques sur plusieurs surfaces

Au chapitre 4, nous avons montré les opportunités interactionnelles offertes par le couplage de surfaces. Si plusieurs surfaces sont en relation de couplage, il convient alors de s'interroger sur la distribution des interacteurs constitutifs de l'espace numérique, mais aussi sur leur migration entre surfaces. Pour une IHM, « être distribuée » signifie que sa projection a lieu sur plusieurs surfaces simultanément, ces surfaces étant gérées par des calculateurs distincts. Pour une IHM, « migrer », c'est changer de surface, les surfaces sources et cibles étant gérées par des calculateurs distincts. Analysons quelques exemples de l'état de l'art.

La métaphore du peintre de Rekimoto [Rekimoto et Saitoh 1999] s'appuie sur une répartition de l'IHM entre deux surfaces. Cette répartition est guidée par les caractéristiques physiques des surfaces en question : la grande surface verticale fixe (un tableau de type smartboard) sert de zone de dessin alors que la petite surface mobile qui tient dans la main (un assistant personnel) est utilisée pour afficher les palettes d'outils. Cette distribution est statique. Une fois l'IHM répartie entre les surfaces, l'allocation des interacteurs aux surfaces ne peut être changée. On assiste à une IHM distribuée mais non migrable. À l'inverse, des systèmes comme PDS et EDP, déjà évoqués, autorisent la migration mais pas la distribution. Il s'agit d'IHM centralisées, mais migrables.

Dans cette section, nous analysons plus avant la nature de la distribution et de la migration. Nous utilisons pour cela le critère de granularité et son lien technique avec la fonction de projection, puis son effet ergonomique avec le problème de discontinuité visuelle.

V.3.1. NIVEAUX DE GRANULARITÉ

Nous distinguons cinq grains de migration et/ou de distribution : système d'exploitation, application, espace de travail, interacteur, pixel.

Niveau système d'exploitation ou totalité. La fonction de projection s'applique de la même façon à toute l'IHM. Par exemple, quand une fenêtre migre d'une surface A vers une surface B alors toutes les autres

fenêtres visibles sur A suivent ce mouvement. Ce cas est notamment illustré par les systèmes fondés sur VNC comme PointRight [Johanson et al 2002a] et MightyMouse [Kellogg 2002]. VNC ne permettant qu'une réplication de l'interface graphique, quand cette interface est projetée sur une autre surface, la migration est totale. La totalité peut aussi s'exprimer sur les autres paramètres de la fonction de projection. Par exemple, avec notre table GloSS, la valeur de la rotation s'applique à tous les interacteurs. Autrement dit, c'est toute l'interface graphique qui tourne quand l'utilisateur applique une rotation.

Niveau application. Chaque application dispose de sa propre fonction de projection. C'est le cas d'EasyLiving [Brumitt et Shafer 2000] qui permet à l'application « courrier électronique » de suivre les déplacements de l'utilisateur et s'afficher sur le mur le plus proche, alors que l'application « musique de fond » reste sédentaire.

Niveau espace de travail. Chaque espace de travail dispose de sa fonction de projection. Un espace de travail est un interacteur de type « container » regroupant un ensemble d'interacteurs qui permettent à l'utilisateur de réaliser un ensemble de tâches logiquement connectées [Normand 1992]. Par exemple, avec Exposé [Apple] (le nouveau système de gestion du bureau d'Apple), les fenêtres peuvent, sur pression d'une touche du clavier, se réfugier sous les bords de l'écran rendant les icônes du bureau visibles. Cet effet visuel se traduit par l'application d'un facteur de translation différent pour chaque espace de travail (des fenêtres dans ce cas). L'incrément progressif du paramètre permet de définir des interfaces de transitions entre deux états de l'interface.

Niveau concept du domaine ou niveau interacteur. Chaque interacteur a sa propre fonction de projection. Ainsi, dans l'exemple de DiamondSpin, chaque interacteur peut être tourné et agrandi indépendamment des autres.

Niveau pixel. Une migration au niveau pixel signifie que l'IHM peut bouger d'un pixel. Une distribution au niveau pixel signifie que deux pixels constitutifs d'un interacteur peuvent être situés sur deux surfaces distinctes. Le niveau pixel offre une grande souplesse qui n'est pas sans poser quelques difficultés. Nous les évoquons ci-dessous.

V.3.2. CONSTRUCTION D'UN ESPACE UNIFIÉ

Lorsqu'un interacteur est rendu à cheval sur plusieurs surfaces, ses fonctions de projection doivent être mises en relation afin de créer l'illusion d'un espace unifié. À titre ilustratif, prenons l'exemple du Dynawall [Streitz et al 1999]. Le Dynawall définit une surface interactive de la taille d'un mur par composition de trois Smartboard identiques juxtaposés verticalement [SmartBoard]. Chaque SmartBoard est relié au niveau des entrées à une machine spécifique alors que l'affi-

chage est géré de manière centralisée par la carte graphique de l'une des trois machines.

La surface située la plus à gauche joue le rôle de surface de référence : les fonctions de projection sont construites par rapport à la position de cette surface dans l'alignement. Pour la surface de référence, la matrice de projection est la suivante : [zx, 1, 0][1, zy, 0][0, 0, 1]. Alors, pour la surface située à sa droite, la matrice de projection devient [zx, 1, tx][1, zy, 0][0, 0, 1] où tx exprime la translation qu'il faut appliquer sur l'axe des x afin de tenir compte de la longueur du tableau de référence. Ce nombre est négatif car l'interface doit être translatée vers la gauche. De même, pour la surface située à l'extrême droite il faut appliquer une translation deux fois plus grande avec la matrice de projection : [zx, 1, 2*tx][1, zy, 0][0, 0, 1]. Cette mise en correspondance des matrices permet au système de donner l'illusion à l'utilisateur que les trois tableaux ne forment qu'un.

En y regardant de plus près, on constate que tx ne tient compte de la taille du cadre physique du tableau dans lequel est serti la surface inscriptible proprement dite. De plus, les tableaux ont la même résolution d'affichage et la même taille ce qui définit une composition homogène. L'hétérogénéité inhérente aux espaces interactifs appelle les remarques suivantes :

A) En ne modifiant la matrice de référence que d'une seule translation, on ne tient pas compte de la taille physique des surfaces. Si les surfaces ne sont pas homogènes, il faut ajouter un facteur d'échelle afin d'éviter la discontinuité visuelle visible sur la figure 43. Ce phénomène, très courant sur les ordinateurs équipés d'une carte graphique multi-écrans, est généralement limité par le fait que les utilisateurs évitent de positionner des interacteurs à cheval et préfèrent disposer leurs espaces de travail sur une surface ou sur une autre [Tan et Czerwinski 2003].

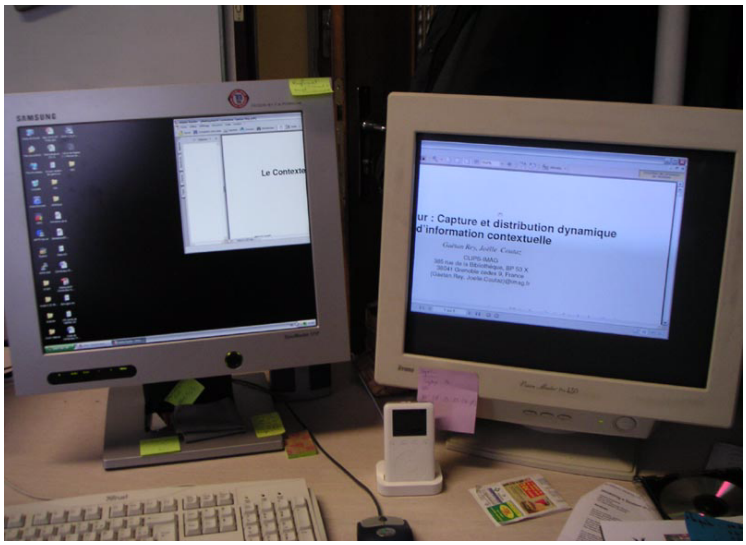

Figure 43. Interface distribuée entre deux surfaces gérées par la même machine. Cette figure illustre le problème de la discontinuité visuelle.

B) La translation ne tient pas compte de la taille physique des bords des surfaces sur lesquels l'affichage n'est pas toujours possible. Ce problème, très apparant en deux dimensions, crée un phénomène de discontinuité visuelle[Mackinlay et Heer 2004][Hinckley et al 2004][Baudish et al 2004].

C) Lorsque les surfaces sont positionnées dans une configuration non classique, comme dans les ConnecTables [Tandler et al 2001], bord haut contre bord haut, il faut appliquer une rotation à l'interface.

D) L'utilisation de projecteur vidéo nécessite l'utilisation du paramètre de glissement afin de corriger les déformations éventuelles. Dans ce cas, la construction de la matrice est une opération complexe.

Tous ces éléments font apparaître clairement un risque de discontinuité visuelle que seuls une connaissance des attributs physiques des surfaces, une utilisation adéquate de tous les paramètres de la matrice et une mise en relation des fonctions de projection permettent de résoudre. Ainsi, nous ajoutons au diagramme de la figure 35 une association entre deux fonctions de projection qui traduit leur mise en relation. Comme le couplage de deux surfaces peut justifier cette mise en relation, celle-ci devient un attribut de la classe Couplage.

V.3.3. DISCONTINUITÉ VISUELLE

De précédents travaux en psychologie cognitive ont montré qu'il était raisonnable de penser que la performance de l'homme pour la réalisation d'une tâche pouvait être influencée par la configuration des surfaces (on parlera de topologie) ainsi que de leurs bords [Campbell et Maglio 2003], [Tan et al 2003]. Dans [Tan et Czerwinski 2003], Tan et

al rapportent une étude sur les effets de la séparation visuelle et des discontinuités physiques lorsque le contenu numérique relié à la même tâche est distribué sur plusieurs écrans. Leur expérimentation montre que les discontinuités physiques introduites par les bords ou une différence de profondeur seule, ne semblent pas avoir un effet significatif sur la performance des sujets pour la comparaison de texte. Cependant, dans leur expérimentation, les fenêtres sont affichées sur un seul écran à un moment donné. Comme nous allons le montrer ci-dessous, la projection sur plusieurs surfaces produit différents résultats selon que les bords sont ignorés ou pris en compte.

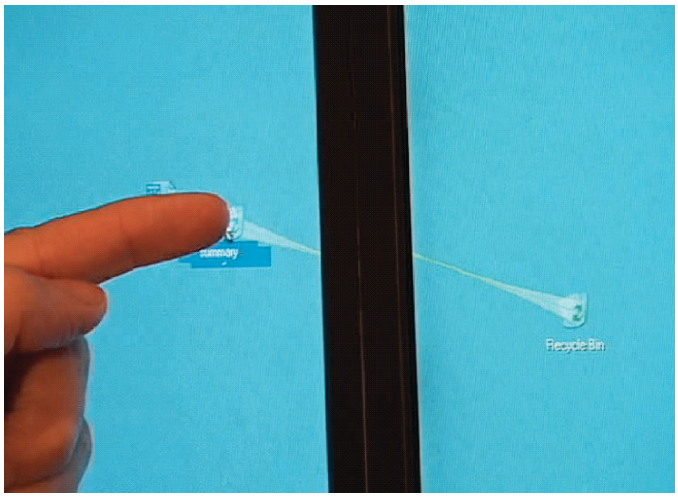

Figure 44. Illustration de discontinuité visuelle lorsque les bords ne sont pas pris en compte.

L'exemple de la figure 43 illustre l'effet visuel lorsque la fonction de projection ignore l'existence des bords. La technique d'interaction utilisée, dite du «Drag & Pop», [Baudisch et al 2003] vise à réduire la trajectoire d'interaction. Cette image montre un trait qui relie deux «icônes», chacune située sur un tableau différent. Du fait des bords, le trait, qui matérialise l'équivalence des deux icônes, semble cassé. En effet, la présence des bords a ajouté une translation sur l'axe des x à la fonction de projection déjà effectuée par la carte graphique.

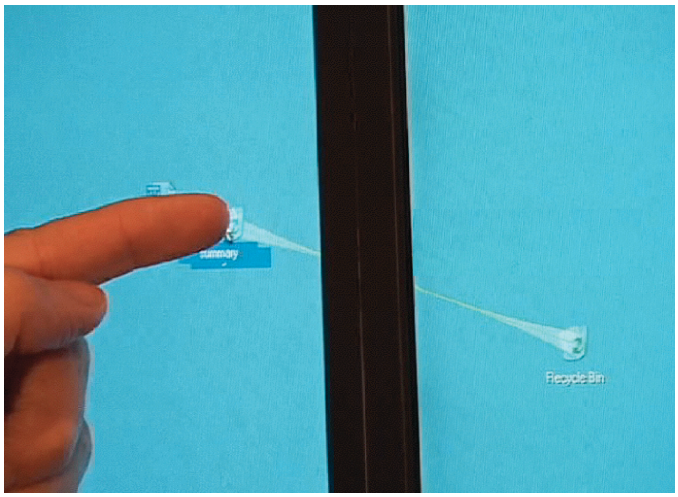

Figure 45. Prise en compte de la taille des bords lors de la projection d'un même interacteur sur plusieurs surfaces.

Sur la figure 45, nous avons modifié l'image afin d'enlever l'effet visuel produit par les bords. Nous avons ainsi exprimé une mise en relation des fonctions de projection de l'interacteur qui permet de rendre le trait "continu". Cependant, cette opération, qui vise à préserver la vision d'un espace unifié, n'est pas toujours possible. Prenons un exemple. La figure 46 montre un espace physique composé de trois surfaces mises côte à côte en deux dimensions pour rendre visible un espace numérique contenant un graphe constitué de trois noeuds A, B et C tous reliés. La ligne du haut de la figure montre le résultat final de la projection perçu par l'utilisateur pour trois fonctions de projection différentes. La ligne du bas montre comment l'espace numérique est projeté sur l'espace physique dans chacun des cas. La zone rendue visible par chacune des surfaces est représentée en pointillés.

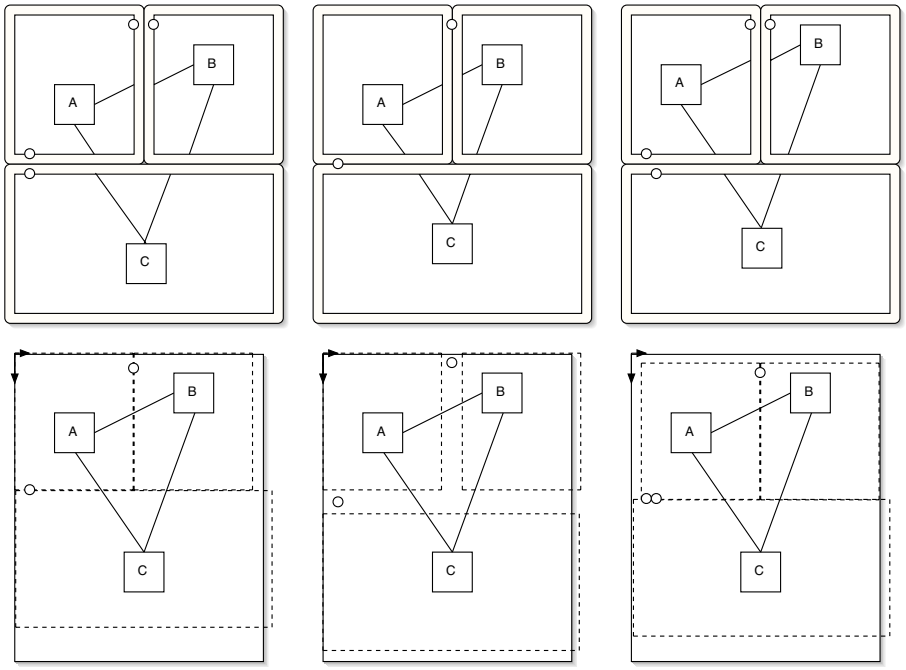

Figure 46. Exemples d'interfaces distribuées sur trois surfaces illustrant le problème de discontinuité visuelle dûe aux bords des surfaces.

Sur le cas de gauche, les bords sont ignorés. Le résultat ressemble à la vision d'un miroir cassé comme sur la figure 44. Sur le cas du milieu, les bords sont pris en compte et sont traités comme une zone opaque. Le graphe semble correct mais les pixels qui tombent sous les bords sont perdus. Toute l'information n'est donc pas visible. Enfin sur le dernier cas, on note un effort pour limiter la perte de pixels tout en préservant la sémantique de la figure. Cependant, ce cas semble difficile à automatiser. Y a-t-il des pixels plus importants que d'autres qu'il ne faudrait pas perdre ? Si c'est le cas, comment le savoir et comment réagir ?

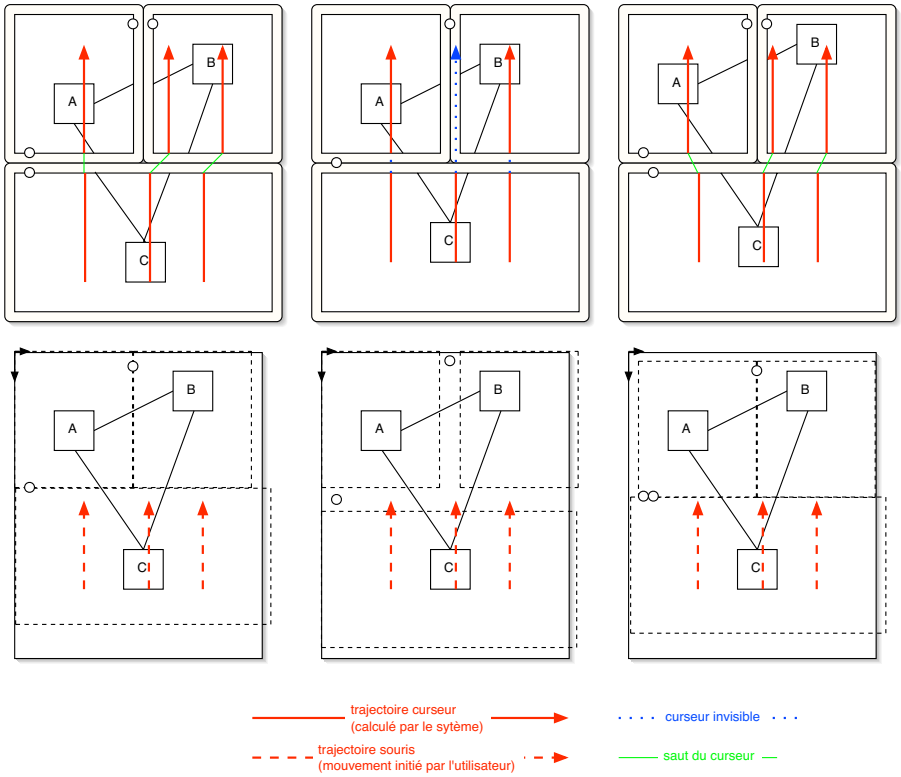

Figure 47. Trajectoire du curseur souris dans une interface graphique rendue sur trois surfaces.

La figure 47, illustre, quant à elle, le même problème mais au niveau d'un interacteur particulier : le curseur de la souris. Ici, l'utilisateur est supposé être assis en face de cette configuration de surfaces. Il déplace le curseur de la souris verticalement en partant de la surface du bas et en allant vers les surfaces du haut. Les flèches en pointillés de la ligne du bas décrivent la trajectoire décrite par la souris. Quant aux flèches pleines du haut, elles représentent la trajectoire suivie par le curseur asservi aux déplacements de la souris. La figure reprend les trois fonctions de projection de l'exemple précédent.

Nous constatons qu'il est très difficile de maintenir la même trajectoire dans l'espace physique et dans l'espace numérique excepté pour le cas central où les bords sont considérés comme des zones opaques. Cependant, avec cette politique, le curseur peut disparaître quand il franchit la frontière définie par les bords des écrans. Quelle solution doit-on privilégier? La question reste ouverte.

Le problème de la discontinuité visuelle met en évidence les différences de nature entre les espaces physique et numérique. Nous avions déjà évoqué cette différence au chapitre 3 à propos de la topologie des surfaces de iRoom [Johanson et al 2002b]. En effet, même s'il existe

un espace d'un mètre entre deux surfaces physiques, le système peut choisir de l'ignorer, tout comme il peut ignorer les bords.

Il est donc utile de distinguer l'espace physique de l'espace logique. D'ailleurs, pourquoi ne pas supposer que, chaque application, voire chaque interacteur, puisse choisir son monde logique. Cependant, il faut faire attention : cette liberté peut être source de problèmes car l'espace interactif global, composition des IHM des applications, risque d'être incompréhensible et donc inutilisable. Afin de limiter ces risques, le bons sens suggère de calquer la forme de l'espace logique sur celle de l'espace physique ou, si cela est impossible, de s'en approcher.

V.4. Synthèse

La vision de Weiser, « le monde comme espace d'interaction », nécessite une modélisation poussée du monde physique non traitée dans le logiciel de base pour l'Interaction Homme-Machine. Cette connaissance, alliée au potentiel de communication et d'affichage des calculateurs, permet d'envisager la création d'interfaces graphiques migrables et distribuables. Ces interfaces, mieux adaptées au contexte de l'informatique diffuse que les interfaces conventionnelles, reposent, comme ces dernières, sur le concept de fonction de projection. Mais l'hétérogénéité tout comme les caractéristiques physiques des surfaces (existence de bords) et leur topologie posent de nouvelles questions tant sur le plan technique que sur le plan ergonomique. Notre ontologie a permis d'exprimer ces problèmes de manière rationnelle. Au chapitre suivant, nous les traduisons en termes de requis logiciels et analysons l'offre de l'état de l'art au regard de ces requis. Cette analyse nous permettra ensuite de présenter notre solution : I-AM.

Chapitre VI *Requis logiciels et analyse de l'état de l'art*

Jusqu'ici, nous avons présenté les éléments d'une ontologie conçue pour comprendre les faits saillants de l'interaction en informatique ambiante : c'est l'ancrage dans le monde physique, ce sont les relations spatiales entre les entités de ce monde, leurs rôles de surface et d'instrument, leur couplage, et la mise en correspondance des espaces numériques et physiques. Dans ce chapitre, nous en analysons les conséquences du point de vue du Génie Logiciel. Dans la section qui suit, nous exprimons ces conséquences sous forme de requis logiciels. Ces requis servent ensuite à analyser l'offre de l'état de l'art de manière systématique.

Reprenant nos notions centrales de surface et d'instrument, nous avons choisi de répartir les solutions de l'état de l'art en quatre catégories :

- A. les systèmes monosurface et mono-instrument,
- B. les systèmes monosurface et multi-instrument,
- C. les systèmes multisurface et mono-instrument,
- D. les systèmes multisurface et multi-instrument.

Cette classification complète le modèle analytique UDP/C de Lecolinet [Lecolinet 2003a]. UDP/C (User, Display, Pointer/Computer »), vise une caractérisation synthétique des systèmes multipointeur. Pour un système donné, UDP/C exprime la cardinalité de chacun des éléments du quadruplet <U, D, P, C>. Par exemple, l'expression <1-1-N>/2 dénote un système mono-utilisateur, doté d'un seul écran et d'un nombre quelconque de pointeurs, deux calculateurs gérant le tout.

Pour chacune des quatre catégories (A, B, C, D ci-dessus) de système, nous définissons l'expression UDP/C correspondante et nous proposons un tableau récapitulatif des systèmes représentatifs de la catégo-

rie. Chaque système est caractérisé par sa capacité à répondre à nos requis au moyen d'une note relative :

- (-) signifie que le requis n'est pas assuré,
- (+) montre que le système prend en compte le requis, mais de manière partielle,
- (++) indique que le requis est assuré,
- une case noircie dénote un requis non pertinent.

Ce barème approximatif sert d'indicateur de tendance, les publications ne permettant pas toujours d'extraire l'exacte couverture d'un système.

Une brève synthèse nous permet, en troisième partie, d'introduire I-AM, notre contribution technique présentée au chapitre suivant.

VI.1. Requis logiciels

En matière de Génie Logiciel, les facteurs et critères de McCall [McCall 1977] et de l'AFNOR servent de référence générale. Pour le cas précis qui nous intéresse, nous avons retenu huit requis, d'égale importance, répartis comme suit :

- Des requis fonctionnels fondés sur les éléments de notre ontologie : découverte des acteurs et ressources d'interaction, localisation d'entités physiques, modélisation du couplage, migration et distribution d'IHM ;
- Des requis non fonctionnels fondés sur des principes de l'Interaction Homme-Machine et du développement de système : architecture distribuée, latence, prise en compte de l'existant (legacy system) et facilité de programmation.

Nous aurions pu, dans le contexte de l'informatique ambiante, considérer d'autres requis comme le passage à l'échelle, la sécurité, la protection de la sphère privée et l'éthique. Ces sujets constituent à eux seuls des sujets de recherche, véritables verrous au déploiement effectif de l'informatique ambiante. Ils ne seront pas traités dans le cadre de cette thèse. Dans les sections qui suivent, nous passons successivement en revue les requis retenus.

VI.1.1. DÉCOUVERTE DES ACTEURS ET RESSOURCES D'INTERACTION

Le monde physique est, par essence, dynamique et peuplé, on l'a vu, d'entités physiques parmi lesquelles, les acteurs naturels et les ressources d'interaction. Le logiciel de base qui sous-tend la mise en œuvre d'espaces interactifs, doit donc être capable de découvrir les entités pertinentes.

Découvrir inclut trois fonctions :

- 1. *Reconnaître*, parmi les entités du monde, les acteurs naturels et les ressources d'interaction. Pour cela, le système doit être capable d'observer et de modéliser les attributs et les propriétés des entités pertinentes. C'est par l'analyse des attributs observés et des propriétés calculées que le système peut décider de la présence d'acteurs naturels et d'entités pouvant jouer (ou jouant) le rôle de ressource d'interaction. Ou encore, c'est en connaissant les attributs d'une surface que le système peut répondre à une requête utilisateur où il est fait référence aux attributs d'une entité, par exemple, « migrer l'IHM vers la plus grande surface verticale ».
- 2. *Identifier* chaque exemplaire d'acteurs et de ressources d'interaction de façon à les distinguer dans le processus d'interaction. Par exemple, un système utilisant plusieurs caméras, comme EasyLiving, [Brumitt et al 2002] et observant une même personne, doit fusionner l'information en provenance de plusieurs sources pour comprendre qu'il s'agit de la même personne. Ou encore, plusieurs machines observant la même entité doivent comprendre qu'il s'agit de la même entité. Un espace de noms est donc nécessaire.
- 3. *Détecter* voire suivre la présence, l'arrivée et le départ des acteurs naturels et des ressources d'interaction : le monde est dynamique et façonnable sous le contrôle de l'utilisateur. Ces détections de phénomènes imprévisibles se modélisent classiquement sous forme d'événements. La création d'événements reflétant l'évolution de l'espace interactif et la possibilité donnée aux programmeurs de s'y abonner vise à réduire la complexité du monde physique. En outre, il est envisageable de structurer ces événements en différents niveaux d'abstraction en fonction du type de données.

La découverte de ressources d'interaction s'appuie sur une infrastructure réseau et ses protocoles tels que Zéroconf [ZeroConf] et UPnP [UPnP]. Précisons toutefois que ces services de découverte dénotent la capacité de communiquer, mais ne signifient pas nécessairement la présence physique de l'entité dans l'espace interactif. Des informations topologiques (relations spatiales entre entités physiques) sont nécessaires pour compléter les services de découverte réseau et déterminer la présence tangible ou l'absence d'une entité. Ces informations sont fournies par les services de localisation.

VI.1.2. LOCALISATION D'ENTITÉS PHYSIQUES

Un service de localisation, on l'a vu au chapitre 3, couvre quatre niveaux d'abstraction successifs :

- 1. Le niveau de localisation discrète et identification où les entités physiques sont positionnées dans le repère des capteurs.
- 2. Le niveau 2 de localisation métrique et géométrique où les entités

sont toutes repérées dans un espace de coordonnées absolues.

- 3. La localisation structurelle et relationnelle où les entités physiques sont liées sur la base de leurs positions relatives telles que “à droite de”, “à l’intérieur de”.
- 4. La localisation sémantique où le repérage d’une entité s’effectue dans un espace symbolique dépendant de l’application, par exemple à Grenoble.

À l’évidence, tous les niveaux d’abstraction sont souhaitables, mais les niveaux 1, 2 et 3, (localisation discrète et identification, localisation métrique et géométrique, localisation structurelle) sont nécessaires. En outre,

- Pour les niveaux 1 et 2, le requis de précision s’impose. Sans localisation géométrique précise, la distribution d’IHM au niveau pixel sur plusieurs surfaces risque d’entraîner des discontinuités visuelles inacceptables pour l’utilisateur ;
- Le niveau 3 intervient dans l’assemblage de l’espace d’interaction (par opération de couplage par exemple), mais aussi sert à adapter la présentation des informations (par exemple, connaissant la position relative de l’utilisateur et des surfaces d’interaction, le système peut afficher l’information face à lui, non pas derrière lui ou à l’envers),
- Tous les niveaux doivent fonctionner en permanence de façon à assurer le suivi dynamique des entités mobiles. Se pose alors le requis de résolution spatio-temporelle : de combien d’unités spatiales l’entité doit-elle se déplacer pour que son mouvement soit détecté et mesuré par le système de localisation ? Quelle est la fréquence des mesures du déplacement ? Il n’y a pas de réponse unique à ces questions si ce n’est qu’elle dépend des techniques d’interaction envisagées.

VI.1.3. MODÉLISATION DU COUPLAGE

Au chapitre 4, nous avons montré que le couplage était mal formalisé, mais incontournable dans l’assemblage d’espaces interactifs.

Reprenant à notre compte les travaux de Barralon [Barralon et al 2004], nous imposons comme requis :

- La modélisation explicite du couplage de ressources d’interaction, à raison d’un automate par couplage. En particulier, nous mettons l’accent sur la nécessité de modéliser les couplages instrument-instrument, instrument-surface et surface-surface,
- L’expression explicite auprès de l’utilisateur de l’état courant de chaque automate en sorte que les propriétés de robustesse et de sou-

plesse de l'interaction soient satisfaites.

VI.1.4. Migration et distribution d'IHM

La migration et la distribution d'IHM impose des requis sur les fonctions de projection des espaces numériques, sur les boîtes à outil, sur les pointeurs.

Fonctions de projection d'espace numérique sur l'espace physique.

Ces fonctions doivent :

- Tenir compte des attributs des surfaces et des relations spatiales entre l'effecteur de la projection et les entités jouant le rôle de surface ;
- Être spécifiques à chaque espace numérique en sorte que chaque espace numérique puisse exploiter l'espace physique en fonction des besoins du domaine d'application.

Boîtes à outil graphiques.

Afin d'assurer la plus grande souplesse, il convient qu'une boîte à outil graphique dite nouvelle génération assure la distribution, la migration, la rotation et le facteur d'échelle au niveau du pixel. À partir de là, tous les autres grains (interacteur, espace de travail, application, etc.) peuvent être assurés.

Gestion de plusieurs pointeurs.

Puisque les espaces interactifs sont multi-instrument incluant éventuellement plusieurs instruments de pointage, l'infrastructure doit être capable de gérer plusieurs pointeurs (curseurs) sur une même surface, de même leur migration. En effet, si les interacteurs peuvent migrer, il doit en aller de même pour les pointeurs auxquels les interacteurs sont liés.

VI.1.5. Architecture distribuée

Dans la vision de l'informatique ambiante, les utilisateurs peuvent créer des îlots d'interaction au-dessus d'un réseau ad hoc. L'image de droite de la figure 3 du chapitre 1 en donne un exemple : deux amis assemblent leur PDA pour former un espace privé indépendant de la planète. Ici, un service de couplage fondé sur l'existence d'un serveur distant, ne serait pas assuré.

En conséquence, tout service devant être fourni pour des îlots d'interaction (et notamment les services de couplage, de projection, déjà cités), doit adhérer au modèle d'architecture paire à paire ("peer-to-peer"). Aucune machine ne jouant de rôle privilégié, on obtient alors une souplesse logicielle en conformité avec la créativité spontanée des humains.

VI.1.6. Conformité de la latence

Le temps de latence, ou plus simplement latence, d'un acteur est le temps écoulé entre la présentation d'un stimulus à cet acteur (par exemple l'acquisition d'un événement par un acteur artificiel) et le début de la réponse correspondante (par exemple, la mise en grisé

d'une zone de l'écran). La conformité de la latence de l'acteur artificiel à celle de l'acteur naturel est l'un des requis fondamentaux de l'IHM (cf. le fameux principe du « retour d'information immédiat » de Schneiderman [Schneiderman 87])

Dans ses travaux sur l'Interaction Fortement Couplée (IFC), Bérard [Bérard 1999] montre que la latence du système doit être de l'ordre de 50ms. Le déplacement de la souris vers une cible est un exemple d'IFC. Plus formellement, « une interaction est fortement couplée sur un intervalle de temps donné lorsque les acteurs humain et artificiel sont engagés de manière continue dans l'accomplissement d'actions physiques mutuellement perceptibles et dépendantes sur cet intervalle » ([Bérard 1999] p.9). Ces règles s'appliquent à l'interaction multisurface multi-instrument. On retiendra donc une latence système de 50 ms.

Lee et al décomposent la latence système [Lee et al] en :

- latence système interne : c'est la latence inhérente d'un dispositif ou d'un algorithme,
- latence externe : c'est la latence due à la communication entre les dispositifs et les ordinateurs (ex : souris sans fil) et entre les ordinateurs (ex : connexion réseau sans fil).

En architecture distribuée, une mesure rigoureuse de la latence est difficile à réaliser, amplifiée par le fait qu'il n'est pas toujours possible d'avoir accès à un procédé fiable de mesure de l'écoulement du temps. Par exemple, en Java, la précision de l'horloge système est de 10 ms ce qui entraîne une erreur de mesure de 20% pour une cible de 50ms. Néanmoins, il est toujours possible de tester la *latence perçue système* par des campagnes d'expérimentation avec des utilisateurs.

VI.1.7. RÉUTILISATION DE L'EXISTANT (LEGACY SYSTEMS)

La réalisation des IHM conventionnelles se solde aujourd'hui par la production de millions de lignes de code. Il convient donc de s'interroger sur l'opportunité, pour une infrastructure middleware multisurface multi-instrument, d'accueillir des applications patrimoniales.

Si l'on est tenté de répondre positivement à cette question pour des raisons de coût, il n'en va pas de même si l'on se place du point de vue utilisateur : ces logiciels ayant été conçus pour du mono-surface, mono-instrument, l'interaction en milieu multisurface, multi-instrument, pourrait porter à confusion. Par exemple, un seul clavier serait utilisable alors que plusieurs sont présents dans la salle.

Ce constat motive notre position : il n'est intéressant de réutiliser les applications existantes que si elles tirent pleinement profit des opportunités interactionnelles données par l'interaction multi-instrument et multisurface. Il s'agit donc de trouver un bon compromis entre le nom-

bre de modifications à effectuer sur les applications existantes et le nombre de nouveaux services offerts, sachant qu'il est souhaitable de minimiser le premier et de maximiser le second.

VI.1.8. FACILITÉ DE PROGRAMMATION

D'après Myers et ses co-auteurs, le succès d'un outil logiciel pour le développement d'IHM tiendrait en cinq points [Myers et al 2001a]:

- Les parties IHM que couvre l'outil sont en adéquation avec les besoins du développeur (l'outil fournit juste ce qu'il faut). Appliqué à notre cas, il s'agira de fournir les services identifiés par les éléments de notre ontologie (à supposer qu'elle constitue le bon fondement) et notamment masquer au programmeur l'hétérogénéité des plates-formes, leur arrivée et départ dynamique, le couplage, la projection des espaces numériques, la migration, la distribution, etc.
- Le rapport entre le seuil d'apprentissage acceptable pour maîtriser l'outil et le plafond fonctionnel du produit obtenu avec l'outil convient. Il semble que, jusqu'ici, les outils ayant rencontré le plus de succès relèvent du profil « seuil bas-plafond bas » (peu d'apprentissage, mais peu de fonctions produites automatiquement avec l'outil) ou « seuil haut-plafond haut » (lourd effort d'apprentissage, mais un riche niveau de production). Le défi des outils à « seuil bas-plafond haut » reste entier. Dans notre cas, une infrastructure pour interaction multi-instrument, multisurface, a nécessairement un plafond fonctionnel bas : c'est un outil à bas niveau d'abstraction. Il faudra donc viser un seuil d'apprentissage bas.
- Le chemin de moindre résistance : l'outil conduit les développeurs à faire les choses justes tout en prévenant les erreurs.
- Prévisibilité : les résultats obtenus avec l'outil sont ceux que l'on attendait (ce qui n'est pas toujours le cas avec les générateurs d'IHM, par exemple).
- Cibles mouvantes : disposer du bon outil au bon moment. C'est l'objet même de ces travaux de recherche.

Ayant présenté les requis d'une infrastructure multi-instrument multisurface pour informatique ambiante, nous sommes maintenant en mesure d'analyser l'offre de l'état de l'art.

VI.2. Analyse de l'état de l'art

Nous structurons l'analyse selon les quatre classes présentées en introduction de ce chapitre : les systèmes monosurface mono-instrument, les systèmes monosurface multi-instrument, puis les systèmes multisurface mono-instrument en finissant par le cas le plus général : les systèmes multisurface multi-instrument.

Dans le contexte de cette analyse,

- Un système est *multi-instrumen*t :

  - s'il permet l'utilisation de plusieurs couples « instruments de pointage, instrument de saisie de texte » (souris-clavier par exemple),

  - ou s'il autorise plusieurs instruments de pointage par surface.

- Un système est multisurface :

  s'il permet l'utilisation de plusieurs surfaces gérées par des ordinateurs différents. Plusieurs écrans gérés par un même ordinateur au moyen d'une carte graphique multi-écran n'est pas multisurface. Le concept de display utilisé par Lecolinet comme dans X Window suit la même définition.

VI.2.1. SYSTÈMES MONOSURFACE MONO-INSTRUMENT

Le système d'exploitation conventionnel qui équipe nos machines est mono-instrument monosurface car même s'il peut gérer plusieurs écrans leurs gestions se fait par l'intermédiaire d'une seule et même carte graphique. Suivant la classification UDP/C, un système comme celui-ci se note 1-1-1 car il n'est utilisable que par un utilisateur à la fois, il ne met en jeu qu'un display et un seul couple d'instrument de pointage et de saisie de texte.

L'adéquation des systèmes de cette catégorie par rapport aux requis se définit de la manière suivante :

- *points forts* : une découverte des ressources d'interaction (limitée aux connexions cablées comme USB), une faible latence et une facilité de programmation appéciable car le développeur a à sa disposition des boites à outils éprouvées.
- points faibles : aucune localisation automatique des entités physiques (il existe une localisation des écrans mais c'est l'utilisateur qu'il exprime) et une modélisation très flou du couplage.

Ici, le requis de réutilisation de l'existant n'a aucun sens puisque toutes les applications ont été conçues pour ce type de systèmes et sont donc directement utilisables. De même, les requis d'architecture distribuée et d'interface migrable et distribuable sont ici non pertinent.

L'adéquation de ce type de système par rapport aux requis est synthétisée sur le tableau récapitulatif de la figure 48 ci-dessous. Nous rappellons que le barême, expliqué au début du chapitre, est donné à titre indicatif.

Systèmes mono-instrument monosurface Notation UDP/C : 1-1-1	Découverte des acteurs et des ressources d'interaction	Localisation des entités physiques	Modélisation du couplage	Interface migrable et distribuable	Architecture distribuée	Conformité de la latence	Réutilisation de l'existant	Facilité de programmation
Système d'exploitation usuel	+	-	+			++		++

Figure 48. Classification des systèmes mono-instrument monosurface.

Les trois sous-catégories suivantes définissent les cas où plus d'une instance de ressource d'interaction est utilisable à un intant t. Dans cette situation, il est indispensable de pouvoir découvrir dynamiquement l'arrivée et le départ des ressources d'interaction, leurs configurations spatiales et leurs couplages.

VI.2.2. SYSTÈMES MONOSURFACE MULTI-INSTRUMENT

Classées dans cette catégorie de systèmes, on trouve les solutions logicielles suivantes :

- MMM (Multi-Device Multi-User Multi-Editor) [Bier et Freeman 1991],
- MID (Multiple Input Devices) [Hourcade et Bederson 1999],
- Pebbles [Myers et al 1998],
- Dynamo [Izadi et al 2003],
- DiamondSpin [Shen et al 2004],
- Table Magique [Bérard 2003],
- Group Pointer Interaction [Vogt et al 2004] et d'autres encore.

De manière générale, on appelle ce type de systèmes un "Single Display Groupware" (SDG) [Stewart et al 1999]. Un SDG est un système qui permet à plusieurs utilisateurs de travailler ensemble sur une même surface partagée en utilisant chacun un instrument de pointage particulier. Cet instrument de pointage peut être une souris (MMM, MID, Dynamo), un assistant personnel (Pebbles), un stylet (DiamondSpin), un jeton en plastique (Table Magique) ou encore un pointeur laser (Group Pointer Interaction).

Par rapport à la classification UDP/C, cette catégorie de systèmes se note N-1-N car plusieurs utilisateurs, un seul display et plusieurs pointeurs sont mis en jeu. Si plusieurs machines sont nécessaires à la réalisation du système, comme dans le cas de Peebles (plusieurs assistants personnels), la notation devient N-1-N/*.

Découverte des acteurs et des ressources d'interaction

Les systèmes Dynamo, Table Magique et Group Pointer Interaction sont les seuls systèmes de cette catégorie à découvrir dynamiquement les instruments utilisables. Les autres systèmes n'ont pas cette faculté et ont besoin d'une intervention utilisateur pour configurer l'arrivée ou le départ d'un ou plusieurs instruments. Les trois systèmes cités sont ainsi en adéquation avec le requis de découverte dynamique des ressources d'interaction. Cette adéquation provient des technologies utilisées (USB pour Dynamo et vision par ordinateur pour les deux autres).

Localisation des entités physiques

La prise en compte des relations spatiales entre les ressources d'interaction, et plus précisément ici la localisation des instruments, est le point faible des systèmes cités. En effet, seule la Table Magique localise en deux dimensions les instruments (des jetons en plastique) sur la table en temps réel. A l'inverse, chez les autres systèmes, ces informations spatiales sont ignorées. Elles ne peuvent donc pas servir à la définition du couplage des instruments.

Modélisation du couplage

Ces systèmes mettent en avant l'instrument de pointage par rapport à l'intrument de saisie de texte : en effet, aucun de ces systèmes, sauf Dynamo, ne gère l'instrument de saisie de texte. Ici, on ne peut donc pas parler de couplage de type clavier-souris. Si Dynamo modélise ce couplage, il est indispensable que le clavier et la souris soient connectés au même point d'interaction (interaction Point) ; c'est-à-dire qu'ils fassent partie d'un même regroupement de dispositifs. On parle de regroupement si les instruments sont rattachés au même concentrateur USB (HUB) ou à la même machine. Toutefois, le principe du point d'interaction a une limite : un instrument ne peut pas être rattaché n'importe où car Dynamo définit chaque point d'interaction comme appartenant à un utilisateur. Ainsi, si un utilisateur veut utiliser le clavier de son voisin (changer le couplage), il doit débrancher le clavier du concentrateur de son voisin et le connecter au sien. La prise en compte du couplage clavier-souris est dynamique mais ne peut être modifié que par l'utilisateur et non par le système. En outre, ce couplage n'est pas directement observable. Auncun rendu de cette information de couplage n'est produit sur l'interface. Seules les connexions filaires permettent de déduire les couplages existants. Ces manques vont à l'encontre des critéres de souplesse et de robustesse exprimé au chapitre 4.

Quant au couplage d'instruments de pointage, seuls les systèmes MMM et Table Magique le gèrent. Dans ces deux cas, le couplage vise à permettre une interaction à deux mains ou à plusieurs sur un même interacteur. Certes, avec les autres solutions, les interacteurs sont accessibles par tous les pointeurs mais aucune nouvelle fonctionnalité n'est offerte lorsque plusieurs utilisateurs agissent sur un même interacteur (rotation, étirement, suppression, etc).

Interface migrable et distribuable

Dans cette catégorie de système, il n'y a qu'une seule surface utilisable. Ainsi, la migration et la distribution de l'interface graphique ne sont pas des fonctionnalités pertinentes. Par contre, la prise en compte du facteur d'échelle et de la rotation des interacteurs peut être très utile surtout si la surface est située à l'horizontale. Parmi les systèmes cités, seuls la Table Magique et DiamondSpin gèrent et utilisent ces caractéristiques.

Le plus complet de ces deux systèmes est à l'heure actuelle DiamondSpin car la boîte à outils reprend les interacteurs de la boîte à outil graphique Swing de Java alors que la Table Magique utilise ses propres composants en cours de développement. Par contre, en ce qui concerne la qualité du rendu graphique, la Table Magique utilise openGL, ce qui lui garantit une faible latence de l'affichage lorsque le nombre d'utilisateurs et d'interacteurs augmente.

Architecture distribuée

Les systèmes multi-instrument monosurface que nous avons cités permettent l'utilisation d'un nombre important, mais limité, d'instruments (4 pour Diamondspin, 256 pour Dynamo par exemple). Cette fonctionnalité est restreinte par l'utilisation d'un serveur qui centralise les événements produits par les instruments. Si ce serveur tombe en panne le système s'écroule alors que la majeure partie des autres composants pourraient continuer à fonctionner correctement. De plus, au regard de nos requis, quel que soit le lieu de l'interaction, il faut que l'utilisateur puisse créer un espace d'interaction unifié. Le choix d'un serveur est une bonne solution quand le lieu de l'interaction est une pièce équipée (Roomware) mais il ne l'est pas, on le rappelle, pour un utilisateur en situation de mobilité.

Conformité de la latence

Nous n'avons aucune donnée quantitative permettant de savoir si les systèmes présentés dans cette catégorie vérifient ce requis de faible latence. Nous avons simplement des mesures qualitatives tirées des différentes vidéos que nous avons pu avoir de ces systèmes. Ces vidéos permettent de penser que ces systèmes sont en adéquation avec ce requis ; hormis peut-être DiamondSpin lorsque le nombre d'interacteurs visibles devient important.

Réutilisation de l'existant

Pour afficher plusieurs pointeurs sur la même surface, il faut aujourd'hui utiliser une boîte à outil spécifique qui, en général, interdit la réutilisation des applications préexistantes. Parmi les systèmes que nous venons de présenter seul Dynamo permet de réutiliser une grande partie des applications fonctionnant sur le système d'exploitation Windows sans qu'aucune modification ne soit nécessaire.

Cependant, cet avantage doit être pondéré par le fait que Dynamo n'autorise l'utilisation collective de la même application que si les actions des utilisateurs ont lieu successivement. En particulier, les

applications ne peuvent distinguer les instruments car elles sont construites sur un modèle mono-instrument et ne sont pas modifiées par Dynamo pour tenir compte de ce fait. Ainsi, elles ne considèrent qu'un seul instrument logique qui existe par fusion des événements de tous les instruments présents. A contrario, lorsque les actions ont lieu entre les applications (c'est-à-dire au niveau du bureau), des actions en parallèle sont permises car Dynamo gère un identifiant unique pour chaque instrument.

Contrairement à Dynamo, MMM autorise des actions en parallèle sur tous les composants de l'interface car les instruments sont continuellement différenciés. Ainsi, les niveaux de partage tolérés par ces systèmes ne sont pas identiques. MMM est plus souple que Dynamo en terme d'utilisation ; mais à l'inverse de ce dernier nécessite le développement d'applications dédiées.

DiamondSpin réutilise la boîte à outil Swing de Java et encapsule les interacteurs de cette API dans des objets dédiés utilisable par l'API de DiamondSpin. Ainsi, il est peut être envisageable de transformer automatiquement une application construite sur l'API Java en une application compatible DiamondSpin. Cependant, ce n'est pas le cas actuellement. Le programmeur s'il veut utiliser une application Java sur DiamondSpin doit effectuer lui-même les modifications nécessaires.

Facilité de programmation

Afin de garantir une programmation simple, ces systèmes introduisent le champ "identifiant de l'instrument" au sein de la structure des événements souris (c'est le cas pour tous les systèmes cités) et des événements clavier (seulement Dynamo gère cette fonctionnalité). Pour tirer profit d'un système multi-instrument monosurface, le programmeur doit utiliser ce champ afin de distinguer les différents instruments et par là même, si cela suffit, les différents utilisateurs. Cependant, la seule présence de cet identifiant ne garantit pas une gestion simple du couplage car ses systèmes ne formalisent pas celui-ci. Tout est à la charge du développeur.

Synthèse

Afin d'avoir une vision claire des requis respectés ou non par ces systèmes multi-instrument monosurface, nous récapitulons l'adéquation aux requis au sein du tableau de la figure 49.

Systèmes multi-instrument monosurface Notation UDP/C : N-1-N/1, sauf pour Pebbles : N-1-N/*	Découverte des acteurs et des ressources d'interaction	Localisation des entités physiques	Modélisation du couplage	Interface migrable et distribuable	Architecture distribuée	Conformité de la latence	Réutilisation de l'existant	Facilité de programmation
MMM (Multi-Device Multi-User Multi-Editor)	-	-	+	-	-	++	-	+
MID (Multiple Input Devices)	-	-	-	-	-	++	+	++
Pebbles	-	-	-	-	-	++	-	+
Dynamo	-	-	-	-	+	++	++	++
DiamondSpin	-	-	-	+	-	+	+	++
Table Magique	++	++	+	+	-	++	-	++
Group Pointer Interaction	+	+	-	-	-	+	+	+

Figure 49. Tableau récapitulatif de la classification des systèmes multi-instrument monosurface.

Ici, nous parlons de système monosurface. Ainsi, le requis d'interface migrable et distribuable n'a pas beaucoup de sens. Cependant, comme ce requis comprend la gestion du facteur de rotation et de zoom, nous utilisons la case du tableau associée pour voir les systèmes qui utilisent ces paramètres.

En somme, la Table magique est le meilleur de ces systèmes si l'on considère l'obligation d'utiliser une nouvelle boîte à outil comme une action incontournable. Si par contre l'on considère qu'il faut absolument réutiliser l'existant, alors Dynamo est le plus performant des systèmes existants car il réutilise une partie des applications qui fonctionnent sur Windows.

VI.2.3. SYSTÈMES MULTISURFACE MONO-INSTRUMENT

La catégorie mono-instrument multisurface comprend les systèmes qui utilisent plusieurs surfaces, chacunes gérées par des machines différentes, et qui n'autorisent qu'un instrument de pointage, ou qu'un couple d'instruments de pointage et de saisie de texte, par surface. Ces systèmes permettent la création d'un espace d'interaction unifié composé de

plusieurs surfaces hétérogènes sur lesquelles l'IHM peut être distribuée.

Les systèmes mono-instrument multisurface que nous retenons de la littérature pour analyse sont les suivants :

- EasyLiving [Brumitt et al 2002],
- MightyMouse [Kellogg 2002],
- PointRight [Johanson et al 2002a],
- Beach [Tandler 2001].

Suivant la notation UDP/C, les systèmes de cette catégorie se notent N-N-1 car plusieurs utilisateurs, plusieurs "displays" et un seul pointeur par display sont mis en jeu. Comme tous ces systèmes, pour permettre l'utilisation de plusieurs surfaces, utilisent plusieurs ordinateurs la notation UDP/C complète est N-1-N/*. Il faut noter que MightyMouse est mono-utilisateur et se note donc 1-N-1. Cependant, une extension se rapprochant de la solution PointRight pour le cas multi-utilisateur est envisagée par les auteurs.

EasyLiving permet à un ou plusieurs utilisateurs, situé(s) dans un bâtiment, de disposer l'interface graphique de leurs applications sur la surface la plus appropriée (ex : la plus proche de l'utilisateur). A tout instant, l'utilisateur peut décider la migration de l'interface d'une application d'une surface à une autre. EasyLiving est assez souple puisqu'il unifie l'espace d'affichage en permettant la migration et la distribution des interacteurs au grain application. La migration et la distribution d'une application ne sont pas dépendantes du choix effectué pour les autres applications. Cependant, le curseur ne pouvant pas migrer, ce système ne permet pas de contrôler, avec les instruments permettant d'agir sur une surface S1, des applications dont l'IHM est visible sur une surface S2. Pour agir sur un interacteur distant l'utilisateur doit, soit changer d'instrument, soit utiliser une interface de contrôle pour effectuer la migration de l'application concernée vers une surface sur laquelle une interaction est possible. Opérations qui peuvent vite devenir fastidieuses.

MightyMouse et PointRight sont deux systèmes très semblables basés sur VNC qui, grâce à la redirection d'événements via un serveur, permettent à un ou plusieurs utilisateurs d'utiliser un espace d'interaction unifié composé de plusieurs surfaces. Dans une pièce équipée d'un tel serveur, un utilisateur peut très simplement avec la même souris agir sur n'importe quelle surface. La technique d'interaction est en effet ingénieuse : lorsque le curseur sort d'une surface S1, il réapparait sur la surface S2 située dans sa continuité. Cette migration du curseur est une illusion puisque le curseur ne migre pas sur S2 mais l'utilisateur prend le contrôle du curseur géré par le système d'exploitation local à la sur-

face S2. En effet, le curseur de S1 est toujours visible ; il ne reçoit simplement plus les événements de l'instrument qui sont redirigés vers S2. Avec cette technique, il ne peut malheureusement y avoir qu'un seul curseur sur une surface (celui géré par le système d'exploitation). Ainsi, une même surface ne peut pas être utilisée au même moment par deux utilisateurs. Ici, c'est chacun son tour ! Dans ces circonstances, des règles sociales doivent être définies par les utilisateurs.

Avec Beach, nous retrouvons le principe d'un seul instrument de pointage par surface. Cependant, à l'inverse de MightyMouse et PointRight, Beach ne permet pas la migration du curseur. La technique d'interaction appellée «Drag & Pop» doit permettre de surmonter cette impossibilité en rapprochant de l'utilisateur l'interface qu'il veut manipuler. Nous verrons dans la section concernant le requis d'interface migrable et distribuable, que l'avance de Beach sur ces trois systèmes se situe dans la gestion de la migration et la distribution des interacteurs entre les surfaces.

Découverte des acteurs et des ressources d'interaction

L'ajout et le retrait des ressources d'interaction n'est pas dynamique. Il ne peut se produire qu'entre deux sessions. Pour être découverte les surfaces doivent avoir été décrites dans un fichier stocké sur le serveur de ces "roomwares". Seul PointRight laisse entrevoir un peu de souplesse en permettant de déclarer en cours d'utilisation du système une surface active ou inactive. La dynamicité n'est donc pas le point fort de ces solutions qui supposent un environnement augmenté plutôt stable et homogène.

Localisation des entités physiques

Tous ces systèmes multisurface modélisent la configuration spatiale des surfaces. Il semble cette configuration soit gérée par l'intermédiaire d'un fichier de configuration chargé au démarrage. La description la plus complète et la plus souple est celle définie par PointRight. Cette décription est visible dans le chapitre 3 sur les relations spatiales. Les surfaces peuvent être assemblées en 2D1/2 alors que MightyMouse n'autorise qu'une configuration linéaire gauche-droite (1D ou 2D linéaire). Beach permet aussi des assemblages en 2D1/2 mais les auteurs ne donnent aucune indication sur la gestion de cette caractéristique. On suppose que tous ces informations sont incluses dans le code.

Même si ces systèmes utilisent la notion de relation spatiale, sa modélisation est plutôt succinte et simpliste. Néanmoins, il faut noter l'effort fait par les auteurs de MightyMouse qui ont défini une interface graphique à leur système permettant à l'utilisateur de modifier la configuration spatiale des ressources.

A ces caractéristiques qui montrent que ce requis n'est pris en compte que partiellement, s'ajoute le fait que les relations spatiales logiques sont directement construites sur les relations physiques ne laissant

aucune souplesse d'interprétation aux applications. Par exemple, si le fichier de configuration définit un espace d'un mètre entre deux surfaces alors le pointeur de la souris sera invisible pendant le franchissement cet espace. Alors que, comme la figure 47 du chapitre 5 le montre, le curseur pourrait passer directement d'une surface à l'autre si l'espace logique était construit par interprétation de l'espace physique de manière à annuler cet espace.

Modélisation du couplage

Ces systèmes gèrent le couplage de surfaces afin de créer un espace d'interaction plus grand. Avec PointRight et MightyMouse, le couplage instrument-surface évolue dynamiquement lorsque le curseur franchit la frontière entre deux surfaces. Cependant, aucun des états de l'automate proposé par Barralon n'est directement observable : aucune information ne permet de savoir comment les surfaces sont couplées, quelle sémantique définit ce couplage et comment procéder à une modification de celui-ci si cela est possible.

Quant au couplage entre surfaces, il est statique car il est basé soit sur une localisation des ressources d'interaction tirée d'un fichier de configuration soit défini directement dans le code du système.

En ce qui concerne le couplage instrument-instrument, aucune information provenant de l'interface ne permet de savoir quel est l'état du couplage et s'il est vraiment géré. Nous pouvons simplement supposer qu'avec le système PointRight les événements claviers vont à l'interacteur sélectionné par la souris associée. Pour MightyMouse il en est de même car aucune précision n'est donné sur ce mécanisme. Enfin, avec les deux autres systèmes ce problème est complètement ignoré.

Interface migrable et distribuable

Ces solutions techniques permettent la migration et la distribution des interacteurs entre les surfaces. Cependant, ces fonctionnalités ne sont pas gérées avec la même souplesse selon le système :

- EasyLiving gère la migration et la distribution au niveau application mais pas du tout le zoom, la rotation et le glissement.
- MightyMouse et PointRight gèrent la migration et la distribution au niveau global (interdisant l'affichage simultanée sur la même surface de l'interface graphique de deux applications fonctionnant sur deux plate-formes différentes) et omettent le zoom, la rotation et le glissement.
- Beach gère la migration et la distribution au niveau interacteur et la rotation au niveau espace de travail mais pas le zoom ni le glissement.

Comme Beach ne gère pas le facteur d'échelle (comme les autres d'ailleurs), il est impossible, sauf si les surfaces sont homogènes, de disposer une interface graphique à cheval entre celles-ci sans problème

de discontinuité. Ainsi, Dynawall et les ConnecTables utilisent à leur avantage l'homogènéité des surfaces pour donner l'impression de résoudre ce problème.

Le grand avantage de Beach sur les autres systèmes réside dans sa gestion de la rotation de l'interface graphique qui permet l'utilisation d'une table (InteracTable) dans un contexte multi-utilisateur. Beach est, par toutes ses caractéristiques, une solution très avancée du point du vue graphique.

Architecture distribuée

Ces différents roomwares utilisent un serveur pour gérer la cohérence globale du système. Comme pour les solutions multi-instrument mono-surface sans ce serveur tout le système s'écroule. Ce requis n'est donc pas respecté.

Conformité de la latence

Même si nous n'avons pas de donnée quantitative, l'analyse des vidéos de ces systèmes montre que qualitativement ce requis n'est respecté que dans certains cas. Les situations où la latence du système est perceptible s'expliquent par, soit un temps de traitement trop long des événements (PointRight), soit une gestion trop lourde de la cohérence du système lors de la migration de l'IHM (Beach), ou encore, par le temps pris pour la migration de threads (EasyLiving).

Réutilisation de l'existant

EasyLiving, MightyMouse et PointRight, contrairement à Beach, permettent la réutilisation des applications existantes sans aucune modification. En effet, ces systèmes récupèrent soit les événements bas-niveau, soit les Threads systèmes, soit une image de l'IHM et les font migrer de machine en machine donnant ainsi l'illusion d'un espace unifié. Cependant, il y a une différence : MightyMouse et PointRight permettent une réutilisation de l'existant quel que soit le système d'exploitation alors qu'EasyLiving ne fonctionne qu'avec Windows.

De son coté, Beach ne permet pas la réutilisation des applications existantes car pour être utilisées avec ce système, une application doit être construite sur l'intergiciel Coast [Schuckmann et al 1996] dont Beach est une sur-couche. Cette caractéristique est donc un handicap que les autres systèmes n'ont pas.

Facilité de programmation

Comme EasyLiving, MightyMouse et PointRight réutilisent directement les applications préexistantes, ils vérifient ce requis. Cependant, ils offrent moins de services et de souplesse que Beach, qui lui est beaucoup plus difficile à appréhender. En particulier, certains choix techniques provenant de Coast ont tendance à complexifier la programmation (objets distribués, conception centrée document, mauvaise documentation, complexité de l'architecture).

Synthèse Dans cette catégorie de système, le programmeur a le choix entre des systèmes qui offrent une programmation simple mais des services limités et peu souples et un système (Beach) qui, à l'inverse, utilisent des concepts de programmation plus complexes mais fournis des services plus importants. Cependant, leurs apports sont encore limités pour apport à ce que le programmeur est en droit d'attendre. Cette analyse se retrouve dans le tableau de synthèse de la figure 50.

Systèmes mono-instrument multisurface Notation UDP/C : N-N-1/*	Découverte des acteurs et des ressources d'interaction	Localisation des entités physiques	Modélisation du couplage	Interface migrable et distribuable	Architecture distribuée	Conformité de la latence	Réutilisation de l'existant	Facilité de programmation
EasyLiving	-	+	-	+	-	+	++	++
MightyMouse	-	+	+	+	-	+	++	++
PointRight	-	+	+	+	-	+	++	++
Beach	-	+	-	+	-	+	-	-

Figure 50. Tableau récapitulatif de la classfication des systèmes mono-instrument multisurface.

Beach est le meilleur de ces systèmes si l'on considère l'obligation d'utiliser une nouvelle boîte à outil comme un souci négligeable. En effet, il offre plus de souplesse pour la distribution de l'IHM que ses concurrents. Si la réutilisation de l'existant prime, alors on peut considérer que les trois autres systèmes tirent leur épingle du jeu. Par contre, comme ils offrent des fonctionnalités différentes, désigner le meilleur de ces systèmes est une chose délicate.

VI.2.4. SYSTÈMES MULTI-INSTRUMENT MULTISURFACE

Cette catégorie regroupe les systèmes qui permettent l'utilisation simultanée de plusieurs surfaces et de plusieurs instruments par surface. Autrement dit, suivant la notation UDP/C, ces systèmes rentrent dans la catégorie 1-N-N/* si le système est mono-utilisateur ou N-N-N/* s'il est multi-utilisateur.

Historiquement, le premier système de cette catégorie est le prototype «Augmented Surfaces» de Rekimoto [Rekimoto et Saitoh 1999]. Néanmoins, nous ne pouvons retenir ce prototype comme une solution

effective au problème de l'interaction multi-instrument multisurface car aucune indication sur son fonctionnement n'a été publiée. De ce fait, ce prototype illustre seulement l'intérêt d'un environnement multi-instrument multisurface. Il faut noter que ce système introduit des techniques d'interaction dédiées à cette situation. Cependant, celles-ci n'ont jamais fait l'objet d'étude rigoureuse.

Comme ce prototype ne donne aucune piste quant à la réalisation d'une solution logicielle concrète, il n'existe, à notre connaissance, qu'une seule solution effective dans cette catégorie : le système Ubit [Lecolinet 2003a][Lecolinet 2003b] et l'extension associée UMS (Ubit Mouse Server). Ubit gère la partie multisurface alors que UMS gère la partie multi-instrument. Ces systèmes sont tous les deux basés sur XWindow.

Ubit est une boîte à outil graphique basé sur une nouvelle architecture qui utilise les graphes de scènes pour créer des interfaces graphiques innovantes telles que les lentilles magiques, les outils transparents ou le zoom sémantique. Cette architecture rend possible, avec un peu d'effort, la création d'interfaces graphiques distribuées auquelles le principe des vues multiples peut être appliqué dans un contexte multi-surface.

Le serveur multi-pointeurs UMS permet de gérer plusieurs pointeurs sur une même surface. Il fonctionne en collaboration avec le serveur X dont il étend les fonctionnalités. Il peut fonctionner avec des applications X quelconques. Néanmoins, certaines fonctionnalités avancées ne peuvent être exploitées que par des applications compatibles (c'est-à-dire des applications connaissant la structure des événements X étendus par UMS afin d'utiliser l'identifiant stocké dans l'événement dénotant l'instrument émetteur). Comme XWindow ne fournit pas de moyen de gérer plusieurs pointeurs, UMS crée des simulis de pointeurs en utilisant de petites fenêtres sans bordure dans lesquelles un curseur est dessiné. Plusieurs serveurs UMS peuvent être chaînés afin qu'un pointeur puisse sembler passer d'une surface à une autre lorsqu'il atteint un bord. Le serveur qui gère le pointeur migrant prend le contrôle d'un pointeur situé sur la surface destination et envoie l'événement de déplacement au serveur UMS qui gère cette surface. Ici, la gestion des événements n'est donc pas centralisée contrairement à ce que nous avions vu dans la catégorie multi-instrument monosurface.

Découverte des acteurs et des ressources d'interaction

Ubit et UMS permettent la migration et la distribution de l'interface graphique et l'utilisation simultanée de plusieurs instruments de manière statique car ils ne donnent aucune information sur l'arrivée, la présence et le départ de ressources d'interaction dans l'environnement. Le scénario d'utilisation de l'espace d'interaction rendu accessible par Ubit et UMS doit être pré-cablé dans le code des applications. En effet,

pour pouvoir utiliser les surfaces une application doit connaître l'adresse IP du serveur X associé. Toutes les surfaces pouvant être utilisées par une application doivent être connues à l'avance par le programmeur. De même , la connaissance des caractéristiques physiques des ressources d'interaction repose sur le principe d'une connaissance a priori. Ces limitations vont à l'encontre de notre vison d'un espace d'interaction unifié qui se construit dynamiquement.

Localisation des entités physiques

Avec UMS, la configuration spatiale des surfaces utiles à la migration du curseur est spécifiée au chargement des serveurs selon une méthode non précisée par l'auteur. De plus, cette topologie n'est modifiable qu'entre deux sessions d'un serveur UMS. Dès lors, Ubit ne permet pas d'utiliser les relations spatiales pour créer un espace unifié reflétant le monde physique. L'espace logique ne peut au mieux que refléter le monde physique connu du programmeur au moment du développement de l'application.

Modélisation du couplage

Ubit permet le couplage surface-surface afin de créer un espace d'interaction plus grand. Cependant, on retrouve ici les mêmes problèmes liés à la non prise en compte de l'automate de couplage et des propriétés de souplesse et de robustesse associées. Notamment, le couplage des surfaces n'est pas observable. Cependant, ce couplage peut être dynamique ; mais l'acteur responsable de cette tâche est le programmeur. Il doit lui-même gérer la distribution et la migration de l'IHM en fonction des couplages qu'il choisit d'effectuer. Ubit ne lui offre aucun service simplificateur si ce n'est le graphe de scène.

Le couplage entre instruments de pointage existe dans le cas d'une interaction à deux mains ou à plusieurs à condition d'utiliser les interacteurs adéquats. Par ailleurs, comme le couplage clavier-souris ne fonctionne qu'avec le pointeur natif, les pointeurs fictifs liés à des instruments distants ne sont pas utilisables dans toutes les situations. Les utilisateurs doivent donc se mettre d'accord socialement s'ils veulent partager la même surface. De plus, un curseur fictif ne peut pas rendre observable l'état du système car sa forme n'est pas modifiable par programmation. Pour rendre cela possible, il faudrait effectuer une modification du serveur X ce qui reste pour l'auteur une perspective de travail.

Ubit et UMS constituent une avancée par rapport à des systèmes comme PointRight (pour le multisurface) et MID (pour le multi-instrument) mais ils imposent de fortes contraintes sur la gestion du couplage des ressources d'interaction.

Interface migrable et distribuable

Ubit gère la projection d'un même interacteur sur plusieurs surfaces. Cette projection peut être modifiée dynamiquement par programmation. Cette fonctionnalité, rendue possible par l'utilisation de graphes

de scène permet d'afficher simplement une même fenêtre avec la même hiérachie d'interacteurs sur deux surfaces distinctes. Pour cela, il suffit de créer une hiérarchie d'interacteur et une fenêtre sur chaque surface et d'ajouter cette hiérarchie à chacune de ces deux fenêtres. Cette action a pour effet de partager le graphe de scène entre les deux fenêtres. Ubit étant basé sur Xwindow, il faut connaître à l'avance l'identifiant d'une surface (display) pour créer une fenêtre dessus (connaissance à priori). Ce principe rend cette solution non adaptée au cadre de l'informatique diffuse. Néanmoins, il faut noter que le graphe de scène peut être «tagué» afin de procéder à un affichage différent sur chaque surface permettant ainsi une légère plastification de l'IHM.

Ubit gère le zoom des interacteurs contenus dans une fenêtre mais pas leurs rotations. Le zoom de la fenêtre proprement dite n'est pas possible car celle-ci est gérée par le window manager et non par Ubit. Ainsi, à cause de la différence de taille de la barre de titre d'une surface à l'autre (celle-ci dépend de la résolution d'affichage), l'affichage continu d'une fenêtre à cheval entre plusieurs surfaces n'est possible qu'en milieu homogène. En conclusion, la non prise en compte des attributs physiques des surfaces, l'absence de gestion précise des relations spatiales et le manque d'encapsulation de la distribution de l'interface graphique empêche l'affichage correct d'un interacteur à cheval entre deux écrans.

Architecture distribuée

Ubit et UMS héritent tous deux de l'architecture de Xwindow. Ainsi, par rapport aux systèmes précédemment décris, la solution qu'offre la combinaison de ces deux systèmes a réellement une architecture distribuée qui garantit son utilisabilité partielle même si un des serveurs utilisés disparait. Néanmoins, cette solution ne gérant pas la découverte dynamique des ressources d'interaction, si un serveur X utilisé par une application est arrêté puis relancé, celle-ci voit le départ des ressources (fermeture du serveur X) mais pas leur retour.

Conformité de la latence

Nous n'avons aucune donnée quantitative sur le respect de ce requis. Néanmoins, au vu de l'architecture utilisée, nous pensons que cette solution ne souffre pas ou peu du problème de latence.

Facilité de programmation

La principale caractéristique d'Ubit d'être basée sur une architecture atomique. 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 cette boîte à outil. Les widgets sont des combinaisons préétablies de briques qu'il est toujours possible de modifier dynamiquement à l'exécution. Cette caractéristique facilite le développement d'inter-

face graphique. De plus, le graphe de scène offre une base, rudimentaire certes mais une base quand même, au développement d'IHM distribuées et quelque peu plastiques.

Cependant, même si le graphe de scène aide à la distribution, la migration reste l'affaire de programmeur puisque le déplacement conjoint des vues multiples d'une même interface n'est pas un service offert.

Réutilisation de l'existant

Pour utiliser les fonctionnalités d'Ubit, le développeur doit s'appuyer sur l'API proposée par son auteur est non sur XWindow. Ubit n'autorise pas la réutilisation directe des applications existantes. Par contre, une application conventionnelle peut être contrôlée par un faux pointeur géré par UMS. Cependant, seules celles compatibles Ubit peuvent utiliser les fonctionnalités multi-instrument avancées (ex : interaction à deux mains) et multisurface (distribution de l'IHM). Pourtant, l'architecture retenue pour Ubit et UMS permet l'affichage et l'utilisation d'applications compatibles ou non au sein du même espace.

Synthèse

Ubit, grâce à son graphe de scène, est une solution beaucoup plus avancée que les autres. Cependant, l'absence de modélisation du monde physique et du couplage des ressources d'interaction en font en système statique non adéquat au contexte de l'interaction multi-instrument multisurface. Ce constat est résumé par le tableau de la figure 51.

Systèmes multi-instrument multisurface Notation UDP/C : N-N-N/*	Découverte des acteurs et des ressources d'interaction	Localisation des entités physiques	Modélisation du couplage	Interface migrable et distribuable	Architecture distribuée	Conformité de la latence	Réutilisation de l'existant	Facilité de programmation
Ubit et UMS	-	+	+	+	+	++	-	+

Figure 51. Tableau récapitulatif des systèmes multi-instrument multisurface.

VI.3. Synthèse

Cette classification nous a permis d'analyser l'adéquation des solutions logicielles existantes au regard des requis logiciels pour l'interaction en informatique diffuse. Elle motive notre constat : aucun système ne

prend en compte l'ensemble des requis. Par conséquent, aucun système ne peut être aujourd'hui une solution effective.

On pourrait néanmoins espérer combiner élégamment certaines solutions afin d'atteindre notre but : la création d'un espace unifié dynamiquement configurable multi-instrument multisurface. Malheureusement, notre analyse montre que cette possibilité doit être écartée car ces solutions sont trop hétérogènes pour envisager une telle combinaison.

Suite à ce constat, notre approche est de construire notre propre solution en utilisant comme guides les requis précédemment cités. Le chapitre suivant explicite l'architecture de notre solution logicielle et ses fonctionnalités. Ce chapitre est suivi d'une évaluation de notre solution vis-à-vis des requis et d'une présentation des perspectives de travail en vue de son amélioration.

Chapitre VII *I-AM, Machine Abstraite d'Interaction*

I-AM (Interaction Abstract Machine) est une machine du même niveau d'abstraction que les gestionnaires graphiques de base comme Xwindow. Ces gestionnaires gèrent les ressources d'interaction d'une station de travail dont ils assurent le partage entre les applications clientes. Comme ces gestionnaires, I-AM est un intergiciel orienté Interaction Homme-Machine, mais I-AM en étend la couverture fonctionnelle pour tenir compte des requis de l'informatique ambiante. En effet, I-AM permet la construction dynamique d'espaces interactifs dont les surfaces et les instruments sont gérés par des stations de travail distinctes exécutant des systèmes d'exploitation éventuellement différents (MacOS, Windows), et donne l'illusion, au programmeur comme à l'utilisateur, que ces ressources sont gérées de manière uniforme par une seule station de travail.

Plus spécifiquement, I-AM :

- Masque l'hétérogénéité du matériel et des systèmes d'exploitation sous-jacents au bon niveau d'abstraction,
- Assure la découverte dynamique des ressources d'interaction,
- Modélise les relations spatiales entre les ressources d'interaction et leur couplage,
- Permet la distribution et la migration des IHM graphiques au niveau du pixel.

Dans ce chapitre, nous décrivons I-AM en détail. Nous en présentons les principes, la structure générale en niveaux d'abstraction, puis les hypothèses et limitations actuelles. Ces éléments généraux sont ensuite complétés par la présentation de chacun des composants logiciels d'I-AM.

VII.1. Principes

I-AM repose sur deux principes directeurs : un espace d'interaction, premièrement unifié, et deuxièmement configurable par couplage de surfaces.

VII.1.1. ESPACE D'INTERACTION UNIFIÉ

La figure 52 illustre les principes d'I-AM selon le double point de vue de l'utilisateur et du programmeur :

- Au bas de la figure, la vue de l'utilisateur : dans cet exemple, l'IHM graphique est distribuée sur trois surfaces gérées chacune par une machine exécutant des systèmes d'exploitation distincts (MacOS, Windows XP, Windows NT). Les fenêtres peuvent être déplacées dans tout l'espace d'affichage que forment les trois surfaces et ceci par programme ou par l'utilisateur au moyen de n'importe quel instrument. Par exemple, en utilisant la souris reliée à la machine de droite, l'utilisateur peut manipuler une fenêtre affichée sur la surface reliée à la machine de gauche.
- Au sommet de la figure, la vue du développeur. Ici, les fenêtres sont créées et gérées dans un espace logique uniforme. Dans cet espace, l'existence des surfaces physiques et de leur machine hôte est masquée au programmeur, mais leurs caractéristiques restent néanmoins accessibles. La figure 53 complète notre description du principe. On constate que la diversité des ressources d'interaction, des machines et des systèmes d'exploitation sous-jacents est encapsulée et qu'I-AM assure ce service en respectant à la lettre le paradigme de programmation des gestionnaires graphiques classiques.

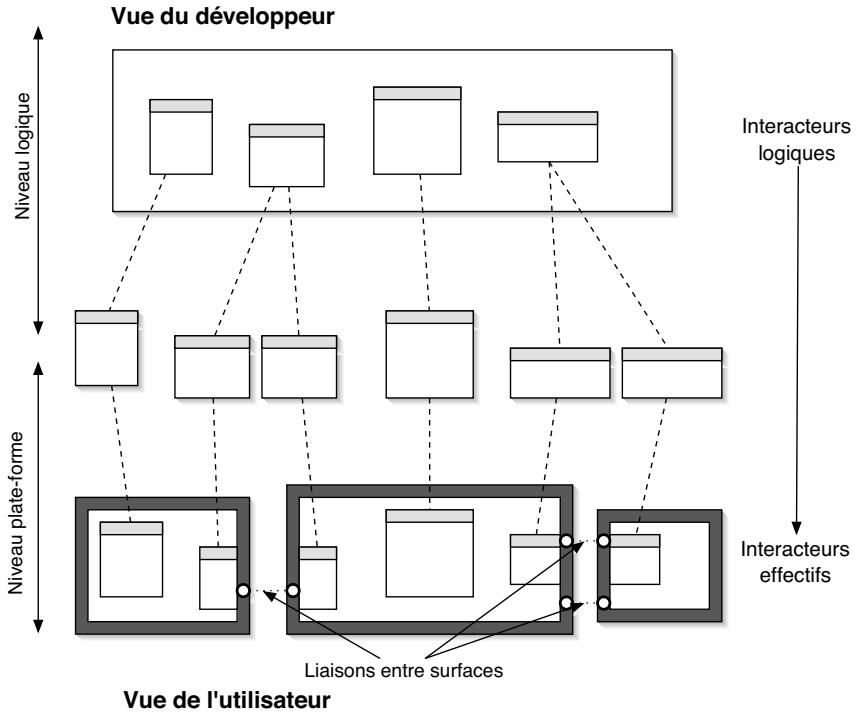

Figure 52. Le principe de la distribution des interacteurs dans I-AM selon la vue du développeur ou celle de l'utilisateur.

I-AM maintient la correspondance entre l'espace numérique logique et l'espace physique d'affichage. La figure 52 montre le principe de cette projection. La fenêtre la plus à gauche de l'espace logique tient sur une seule surface. D'autres, comme la fenêtre la plus à droite, sont réparties sur deux surfaces. Dans ce cas, l'interacteur logique de la vue du développeur est projeté sur chaque surface en deux interacteurs effectifs. Ces derniers sont affichés à la bonne échelle et à la bonne position en sorte que l'utilisateur ait l'illusion d'un interacteur unique. Ainsi, lorsque l'utilisateur manipule l'un des interacteurs effectifs, l'interacteur effectif "jumeau" réagit en conformité avec les réactions du premier. D'une manière générale, le nombre d'interacteurs effectifs correspondant au rendu d'un interacteur logique dépend de la position de l'interacteur logique dans l'espace logique, des relations spatiales des surfaces et de la fonction de projection de l'espace logique sur l'espace physique.

```
// Création d'une IAMapp
IAMApp myIamapp = new IAMApp ();

// création d'une fenêtre centrée sur le point (100, 100)
// de taille 300 par 200

IAMWindow maFenetre = new IAMWindow
                    (myIamapp, 100, 100, 300, 200);

maFenetre.setCenterLocation (100, 150);
```

Figure 53. Le paradigme de programmation d'I-AM.

VII.1.2. CONFIGURATION D'ESPACE INTERACTIF

Un espace interactif, nous l'avons dit, est dynamique. Dans I-AM, la construction d'un espace interactif s'effectue par couplage de surfaces. Le couplage de deux surfaces a lieu par mise en correspondance de points de liaison. Un point de liaison est un point fixe situé sur les bords d'une surface dont I-AM connaît la position et l'identité. Concrètement, il peut prendre la forme de capteurs (capteur infrarouge [Dietz et al 2003], accéléromètre comme dans l'exemple de Hinckley [Hinckley 2003]) ou de taches de couleur repérables par vision par ordinateur. En plaçant face à face deux points de liaison de deux surfaces distinctes, l'utilisateur réalise un couplage de ces surfaces et par voie de conséquence, la connexion de leur machine hôte. La fonction obtenue est un agrandissement de l'espace d'affichage avec héritage des instruments des machines hôtes. Grâce au positionnement et à l'identité des points de liaison, I-AM calcule les relations spatiales entre les surfaces.

Les images de la figure 54, montre I-AM en action. En a), deux machines (un PC et un Macintosh) sont reliées au réseau, mais leurs surfaces ne sont pas couplées. Dans cet état, les fenêtres du PC et du Macintosh ne peuvent donc transiter entre les écrans. En b), l'utilisateur couple les écrans des machines en les rapprochant (l'espace s'agrandit). En c) la fenêtre du Macintosh est en transition vers le PC au moyen de la souris du PC, mais la souris du Macintosh aurait pu être utilisée pareillement. En d) l'utilisateur saisit un texte dans la fenêtre située sur le Macintosh au moyen du clavier du PC. Au préalable, l'utilisateur avait sélectionné l'interacteur de saisie de texte au moyen de la souris du PC.

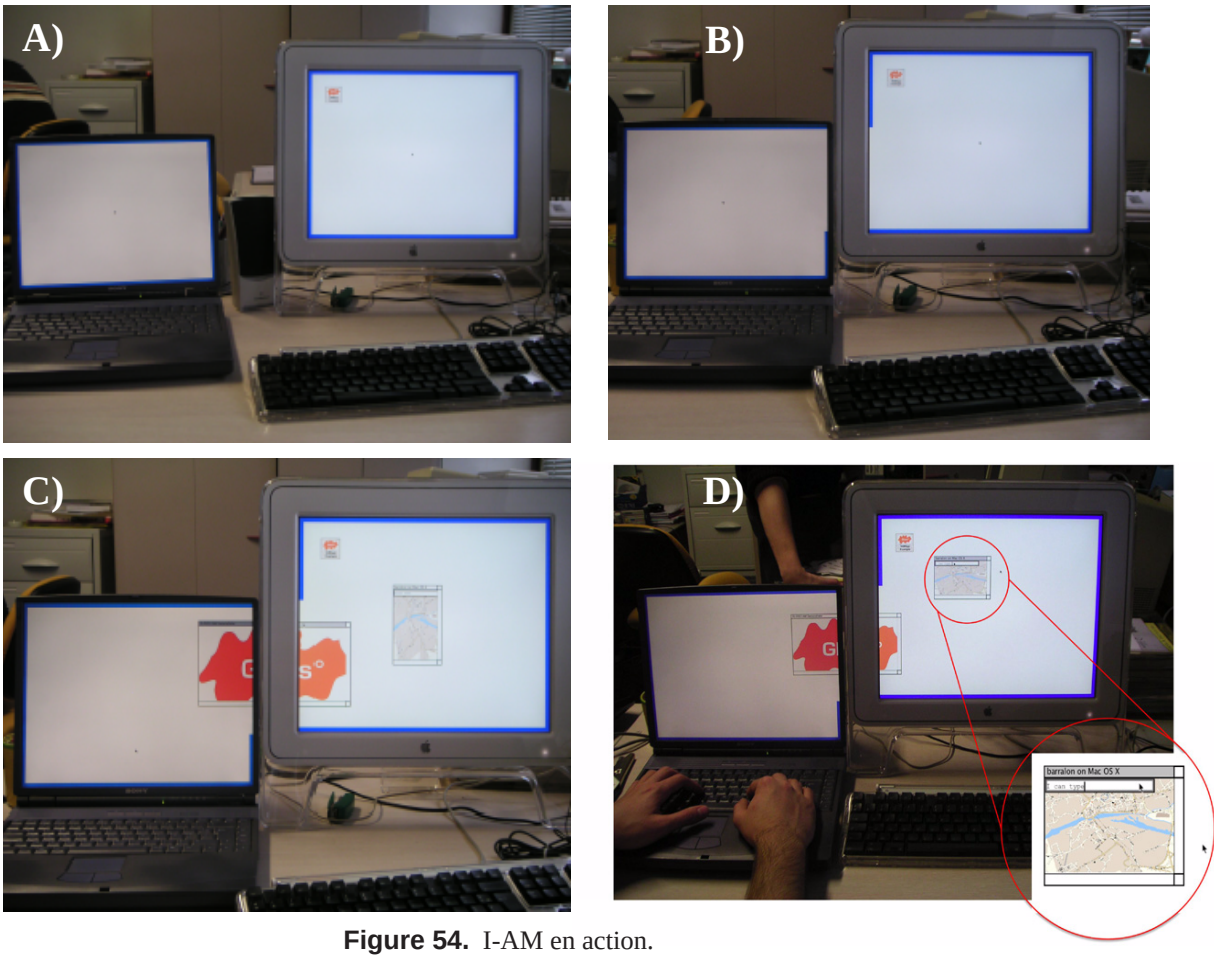

Figure 54. I-AM en action.

Nous venons d'introduire les services d'I-AM dans leurs grandes lignes. En pratique, la réalisation d'I-AM obéit à des hypothèses et limitations que nous précisons ci-dessous.

VII.2. Hypothèses et limitations

La mise en œuvre d'I-AM s'appuie sur deux hypothèses et présente trois limitations.

VII.2.1. HYPOTHÈSES

Hypothèse 1 *Machine hôte IP*

Pour la communication de base entre les machines d'un espace interactif, I-AM suppose l'existence d'un réseau local et des protocoles TCP et UDP dans lesquels les machines sont identifiées par une adresse IP.

Hypothèse 2 *Machines équipées d'instruments*

Nous supposons qu'une machine est toujours équipée d'au moins un instrument de pointage et d'un instrument de saisie de texte. Avec cette hypothèse, la découverte/le retrait d'instruments revient à gérer la découverte/le retrait de leur machine hôte.

VII.2.2. LIMITATIONS

Limitation 1 *Surfaces rectangulaires et couplage planaire*

Les surfaces que gèrent I-AM sont supposées planaires, de forme rectangulaire, éventuellement serties dans un cadre sur lequel le système ne peut projeter des interacteurs. Ces surfaces peuvent être les écrans usuels de nos stations de travail ou des parties planes de l'espace (tables ou mur) si le système utilise des projecteurs vidéo. L'assemblage de surfaces par couplage ne peut s'effectuer que dans un plan. Toutefois, ce plan est pliable à la manière de PointRight (voir figure 23). I-AM pourrait gérer une topologie de surfaces en trois dimensions, mais, pour des raisons de temps, nous nous sommes limités à un espace 2D1/2.

Limitation 2 *Absence de support aux applications patrimoniales*

I-AM ne permet pas la réutilisation d'applications existantes. Les IHM des futures applications sont supposées réalisées au moyen d'interacteurs implémentés au-dessus des services de base d'I-AM. Ce choix est motivé par le potentiel de l'interaction multi-instrument multisurface avec migration/distribution d'interacteurs au niveau du pixel. Si l'on se reporte à notre analyse de l'état de l'art du chapitre 6, aucune des solutions actuelles assurant la migration/distribution d'interacteurs au niveau du pixel ne permet la réutilisation d'applications existantes.

Limitation 3 *Absence de support aux applications multi-utilisateur*

I-AM donne l'illusion d'un espace unifié de ressources d'interaction, mais ne propose pas de service de contrôle d'accès aux ressources d'interaction. Dans notre ontologie, nous avons souligné le caractère public, semi-public, privé d'une ressource d'interaction, et notamment des surfaces. Les mécanismes permettant de jouer sur le niveau de par-

tage d'I-AM n'a pas été étudié. Toutefois, I-AM assure bien le partage des ressources d'interaction entre les applications clientes.

Dans la section qui suit, nous présentons la structure générale d'I-AM avant d'entrer dans le détail de la mise en œuvre.

VII.3. Structure logicielle générale

I-AM reprend les niveaux d'abstraction des gestionnaires graphiques de base puisque, du point de vue des logiciels clients, il remplit le même rôle. Comme le montre la figure 55, on distingue :

- le niveau Plate-forme qui cache l'hétérogénéité des machines et des ressources d'interaction, gère leur découverte et disparition, et en publie l'existence,
- le niveau Application qui gère les espaces logiques propres à chaque application et en assure la projection sur l'espace physique,
- le niveau Interacteur correspondant au niveau boîte à outils graphique.

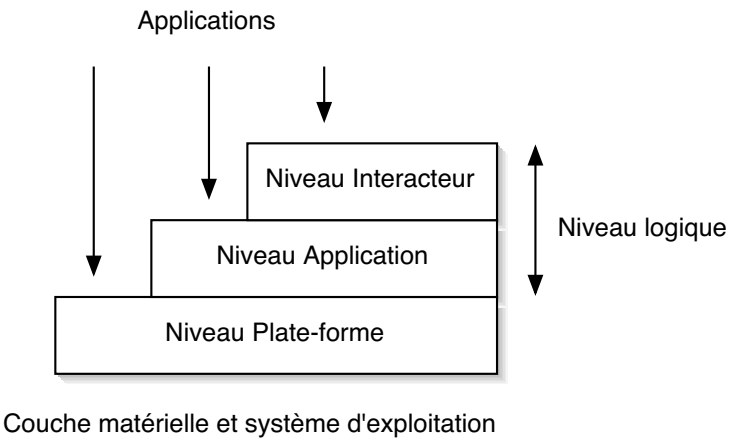

Figure 55. Les différentes couches logicielles définissant I-AM.

Ces trois niveaux fonctionnels sont répartis sur les machines hôtes constitutives de l'espace interactif, machines dont le nombre varie de manière imprévisible et pouvant former un réseau ad-hoc. En réponse à ces contraintes, nous avons adopté :

- une architecture paire à paire qui n'exige pas l'existence d'un serveur dédié ;
- la technique des proxys qui met en relation symétrique client et service : par exemple, toute application A utilisant une surface S est

représentée dans l'espace d'adressage de S par un proxy de A tandis que A représente S dans son espace par un proxy de S, les deux proxys communiquant par une voie point à point ;

- la technique de communication asynchrone de type abonnement/notification qui permet de gérer les événements dont l'occurrence est imprévisible. Par exemple, tout changement dans la configuration physique de l'espace interactif est traité selon cette technique.

Sur le plan technique, I-AM est implémenté en Java (Java 1.4.2). À chacun des niveaux d'abstraction cités ci-dessus, correspond un paquetage : IAMPlatform, IAMApp et IAMInteractor, respectivement. Nous les présentons maintenant de manière détaillée.

VII.4. Le niveau plate-forme d'I-AM (paquetage IAMPlatform)

Sur chaque machine d'un espace interactif est exécuté en tâche de fond un gestionnaire de plate-forme, instance de la classe IAMPlatformManager. Un IAMPlatformManager n'a aucune connaissance de l'existence des autres gestionnaires de plate-forme (architecture paire à paire). Cette absence de référence va dans le sens de la souplesse : l'espace interactif peut évoluer et se reconfigurer dynamiquement. Le lien entre les différentes plates-formes est assuré par le niveau supérieur, la couche IAMApp, qui choisit la glue qui lui convient.

Un gestionnaire de plate-forme doit assurer la gestion des ressources d'interaction locales, c'est-à-dire des ressources d'interaction reliées à la machine hôte du gestionnaire. Pour cela, il doit fournir les services suivants :

1) Acquisition et maintenance des caractéristiques des surfaces locales, et publication de leur existence : pour chaque surface, publication de son identité, de ses caractéristiques et de l'identité d'un port de communication. Ce port sert de point d'entrée initial aux applications intéressées par la surface ;

2) Notification auprès des applications clientes du changement d'état des surfaces par suite d'actions utilisateur ou système.

3) Réception et traitement des requêtes provenant des applications clientes sur les surfaces locales ;

4) Réception des événements provenant des instruments locaux (ou distants) avec redirection vers les applications clientes propriétaires de ces événements ;

Plus précisément, un IAMPlatformManager fonctionnant sur une plate-forme P crée les composants suivants :

- un IDManager, générateur d'identifiants de ressources d'interaction (service usuel de tout gestionnaire de ressources).
- un SurfacesContextor dont le rôle est de collecter les caractéristiques de toutes les surfaces locales de P, d'en publier l'existence et les changements d'état (satisfaction des requis 1 et 2 ci-dessus);
- un SurfaceManager par surface S de P qui assure la gestion de S et son partage avec les applications clientes (requis 3) ;
- un InstrumentsManager qui gère l'ensemble des instruments de P (requis 4);

Ces composants (IDManager, SurfacesContextor, SurfaceManager et InstrumentsManager) sont présentés en détail dans les sous-sections suivantes. La figure 56 en présente un diagramme de classes simplifié.

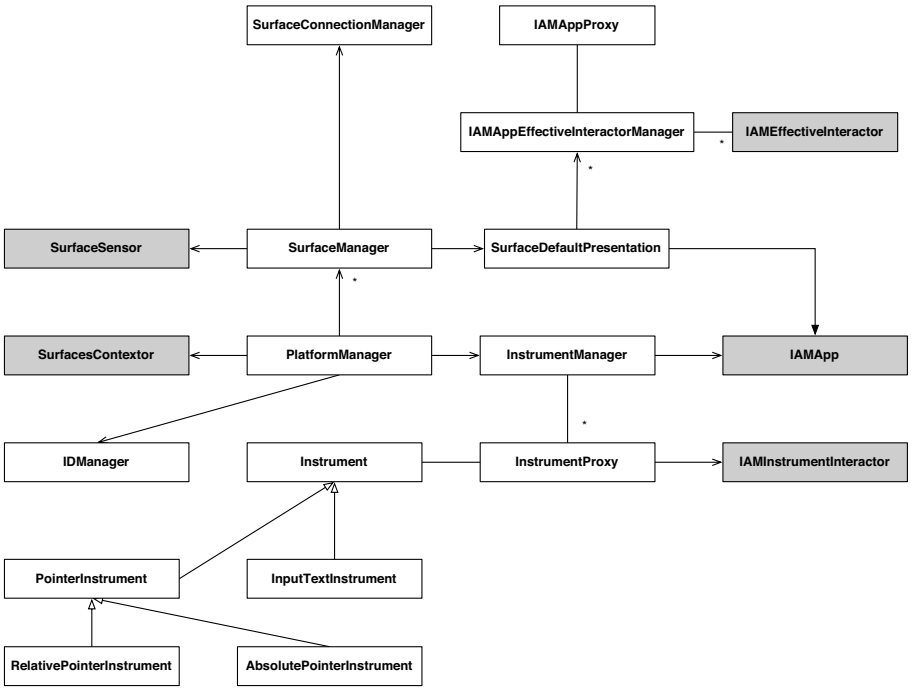

Figure 56. Diagramme de classes simplifié du paquetage IAMPlatform.

VII.4.1. IDMANAGER

Les ressources d'interaction nécessitent d'être identifiées de manière unique. Cette identification peut être assurée par un serveur de noms ou par des espaces de nommage par exemple. Notre modèle centré plate-forme nous permet de gérer le nommage des ressources à partir de l'adresse IP d'une plate-forme.

Une adresse IP définit de manière unique une plate-forme. Nous complétons cette identification de deux manières selon qu'il s'agit d'un

instrument ou d'une surface. Pour les instruments, l'adresse IP de la plate-forme P est suffixée par un entier calculé par l'IDManager de P. Par exemple, «128.56.78.98:21» est un identifiant d'instrument.

Pour les surfaces, l'adresse IP est suffixée par le numéro de port de sortie vidéo qui gère la surface (via un écran ou un vidéo projecteur) et un entier défini comme pour les instruments par l'IDManager. Cet entier est nécessaire car plusieurs surfaces (par exemple plusieurs murs) peuvent être gérées par un même vidéo projecteur comme dans le cas du PDS. Par exemple, "127.34.56.78:1:2" est un identifiant de surface.

Notre technique de génération d'identifiants convient aux besoins identifiés jusqu'ici. Si elle devait se révéler inadéquate, il suffirait de modifier le corps de la méthode getID de la classe IDManager.

VII.4.2. SURFACES CONTEXTOR

Un SurfacesContextor est une sous-classe de Contexteur. On trouvera dans [Rey et Coutaz 2004], une présentation détaillée de cette notion. En bref, un contexteur est un composant logiciel dont le rôle est de capturer de l'information sur le contexte d'interaction et de le fournir aux composants clients au niveau d'abstraction demandé. Les contexteurs s'exécutent au sein d'une infrastructure dite infrastructure de contexteurs dont l'architecture paire à paire permet un déploiement en accord avec nos besoins en connexions spontanées.

Dans I-AM, un et un seul exemplaire de SurfacesContextor est créé à l'initialisation du gestionnaire de plate-forme. Sa mission est la suivante : a) agréger les informations fournies par chaque surface locale grâce au mécanisme d'abonnement-publication de l'infrastructure des contexteurs, b) informer les composants qui en ont fait la demande (et notamment, les applications clientes intéressées par les ressources d'interaction de la plate-forme).

À l'opposé du système PointRight où les ressources sont décrites statiquement, I-AM dispose grâce à l'infrastructure de contexteurs, un mécanisme performant pour découvrir dynamiquement les surfaces, leurs caractéristiques et leurs couplages.

VII.4.3. SURFACE MANAGER

Chaque surface S locale à P dispose de son SurfaceManager. Ce SurfaceManager doit remplir deux grandes classes de fonction :

1) Acquérir et maintenir les caractéristiques physiques de S et les publier en sorte que les applications puissent l'exploiter ;

2) Assurer le partage de S entre plusieurs applications clientes. Ce partage suppose deux types de services :

- gérer les communications avec les applications clientes, éventuelle-

ment distantes, et

- pour chaque application cliente, assurer le rendu de ses interacteurs logiques sous forme d'interacteurs effectifs et inversement,
- acquérir les événements produits par les instruments sur les interacteurs effectifs et les rediriger sur les interacteurs logiques correspondants.

Acquisition, maintenance, publication des caractéristiques de S

L'Acquisition, la maintenance et la publication des caractéristiques de S sont assurées par le SurfaceSensor de S. Ce SurfaceSensor est un contexteur élémentaire que crée le SurfaceManager de S. Comme le montre la figure 57, les caractéristiques d'une surface incluent : la longueur et la largeur (à la fois en pixels et en millimètres) correspondant à une zone de projection, les dimensions du cadre en millimètres et le type de surface (écran, mur, table, etc). Les attributs de largeur et hauteur en pixels de la zone de projection sont calculés par le SurfaceSensor à l'aide de l'API du système d'exploitation natif. D'autres attributs sont récupérés dans un fichier de configuration au format XML. Ces caractéristiques et l'identité d'un port de communication pour les demandes d'utilisation de S sont publiées par le SurfaceSensor de S. Ces publications, on l'a vu en 4.2, sont récupérées par le SurfacesContexteur de P qui a exprimé son intérêt pour ce type d'information et qui publie à son tour cette information à des applications clientes du niveau IAMApp.

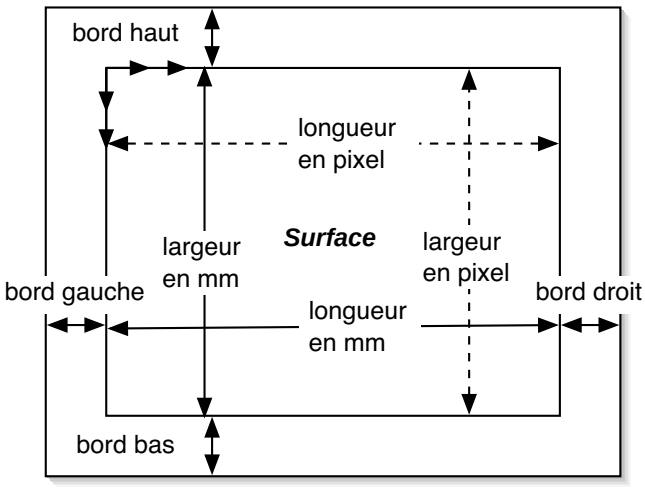

Figure 57. Les caractéristiques d'une surface sont sa taille en millimètre et en pixel ainsi que la taille de ses bords en millimètre.

Communication avec les applications clientes

Pour gérer les communications avec ses applications clientes, le SurfaceManager de S crée un SurfaceConnectionManager et un port de connexion sur lequel les applications envoient leur demande initiale d'utilisation (c'est l'identification de ce port que le SurfaceSensor

publie avec les caractéristiques physiques de S). Si une application A voit sa demande d'utilisation acceptée, elle est désormais représentée par le SurfaceManager de S par un proxy d'application. Symétriquement, A représente S dans son espace d'adressage par un proxy de surface. Dès lors, toutes les requêtes et notifications entre A et S sont échangées par le canal de communication établi entre leurs proxys. Et le SurfaceConnectionManager de S mémorise ainsi l'ensemble des applications clientes de S.

Rendu des interacteurs et acquisition des événements

Pour assurer le rendu des interacteurs et l'acquisition des événements sur S, le SurfaceManager de S doit :

- définir un système de coordonnées sur S. Par défaut, l'origine de ce repère est fixé au coin haut à gauche de S.
- créer un gestionnaire de rendu d'interacteurs et d'acquisition des événements sur S. Nous l'appelons IAMWindowManager par analogie au rôle fondamental des gestionnaires graphiques classiques.

Sachant que S peut être utilisée par plusieurs applications, son IAMWindowManager doit maintenir un état d'exploitation par application cliente. Cet état est géré par des IAMAppEffectiveInteractorsManagers, à raison d'un IAMAppEffectiveInteractorsManager par application cliente A. Le manager d'interacteurs effectifs de A sur S note, pour chaque IE de A, l'interacteur logique I correspondant, mais aussi la transformation affine et son inverse qui permettent de traduire les informations géométriques entre le repère de S et celui de l'espace logique. La transformation affine est fournie par A. Chaque application a ainsi le moyen d'exploiter l'espace interactif à sa façon. Quand une application est fermée, c'est-à-dire quand sa connexion avec S est détruite, son IAMAppEffectiveInteractorsManager est détruit et, par voie de conséquence, ses interacteurs effectifs sur S. Prenons l'exemple de la création d'un interacteur.

La création d'un interacteur logique I par A se traduit par une demande de création d'un IE correspondant sur toutes les surfaces utilisées par A . Cette demande, on le rappelle, est transmise sur les canaux qui relient les proxys des surfaces utilisées (et qui se trouvent dans l'espace d'adressage de A) aux proxys de A (qui se trouvent chacun dans un espace d'adressage de gestionnaire de plate-forme). Pour chaque surface S utilisée par A, l'IAMAppEffectiveInteractorsManager de A sur S crée (et mémorise) l'interacteur effectif IE correspondant à I et le projette sur S en appliquant la transformation affine fournie par A sur les informations géométriques de I (localisation, taille, etc.).

Pour l'acquisition des événements engendrés par l'utilisation d'instruments, l'IAMWindowManager doit :

1) Capturer les événements natifs du système hôte (Windows, MacOS),

2) Normaliser les événements natifs sous forme d'événements d'IAM (les IAMEvents),

3) Identifier, s'il en est besoin, l'IE propriétaire de l'IAMEvent et l'application A propriétaire de cet IE,

4) Identifier l'interacteur I concrétisé par cet IE en sorte que A traite l'effet de l'événement reçu par I.

Pour la condition 1), l'IAMWindowManager de S crée une zone d'affichage de même taille que S. Techniquement, cette zone est une fenêtre sans bord du gestionnaire graphique natif. L'IAMWindowManager s'abonne aux événements souris et clavier natifs de la fenêtre (MouseEvent et KeyEvent).

Pour la condition 2), le type d'événement natif permet d'identifier le type d'IAMInstrument correspondant : un IAMRelativePointerInstrument s'il s'agit d'un événement natif souris ou un IAMInputTextInstrument s'il s'agit d'un événement natif clavier. L'événement natif est transformé en IAMEvent qui contient entre autres, l'identification de l'IAMInstrument et une normalisation éventuelle dans le repère de S des données géométriques fournies par l'événement natif. Par exemple, deux souris n'ont pas nécessairement la même résolution. Par conséquent, pour un même geste humain de déplacement de souris, les <dx, dy> rendus peuvent être différents. Il faut donc ajuster les réglages.

Pour les conditions 3) et 4), l'IE, l'application propriétaire A de cet IE et l'interacteur logique I correspondant sont retrouvés via les IAMAppEffectiveInteractorsManagers. Le détail du processus de récupération de ces informations est présenté dans la section 6. Les informations géométriques exprimées dans le système de coordonnées de S sont traduites dans le système de coordonnées de l'espace logique de A au moyen de la transformation affine inverse que mémorise l'IAMAppEffectiveInteractorsManager de A. L'IAMEvent exprimé dans les termes de S est maintenant exprimé dans les termes de l'espace logique de A. Il est transmis à I via la connexion établie entre le proxy de A et celui de S. A peut alors procéder aux traitements (logiques) attachés à cet événement, et ainsi de suite.

VII.4.4. INSTRUMENTSMANAGER

Il existe un exemplaire d'InstrumentsManager par plate-forme P. Le rôle d'un InstrumentsManager est triple :

1) Normaliser les instruments de façon à gérer la diversité et masquer les détails physiques de fonctionnement ;

2) Gérer les instruments locaux à P : les identifier, découvrir leur type, détecter leur présence et disparition. Certains instruments comme la souris et le clavier sont natifs au système hôte. D'autres,

comme les jetons en plastique de la TableMagique [Bérard 2003], ne le sont pas et nécessitent l'existence d'un processus de traitement externe au système hôte. Il faut donc que l'InstrumentsManager accommode à la fois les instruments natifs comme les instruments exotiques ;

3) Rendre observable l'état des instruments. Par exemple, lorsqu'une souris physique est déplacée, son pointeur graphique, qui est sa représentation observable dans l'espace interactif, doit l'être aussi ;

4) Redigirer les événements normalisés vers le propriétaire concerné.

Normalisation des instruments

Tout instrument est normalisé par abstraction. Il lui est associé une sous-classe d'IAMInstrument qui fournit une représentation abstraite des instruments qu'ils soient natifs ou non.

Au-delà des attributs classiques d'instrument (par exemple, son type), un instrument a un attribut utilisateur, qui, par défaut, prend pour valeur le nom de l'utilisateur logué sur la machine hôte de l'instrument. Bien que nous ne considérions pas le côté collaboratif, nous offrons des mécanismes pouvant servir cette finalité. Par exemple, dans le service de couplage, un instrument peut être verrouillé pour un couplage avec un instrument appartenant à un autre utilisateur.

Gestion des instruments : identification, découverte

Pour satisfaire le double requis de connexion dynamique d'instruments natifs et d'instruments exotiques, nous proposons une solution unifiée avec la technique des proxys. Tout instrument, qu'il soit natif ou non, est représenté par un proxy d'instrument.

- Un proxy d'instrument exotique gère les communications avec le processus de traitement externe au système (par exemple, le système de détection, de suivi de jetons). Ce processus peut éventuellement s'exécuter sur une machine distincte. Du côté plate-forme, le proxy communique avec une abstraction d'instrument exotique, sous-classe d'IAMInstrument.
- Un proxy d'instrument natif communique avec une abstraction d'instrument natif, sous-classe d'IAMInstrument. Dans la mise en œuvre actuelle, les instruments natifs sont de deux types : les instruments de saisie de texte (InputTextInstrument) et les instruments de pointage. Les instruments de pointage incluent deux sous-classes : les instruments de pointage relatifs (RelativePointerInstrument) fournissent une information relative de positionnement (par exemple : la souris a bougé de dx, dy pixels) ; les instruments de pointage absolus comme le pointeur laser, les jetons de Bérard [Bérard 2003] et les stylos pour tablettes PC fournissent une position dans l'espace de coordonnées d'une surface. Pour l'instant, parmi ces deux types,

seule la classe des instruments relatifs est implémentée.

En cohérence avec nos hypothèses (toute plate-forme est supposée équipée d'un instrument de pointage et d'un instrument de saisie de texte), sont créés sur chaque plate-forme P, un exemplaire de RelativePointerInstrument pour gérer un instrument de pointage relatif, et un exemplaire d'InputTextInstrument pour gérer un instrument de saisie de texte. Ces créations sont réalisées par l'IAMWindowManager de P puisque, recevant les événements natifs de la plate-forme (cf section 4.3), il est capable de signaler tout changement à propos des instruments natifs. L'InstrumentsManager crée, pour chaque IAMInstrument, son proxy avec lequel cet IAMInstrument communiquera désormais.

Observabilité de l'état des instruments

L'état des instruments est rendu observable au moyen d'interacteurs logiques spécialisés, les IAMInstrumentInteractors. Ceux-ci sont créés par les proxys d'instrument. Mais comme tout interacteur logique doit appartenir à une application, il existe par plate-forme une application particulière dont l'espace logique contient uniquement des interacteurs de représentation d'instrument. Puisqu'un instrument (comme la souris) doit pouvoir naviguer dans tout l'espace interactif, ce type d'application (il en existe une par plate-forme) exprime son intérêt pour toutes les ressources d'interaction de l'espace.

Un proxy d'instrument acquiert les caractéristiques de son IAMInstrument et selon le type de cet instrument (pointeur relatif ou saisie de texte), créé un IAMPointerInstrumentInteractor ou un IAMInputTextInstrumentInteractor, sous-classes d'IAMInstrumentInteractor. Dès lors, les instruments peuvent être utilisés pour agir sur tout l'espace interactif puisqu'ils bénéficient des propriétés de migration et de distribution des interacteurs logiques. C'est ainsi que la souris locale à une plate-forme P1 peut voir son pointeur graphique affiché sur la surface d'une plate-forme P2. Et si l'espace interactif comporte plusieurs souris, à chaque souris est associé un pointeur, et ceci dynamiquement selon les arrivée et départ de plate-forme hôte.

On connaît la nécessité de visualiser la position du pointeur graphique sur un écran. Avec notre technique, nous pouvons représenter l'état de tout instrument, y compris celui des claviers, ce qui est inhabituel dans les gestionnaires graphiques classiques. L'observabilité de l'état des instruments permet aussi d'exprimer auprès de l'utilisateur les possibilités de couplage. Par défaut, dans I-AM, la souris et le clavier locaux sont couplés. Sur le plan technique, cela signifie que les événements de ce clavier sont transmis à l'interacteur logique sélectionné par cette souris. Ce comportement peut être modifié par programmation afin de créer, si le besoin s'en fait sentir, de nouvelles fonctions de couplage.

Redirection des événements

La redirection des IAMEvents comprend deux volets :

1) Rendre observable l'état de l'instrument, cause de l'IAMEvent ;

2) Rediriger, s'il y a lieu, l'IAMEvent vers l'Interacteur Effectif d'application concerné par cet événement.

Pour illustrer ce processus en deux étapes, prenons l'exemple du déplacement d'une souris locale à la plate-forme P alors que cette souris visite un bouton B appartenant à une application A cliente de la surface S de P. Dans l'espace logique de A, B est un interacteur logique IB concrétisé par un ensemble d'Interacteurs Effectifs IEB (à raison d'un IE par surface utilisée par A). Comme nous l'avons expliqué en 4.3, l'événement souris natif est acquis par l'IAMWindowManager de P, transféré à l'IAMRelativePointerInstrument local, transformé en IAMEvent et transmis à son IAMRelativePointerInstrumentInteractor via le proxy d'instrument qui lui est attaché. Connaissant l'identité de cet interacteur logique, on connaît ses interacteurs effectifs et la fonction de projection. Sa représentation effective dans l'espace interactif est actualisée (le détail est présenté dans la section 5). L'utilisateur voit la souris bouger sur S. L'étape 1 est achevée.

Pour l'étape 2, le « dx,dy » de l'IAMEvent permet à l'IAMWindowManager de P (qui maintient en permanence la position du pointeur) de calculer la nouvelle position Pos du pointeur sur S. Connaissant Pos, l'ensemble des IAMAppEffectiveInteractorsManagers de S permet de retrouver, s'il existe (et c'est le cas dans notre exemple), l'Interacteur Effectif IEB dont l'empreinte contient le point Pos. L'IAMEvent est transmis à IEB, et de là, à IB.

VII.4.5. SYNTHÈSE SUR LE NIVEAU PLATE-FORME D'I-AM

Le niveau Plate-forme cache l'hétérogénéité des machines d'un espace interactif. Elle fournit au niveau supérieur d'I-AM (IAMApp) un niveau logique qui :

- normalise la vue de l'ensemble des ressources d'interaction utilisables sur chacune des machines,
- définit la manière de communiquer avec les surfaces,
- permet de créer facilement de nouveaux types d'instrument utilisables dans un cadre multisurface,
- gère les couplages instrument-instrument (notamment clavier-surface) et instrument-surface (notamment souris-surface).

Alors que la couche Plate-forme est centrée machine, les autres couches sont centrées application. Nous commençons par détailler la couche Application réalisée par le paquetage IAMApp.

VII.5. Le niveau application d'I-AM (paquetage IAMApp)

La glue entre les machines qui exécutent chacune un IAMPlatformManager est réalisée par le paquetage IAMApp. Toute application cliente du niveau plate-forme est un exemplaire d'IAMApp.

Une IAMApp A doit assurer les fonctions suivantes :

1) Découverte des ressources d'interaction de l'espace interactif et notamment des surfaces ;

2) Mise en place de la glue entre les surfaces utilisées pour donner l'illusion d'un espace unifié. Pour cela, il faut que A :
- modélise les relations spatiales entre les surfaces ;
- définisse la projection de son espace logique sur l'espace physique en tenant compte des relations spatiales et caractéristiques physiques (résolution, notamment) des surfaces utilisées ;

3) Gestion de la distribution au niveau pixel de l'IHM graphique sur les surfaces de l'espace interactif.

Plus précisément, la création d'une instance A d'IAMApp a les effets suivants :

- Allocation à A d'un identifiant unique grâce à l'IDManager de la machine où A est créée ;
- Création d'une instance de PhysicalTopologyManager pour acquérir et maintenir un modèle de l'espace physique utilisé par A (requis 1 et 2.a) ;
- Création d'une instance de LogicalTopologyManager qui définit la politique de projection de A (requis 2.b);
- Création d'un Mapper pour établir la correspondance entre les interacteurs logiques de A et leurs interacteurs effectifs en sorte d'assurer la distribution au niveau pixel de l'IHM graphique de A (requis 3).

Le diagramme de classe de la figure 58 illustre les associations entre les principales classes du paquetage IAMApp. Nous présentons maintenant en détail le fonctionnement des trois composants clés : le PhysicalTopologyManager, le LogicalTopologyManager et le Mapper. Un exemple illustre la discussion.

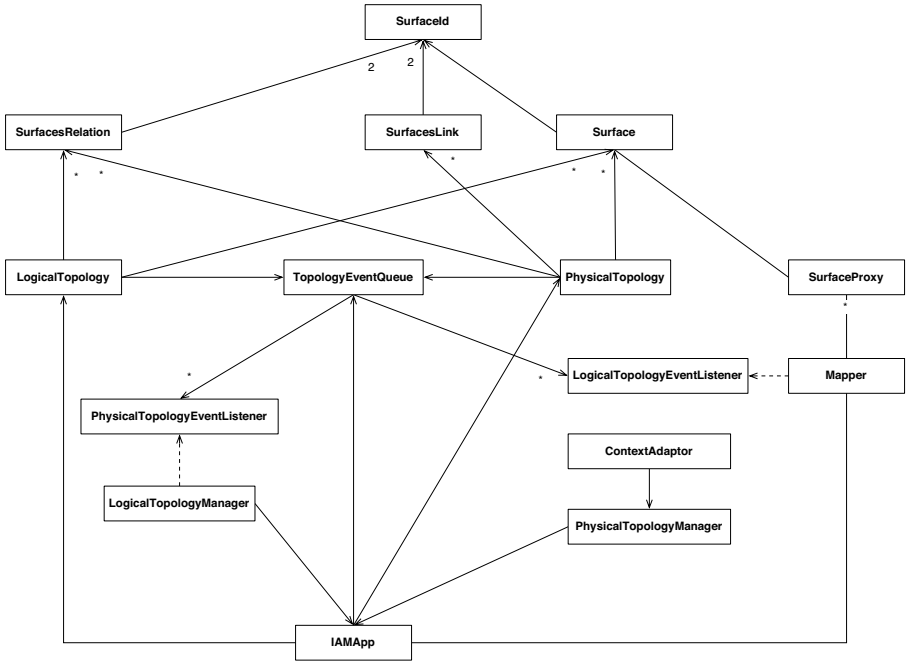

Figure 58. Diagramme de classes simplifié du paquetage IAMApp.

VII.5.1. PHYSICAL TOPOLOGYMANAGER

Découverte des surfaces et modélisation des relations spatiales physiques

Principe À chaque application A, correspond un PhysicalTopologyManager. Ce gestionnaire mémorise dans la structure de données PhysicalTopology, l'existence des surfaces que A exploite ainsi que leurs relations spatiales dans le monde physique.

Pour alimenter et maintenir cette PhysicalTopology, le PhysicalTopologyManager de A crée un ContextAdaptor auquel il s'abonne. Le gestionnaire exprime son intérêt pour tel et tel type de surface, voire toutes les surfaces de l'espace interactif, et/ou toute modification de leur état et de leurs relations spatiales. Cette expression d'abonnement est captée par le ContextAdaptor qui diffuse la demande dans l'infrastructure des contexteurs. La demande est servie par l'ensemble des SurfacesContextor actifs de l'espace interactif (voir section 4.2). Par exemple, le changement de résolution d'une surface S est détecté par le SurfaceSensor de S, transmis au SurfacesContextor de la plate-forme hôte de S (un SurfacesContextor de plate-forme, on le rappelle, est abonné à tous les SurfaceSensor de la plate-forme), puis acquis par le ContextAdaptor de A qui a exprimé son intérêt pour ce type d'information.

Tout changement sur les surfaces est donc automatiquement signalé au ContextAdaptor de A qui est en permanence à l'écoute de l'infrastructure des contexteurs. Cette infrastructure offre plusieurs modes de notification. Nous avons choisi le mode « avertissement sur changement des données d'intérêt» avec une fréquence de sondage de 500ms. Le mode et ses paramètres sont spécifiables dans un fichier de configuration pour chaque classe de contexteurs.

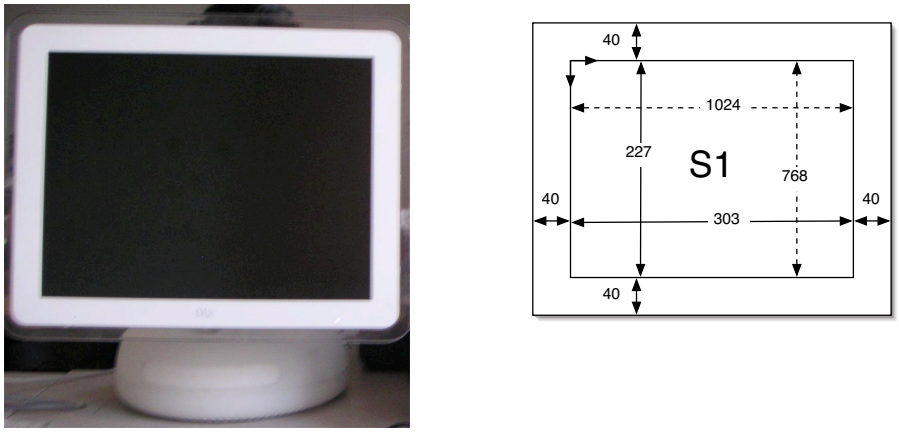

Figure 59. Caractéristiques de la surface S1.

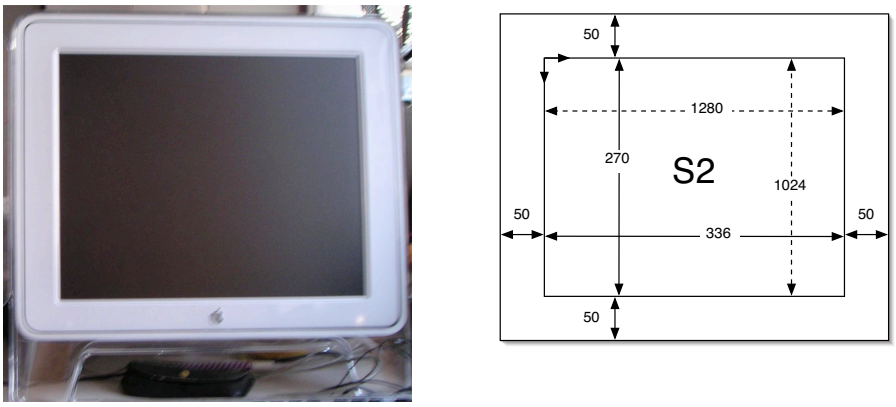

Figure 60. Caractéristiques de la surface S2.

À la réception d'une donnée contextuelle, le ContextAdaptor de A la convertit au bon format (un objet Surface s'il agit d'une information concernant une surface, un objet SurfacesLink s'il s'agit d'une liaison entre deux surfaces) et transmet l'objet à son abonné : le PhysicalTopologyManager de A.

Lorsque le PhysicalTopologyManager de A est prévenu d'un changement à propos d'une surface, il vérifie qu'il ne la connaît pas déjà. Si

c'est le cas, il met à jour les informations sur S qui ont changé. Si la surface est inconnue :

- le PhysicalTopologyManager demande l'ouverture d'un canal de communication entre A et S. Techniquement, le canal de communication est une socket TCP/IP avec la plate-forme hôte de S sur le port dont l'identité est publiée en même temps que les caractéristiques physiques de S. Ce canal, on l'a vu, est géré du côté de A par un SurfaceProxy et du côté de S par un IAMAppProxy. Il est utilisé dans les deux sens : de A vers S pour les actions de A sur S et de S vers A pour permettre à A d'observer S et notamment les actions utilisateur (IAMEvent) qui s'y rapportent.
- Si la connexion réussie, l'objet Surface est ajoutée à la PhysicalTopology avec ses caractéristiques. Dans le cas contraire, rien n'est fait. Prenons comme exemple les caractéristiques physiques correspondant aux écrans de la figure 59 pour S1 et de la figure 60 pour S2.
  - Pour la surface de gauche S1 : une résolution en pixels de 1024 par 768, une résolution de 303 par 227 en mm et des bords de 40 millimètres d'épaisseur.
  - Pour la surface de droite S2 : une résolution en pixels de 1280 par 1024, une résolution en millimètres de 336 par 270, des bords de 50 millimètres d'épaisseur.

Voyons maintenant comment les relations spatiales entre surfaces sont prises en compte.

Découverte et maintenance des relations spatiales entre surfaces

Toute relation spatiale entre deux surfaces S1 et S2 est représentée par une instance de SurfacesRelation. Une SurfacesRelation exprime la transformation affine entre les repères de S1 et S2. On rappelle que dans le cas général, cette transformation est une composition de translation, rotation, et mise à l'échelle (voir chapitre 5). Le calcul de la transformation entre les repères de S1 et S2 résulte de la détection, par le ContextAdaptor de A, d'une liaison entre S1 et S2.

Une liaison a lieu lorsque deux points de liaison, l'un appartenant à S1, l'autre à S2, sont mis en correspondance, par exemple placés l'un contre l'autre. Tout point de liaison de surface est encapsulé dans un SurfacesLinkContextor. Lorsque le SurfacesLinkContextor du point de liaison P1 de S1 « voit » le point de liaison P2 de S2, il diffuse le fait à ses abonnés (ici le ContextAdaptor de A). Si de plus, dans la même fenêtre temporelle, le SurfacesLinkContextor du point de liaison P2 de S2 diffuse le fait qu'il « voit » le point de liaison P1 de S1, le ContextAdaptor en déduit qu'il existe une liaison entre S1 et S2 par les points P1 et P2. Chaque surface connaissant la localisation de ses points de liaison dans son repère, il est possible de calculer les translations et rotations entre les repères de S1 et S2. La connaissance de la résolution de chaque surface permet de compléter la transformation affine avec la

mise à l'échelle, si ces résolutions sont différentes. Une liaison est monodirectionnelle pour davantage de souplesse dans les techniques d'interaction à venir (par exemple, rapprocher S1 de S2 n'a peut-être pas la même sémantique que rapprocher S2 de S1).

Prenons un exemple. La figure 61 illustre la situation où une surface S1 est placée à gauche d'une surface S2. Dans cet exemple (voir figure 62), il existe une liaison L1 entre un point de liaison source P1 situé dans le repère de S1 et un point de liaison cible P2 défini dans le repère de S2. La liaison L1 signifie " le point P1 (343, 30) de S1 correspond au point P2 (-50, 60) de S2".

Figure 61. L'écran de gauche représente la surface S1 et celui de droite la surface S2.

Lorsque le PhysicalTopologyManager de A prend connaissance de l'existence de L1 par l'intermédiaire du ContextAdaptor de A, l'existence d'une SurfacesRelation par la liaison L1 est vérifiée (plusieurs liaisons peuvent correspondre à une même relation spatiale). Si le PhysicalTopologyManager ne trouve pas de relation spatiale associée à L1 dans sa PhysicalTopology, il ajoute à la fois L1 et la SurfacesRelation. Si la SurfacesRelation existe, il ajoute L1 si celle-ci était inconnue. Inversement, à la réception d'une notification de destruction de L1, le PhysicalTopologyManager élimine, si elle existe, L1 de la PhysicalTopology. Si, de plus, la SurfacesRelation qui était associée à L1 n'a plus de liaison, elle est également détruite.

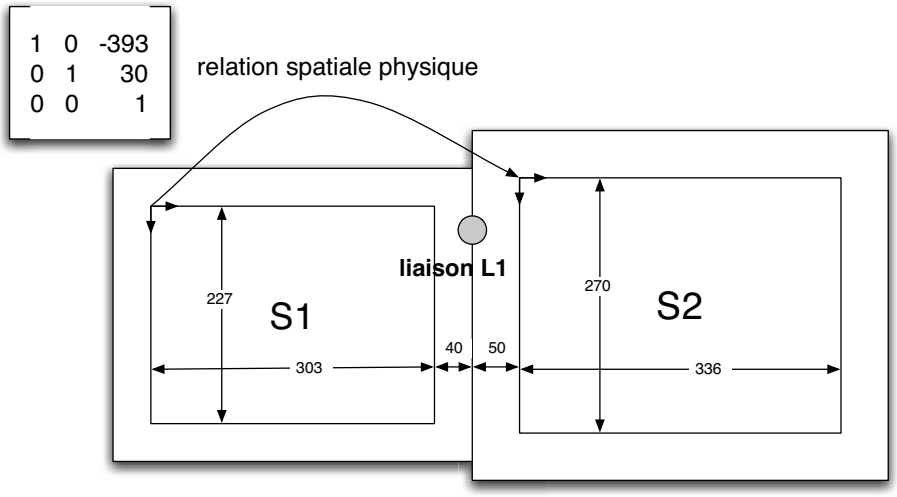

Figure 62. La configuration des surfaces S1 et S2 est représentée en interne dans I-AM sous la forme d'une matrice.

Dans notre exemple, la SurfacesRelation, c.-à-d. la transformation pour passer du repère de S1 au repère de S2 s'exprime par la matrice : [1, 0, -393][0, 1, 30][0, 0, 1] (voir figure 62). Cette matrice représente une translation de (-393, 30) millimètres. Une liaison peut aussi exprimer une rotation. Par exemple, la mise en contact de deux surfaces (écrans) « haut » contre « haut » correspond à une rotation de 180 degrés et « droit » contre « bas » à une rotation de 270 degrés. Ainsi, avec la notion de liaison, I-AM gère un espace planaire en deux dimensions ou en 2D 1/2 (comme PointRight) mais (contrairement à PointRight) avec des rotations possibles de 0, 90, 180 ou 270 degrés.

Les informations de liaison peuvent être fournies par un objet, un composant ou une application faisant partie ou non de I-AM. Cette fonctionnalité, rendue possible par l'infrastructure des contexteurs indépendante d'I-AM, permet la réutilisation de technologies existantes. En effet, les contexteurs de liaison peuvent encapsuler aussi bien des capteurs physiques que logiques. L'infrastructure de contexteurs suggère seulement à ses clients de créer un ContextAdaptor qui sert de pont logiciel entre l'infrastructure et le logiciel client. Pour le moment, nous utilisons des capteurs logiciels fournis par une application dédiée : le configurateur de surfaces (SurfacesTopologyConfigurator). Ce configurateur est censé offrir aux utilisateurs les moyens de configurer leur espace interactif par couplage de surfaces. On trouvera en annexe, un prototype appelé SurfacesConfigurator qui utilise des capteurs IrDa.

Nous savons maintenant comment passer d'un repère de surface à un autre pour maintenir l'illusion d'un espace continu. Si la projection de l'espace logique devait toujours se calquer strictement sur l'espace

physique, les informations retenues dans la PhysicalTopology de A suffiraient. Or, nous voulons donner à A la possibilité de définir sa propre exploitation/interprétation de l'espace physique. D'où la nécessité d'un niveau de traitement supplémentaire : le LogicalTopologyManager.

VII.5.2. LogicalTopologyManager

Projection de l'espace logique

VII.5.3. Principes

Chaque IAMApp A dispose d'un LogicalTopologyManager qui gère une LogicalTopology. La LogicalTopology est une structure semblable à la PhysicalTopology sauf que les relations spatiales entre les surfaces sont maintenant de nature logique. Ce faisant, le LogicalTopologyManager détient la politique de projection de l'espace logique de A sur l'espace physique.

L'espace physique, on le rappelle est construit par couplage de surfaces. Dans I-AM, un couplage/découplage de surfaces signifie un agrandissement/réduction de l'espace d'affichage. Il s'appuie, on l'a vu, sur les relations spatiales des surfaces dans le monde physique. De plus, le couplage/découplage est transitif : si S1 et S2 sont couplées, et si S2 est couplée à S3, alors S1 est couplée à S3. Ce choix vient de notre objectif directeur de fournir un espace d'interaction continu (toutes les ressources semblent être gérées par une seule machine).

Alors que les gestionnaires graphiques usuels appliquent la même fonction de projection à toutes les applications clientes, I-AM permet à chaque application de définir sa propre façon d'exploiter l'espace physique. Mais par souci d'offrir un certain confort au développeur, I-AM propose deux classes de LogicalTopologyManagers offrant chacun une vision du monde :

- une vision « telle quelle » qui maintient une correspondance bi-univoque entre les espaces logique et physique.
- une vision « sans bord » qui ignore l'existence d'espace entre les surfaces de même que leur cadre.

Au chapitre 5 sur la projection d'espaces numériques, nous avons étudié les effets de discontinuité visuelle selon les techniques de projection. Dans la vision « telle quelle », l'information numérique est projetée sur les surfaces mais aussi dans les espaces qui les séparent et sous les cadres. Dans la vision « sans bord », il n'y a pas d'information numérique perdue, mais les lignes droites paraissent brisées.

Le premier travail du LogicalTopology de A est de positionner l'origine du repère de son espace logique dans le repère de l'espace physique. Pour cela, il convient de définir une surface de référence.

Surface de référence

Dans un ensemble de surfaces couplées, on appelle surface de référence pour une IAMApp A, la surface dont le système de coordonnées est utilisé pour localiser l'origine du système de coordonnées de l'espace logique. Par convention, l'origine de l'espace logique d'une IAMApp A correspond à l'origine en millimètres et en pixels de sa surface de référence. Par défaut, la surface de référence est, quand elle existe, la surface de la machine où A est lancée. Ces choix par défaut peuvent être dynamiquement surchargés par programme, ce qui permet

- de choisir pour surface de référence, une surface distante s'il n'existe pas de surface locale comme c'est le cas du Personnal Server [Want et al 2002] ;
- de modifier dynamiquement le positionnement de la surface de référence par rapport au repère de l'espace logique donnant ainsi l'impression d'un espace qui défile sous la surface de référence comme dans le Peephole Display [Yee 2003].

Si l'on reprend l'exemple de la figure 61, une application qui s'exécute sur la machine de gauche affiche, par défaut, son IHM sur S1 alors qu'une application qui s'exécuterait sur la machine de droite afficherait, par défaut, son IHM sur S2. La surface S2, inversement la surface S1, vient étendre l'espace d'affichage de l'application lorsque S1 et S2 sont couplées. La surface S1 dite Sref, de taille Wrefmm et Hrefmm en millimètres, visualise tous les pixels situés dans le rectangle défini par le coin haut gauche (0, 0) et le coin bas droite (Wrefmm * lppmm, Hrefmm * lppmm). Ici, la valeur lppmm définit la taille du pixel logique en millimètres, c'est-à-dire la taille du pixel normalisé des espaces logiques quelle que soit la résolution des surfaces physiques. Nous reviendrons sur la notion de pixel logique.

Disposant d'une surface de référence, le LogicalTopologyManager de A est en mesure de construire et maintenir la LogicalTopology de A à partir du modèle du monde physique : la PhysicalTopology de A.

Construction et maintenance de la LogicalTopology

Le LogicalTopologyManager de A doit être à l'écoute des modifications de la PhysicalTopology de A. Il est donc un listener qui s'abonne à toute information de changement de la PhysicalTopology pour le répercuter dans la LogicalTopology.

Analysons le processus au moyen de l'exemple. On suppose que l'IAMApp A vient d'être lancée sur la machine de gauche. Ses PhysicalTopology et LogicalTopology sont vides. Son LogicalTopologyManager est ensuite prévenu, en raison du changement d'état de la PhysicalTopology, de l'arrivée de la surface S1, puis de S2 et enfin de la liaison L1.

- Apparition de S1. Puisque la LogicalTopology de A est vide, A n'a pas de surface de référence. Le LogicalTopologyManager regarde si

S1 peut jouer ce rôle (il suffit pour cela que l'identifiant de S1 ait le même champ IP que A). S1 est ajoutée dans la LogicalTopology ainsi qu'une SurfacesRelation (logique) entre S1 et S1 qui exprime une transformation Identité : le repère de l'espace logique de A se superpose au repère d'affichage de S1.

- Apparition de S2. Le gestionnaire ne connaît pas de relation spatiale (logique) entre S1 et S2. Il ne peut donc étendre l'espace d'affichage logique avec S2. Le programmeur pourrait néanmoins définir une relation logique sans fondement physique.
- Apparition d'une relation entre S1 et S2 (dûe à la liaison L1). S2 est ajoutée à la LogicalTopology ainsi que la SurfacesRelation (logique) entre S1 (surface de référence) et S2. Cette SurfacesRelation, qui est logique, dénote la partie de l'espace logique que S2 vient de rendre visible. Et la transformation affine qu'elle représente s'appuie sur la transformation affine entre les repères des surfaces (c.-à-d. sur la SurfacesRelation enregistrée dans la PhysicalTopology) combinée à la politique de projection de A (par exemple vision « sans bord »). Ce point est détaillé plus loin.

Si, par la suite, la relation spatiale physique entre S1 et S2 venait à disparaître, le LogicalTopologyManager en serait notifié et la SurfacesRelation (logique) entre S1 et S2 serait détruite. Ce découplage pourrait entraîner le rapatriement sur S1 des composants de l'IHM de A visibles sur S2. Une telle politique, spécifique à chaque application, est à implémenter dans le Mapper de A (voir section 5.3).

Pixel logique Toutes les surfaces n'ont pas la même résolution. Pour s'abstraire de cette hétérogénéité, l'espace logique est un ensemble de pixels logiques. Autrement dit, le pixel logique est l'unité d'affichage d'un espace logique. Il est de type DPI (dot per inch) mesuré en nombre de points logiques par millimètre. Par défaut, le nombre de DPI par mm est fixé à 3,0 (appelé lppmm pour "logical pixel per millimeter"). La taille d'un pixel logique est donc, par défaut, 1/3 mm. Cette valeur peut être modofiée à tout instant par programme.

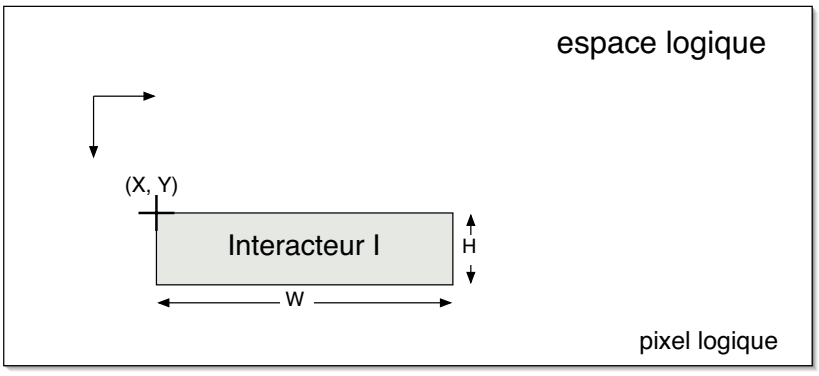

Figure 63. Un interacteur I a été crée dans l'espace logique.

Prenons un exemple. Supposons, comme le montre la figure 63, que l'IHM de A soit constituée d'un IAMinteracteur I (par exemple, la fenêtre maFenetre) de taille W x H (exprimées en pixels logiques) située en X, Y de l'espace logique. Cet interacteur (qui est logique) doit être projeté sur les surfaces S1 et S2 lorsqu'elles sont couplées. L'interacteur I de taille W x H a une taille effective pour l'utilisateur de W/3 x H/3 mm. Le calcul de sa position en mm sur les écrans par rapport à la surface de référence est donc : Xmm = X/3 et Ymm = Y/3.

Par surcharge de la valeur par défaut de lppmm, tout IAMApp peut dynamiquement décider de la taille de son pixel logique, facilitant ainsi le développement d'interfaces zoomables. Comme le montre la figure 64, en changeant la valeur de lppmm, on modifie l'étendue de la zone visualisée par la surface de référence et donc la taille effective de l'interacteur I. Jouer sur la valeur du pixel logique est une façon de modifier l'affichage. Modifier la relation logique entre la surface de référence et les surfaces qui lui sont couplées en est une autre.

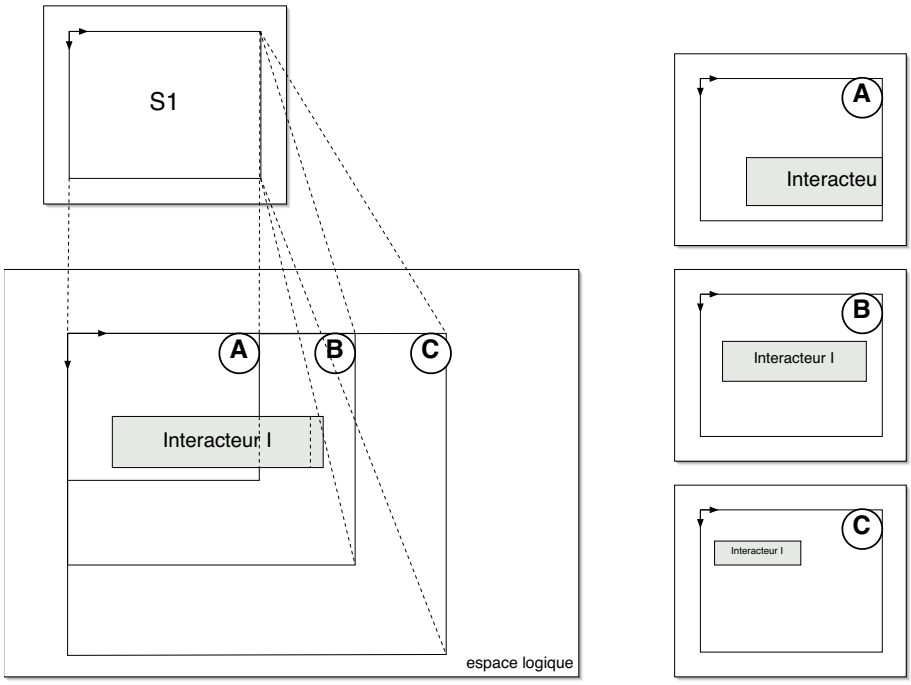

Figure 64. La zone visible de l'interface graphique change suivant la valeur de nombre de pixels logiques par millimètre (lppmm).

Vision « telle quelle » et vision « sans bords »

Par exemple, si pour l'affichage de l'IHM de l'application A, le programmeur choisit la vue « telle quelle », une partie de l'interacteur 1 ne sera pas visualisée car la relation logique entre S1 et S2 tient compte de la taille des bords. La figure 65 illustre cette situation.

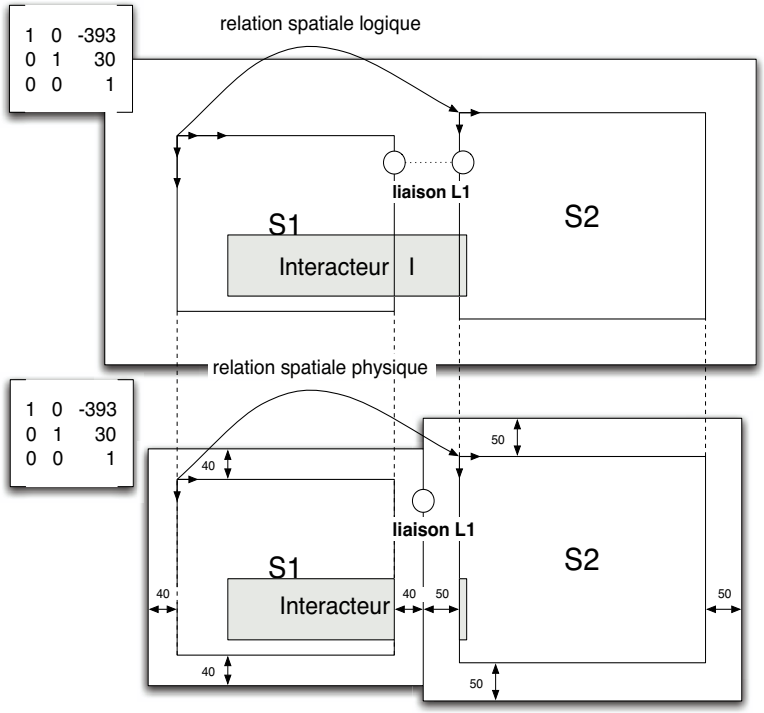

Figure 65. Illustration de l'utilisation de la relation spatiale physique brute pour construire la relation spatiale logique.

A l'inverse, si l'on ne veut pas tenir compte des bords pour que l'interface qui «finit» sur S1 «continue» sur S2, il faut définir une SurfacesRelation logique sans bord et sans espace. Cette relation revient à prendre en compte une liaison virtuelle L1' entre ces deux surfaces du type : «le point P1 situé à droite sur la surface S1 à la coordonnée (303, 30) correspond au point P2 situé à gauche sur la surface S2 à la coordonnée (0, 60)» . La figure 66 illustre ce type de projection de l'espace logique. Dans cette situation, la relation logique est définie par la matrice [1, 0, -303][0, 1, 30][0, 0, 1] qui représente une translation de (-303, 30) millimètres.

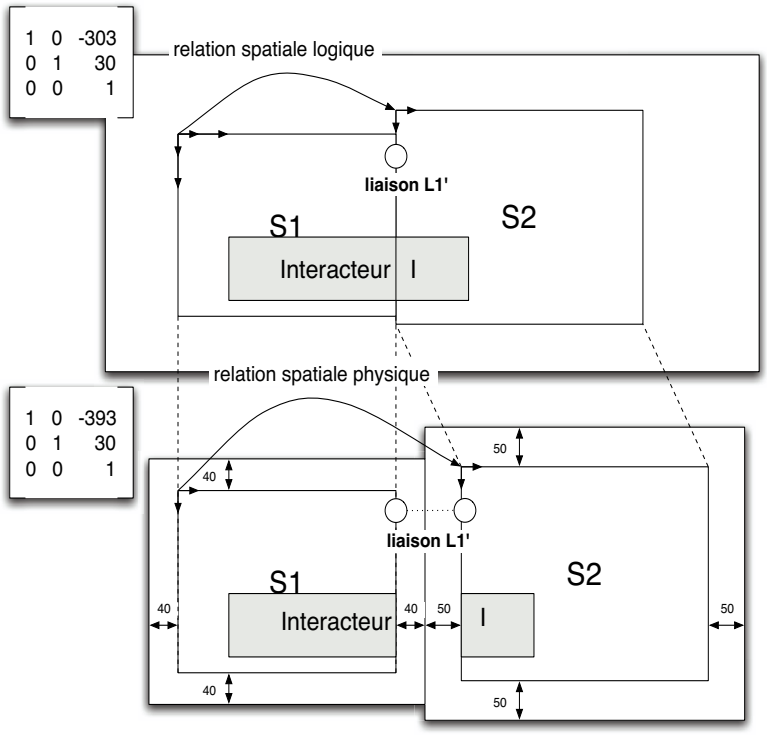

Figure 66. Illustration de l'utilisation de la relation spatiale physique et de la taille des bords des surfaces pour construire la relation spatiale logique.

Grâce à cette SurfacesRelation logique, qui permet de passer de la surface de référence S1 à la surface S2, mais aussi, grâce à la notion de pixel logique et aux caractéristiques de S2, I peut être affiché sur S2 en entretenant l'illusion d'un même interacteur réparti sur deux surfaces. Les composants responsables de cette fonctionnalité sont le Mapper (qui détermine la projection à effectuer sur chaque surface) et le IAMWindowManager (qui produit l'affichage adéquat en termes d'interacteurs effectifs).

VII.5.4. MAPPER D'INTERACTEURS

Il existe un Mapper d'interacteurs par IAMApp. À partir des informations maintenues dans la LogicalTopology de A, ce mapper détermine, pour chaque interacteur logique de A, les interacteurs effectifs correspondants sur les surfaces que A utilise. Il doit intervenir chaque fois que la topologie des surfaces change. Pour cela, il s'abonne aux changements d'état de la LogicalTopology de A (c'est un listener).

À chaque modification a) de SurfacesRelation logique, ou b) du contenu de l'espace logique de A, le Mapper de A envoie à chaque surface S concernée par cette modification, toute information utile à l'affichage des interacteurs effectifs de A sur S, à savoir :

- pour les modifications de type a), la matrice de transformation affine contenue dans la SurfacesRelation (logique) entre la surface de réfé-

rence de A et S et la taille du pixel logique ;

- pour les modifications de type b), la description de l'action sur l'interacteur logique concerné (création d'interacteur logique, etc.).

Ces informations sont envoyées à S via le canal de communication usuel entre A et S et sont traitées par le IAMWindowManager de S. L'IAMWindowManager procède comme suit : chaque interacteur effectif de A est d'abord dessiné comme si ses coordonnées logiques correspondaient à ses coordonnées physiques, puis la transformation affine que le Mapper de A a envoyée à S, est appliquée sur ce dessin pour obtenir le rendu souhaité. La transformation affine est envoyée à S par le Mapper de A à l'arrivée de S dans la LogicalTopology puis actualisée sur toute modification de SurfacesRelation logique dans laquelle S intervient.

Illustrons ce mécanisme avec notre exemple où S1 sert de surface de référence. La position logique (X, Y) de l'interacteur I correspond à la position (X/lppmm, Y/lppmm) en mm dans le repère de S1 et à la position (X/lppmm*1024/303, Y/lppmm*768/227) dans le repère des pixels physiques d'affichage de S1. Par conséquent, la taille de l'interacteur I en pixels physiques sur S1 est égale à W/lppmm*1024/303 pour sa longueur et H/lppmm*768/227 pour sa largeur.

Pour l'affichage de I sur S2, le calcul est analogue à une translation près dûe à la position de S2 par rapport à S1. Dans le cas où toute l'interface doit être visualisée (cas de la figure 66 : vision « sans bord »), la position de I dans le repère de S2 est (X/lppmm - 303, Y/lppmm + 30) en millimètres et sa position en pixels physiques sur S2 est ((X/lppmm - 303)*1280/336, (Y/lppmm + 30)*1024/270). Sa taille en pixels physiques sur S2 est W/lppmm*1280/336 pour sa longueur et H/lppmm*1024/270 pour sa largeur. L'affichage ainsi produit illustre la vision de l'espace unifié que perçoit l'utilisateur (voir figure 67).

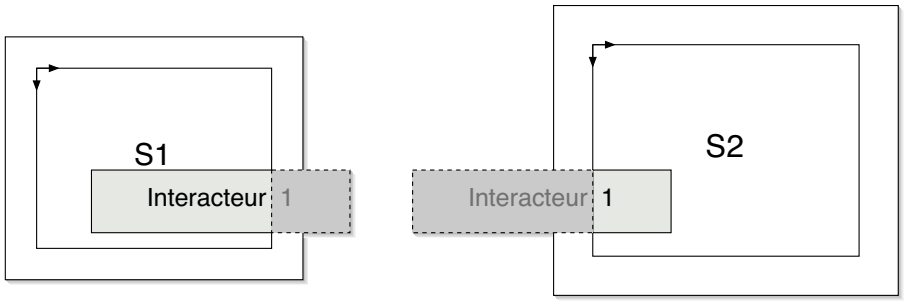

Figure 67. Positionnement de l'interacteur I sur les surfaces S1 et S2 pour que l'utilisateur ait l'impression de l'existence d'un espace unifié construit sur la base d'un couplage de S1 et de S2.

VII.5.5. Synthèse du niveau IAMApp

Le niveau I-AMApp offre l'illusion d'un seul espace d'interaction en offrant des mécanismes de projection du monde logique du développeur sur le monde physique de l'utilisateur. Ce service est fourni pour chaque application qui peut définir sa propre politique d'exploitation de l'espace physique.

Pour chaque application IAMApp A, la correspondance monde logique-monde physique comprend trois aspects complémentaires :

- de l'espace physique à l'espace physique normalisé. Cette étape est assurée par le PhysicalTopologyManager qui obtient les informations sur les surfaces par le biais d'un capteur de contexte, et les maintient dans une structure dédiée : la PhysicalTopology. Cette structure de données modélise, du point de vue de A, le monde physique en termes de surfaces et de leurs relations spatiales ;
- de l'espace physique normalisé à l'espace logique. Cette étape est assurée par le LogicalTopologyManager de A. Ce gestionnaire passe de la PhysicalTopology à la LogicalTopology de A. Cette dernière définit la politique de projection de l'espace logique.
- de l'espace logique à l'espace physique. Cette étape est assurée par un Mapper qui notifie à la couche plate-forme toute modification de l'espace logique (contenu et relations spatiales logiques) et fournit toutes les informations utiles pour que le niveau plate-forme mette à jour le rendu des interacteurs de A.

Le diagramme de séquence UML de la figure 68 récapitule les appels de méthodes et gestions associées qui, ensemble, fournissent l'illusion d'un espace uniforme composé des deux surfaces S1 et S2 sur lesquelles l'interacteur I est affiché.

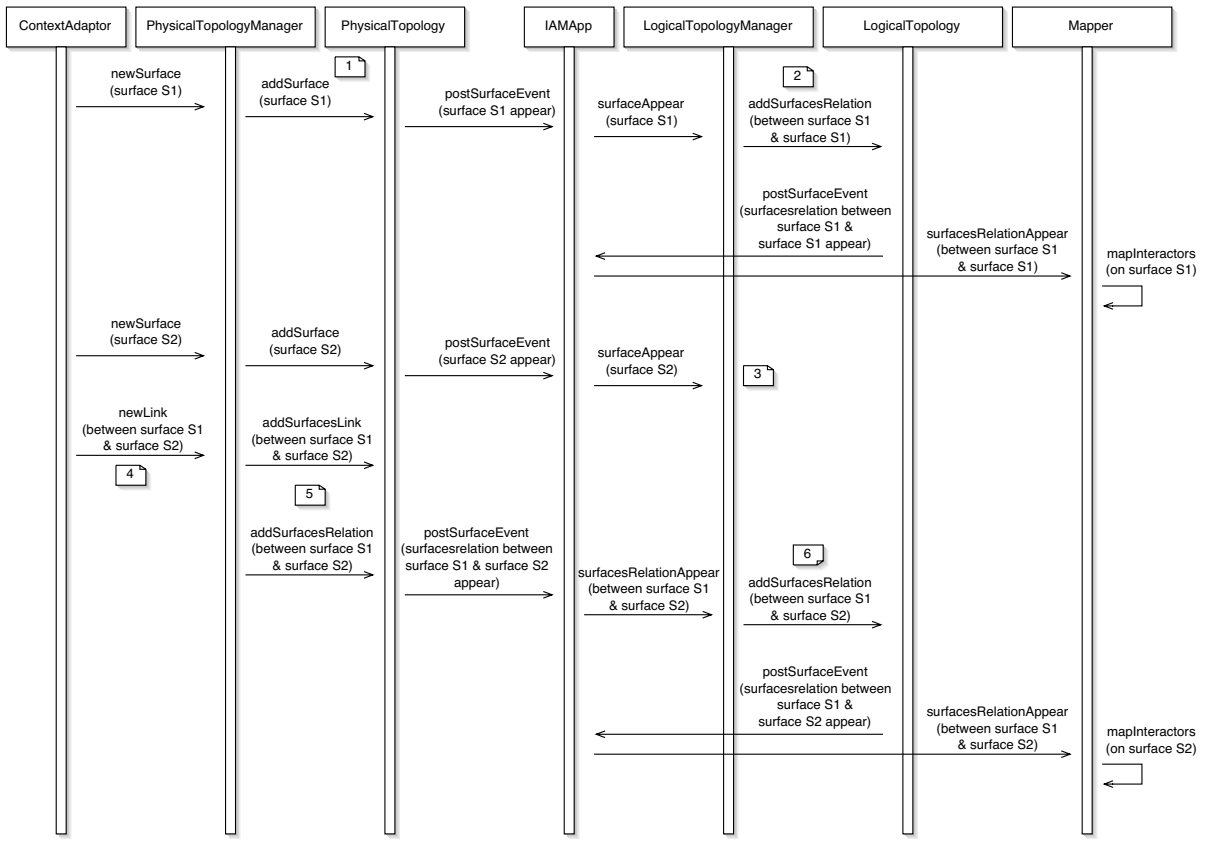

Figure 68. Diagramme de séquence UML illustrant la circulation des messages au sein d'une application IAMApp.

VII.6. Le niveau interacteur d'I-AM (paquetage IAMInteractor)

Le paquetage Interacteur définit la boîte à outils graphique d'I-AM. Un IAMInteractor est un interacteur logique. Du point de vue du programmeur, il a la même fonction que les interacteurs des boîtes à outils graphiques usuelles. Comme eux, il a des attributs de couleur, position, empreinte graphique, etc. Mais contrairement à eux, il n'est pas directement visible. Il est rendu observable par un ensemble d'interacteurs effectifs. En se référant au patron d'architecture MVC, l'interacteur logique définit le Modèle, tandis que les interacteurs effectifs couvrent le Contrôle et la Vue.

Parce que les interacteurs logiques et effectifs vivent dans un milieu dynamique réparti, ils doivent s'autodécrire. Par exemple, dès son arrivée dans l'espace interactif, une surface S doit être avertie de l'état actuel des interacteurs logiques en sorte que les IE correspondants

soient créés en cohérence. Dans les deux sections qui suivent, nous allons montrer de manière très technique, le fonctionnement des interacteurs logiques et effectifs en mettant l'accent sur le maintien de la cohérence des états.

VII.6.1. INTERACTEUR LOGIQUE

Tout interacteur logique hérite de la classe IAMInteractor. Cette classe définit, entre autres, les mécanismes d'introspection d'interacteur.

Soient une IAMApp A, un interacteur logique I de A, IE les interacteurs effectifs de I et LT, la LogicalTopology de A. On suppose que LT n'est pas vide : l'espace interactif dispose d'au moins une surface. La mise en cohérence entre interacteurs logiques et effectifs intervient dans trois cas de figure :

1) Modification de l'état de I : il faut refléter le nouvel état de I à ses IE ;

2) Création de I : il faut refléter l'existence de I à toutes les surfaces selon la politique de projection définie par LT ;

3) Arrivée d'une nouvelle surface : il faut refléter l'existence de tous les interacteurs logiques de A à la nouvelle surface.

Avant de décrire chacune de ces situations, nous présentons le principe de la mise en cohérence commune à toutes ces situations.

Principe

Pour toute action a permettant d'agir sur l'existence d'un interacteur logique I d'une IAMApp A (création ou destruction de I) ou permettant d'agir sur ses attributs (position, couleur, etc.), le processus de mise en cohérence procède comme suit :

1) Réalisation de l'action a sur I dans l'espace logique de A

2) Construction d'un message XML exprimant a

3) Emission de ce message auprès de chaque surface S utilisée par A

4) Pour chaque surface S, traitement du message par le WindowManager de S.

Modification d'attribut d'un interacteur logique

Prenons l'exemple de la fenêtre, maFenetre, instance de la classe IAMWindow, sous-classe d'IAMInteractor. Un programme modifie la position de la fenêtre dans l'espace logique comme suit :

```
maFenetre.setCenterLocation (100, 150);
```

Conformément au principe de mise en cohérence présenté ci-dessus, le corps de cette méthode, héritée de la classe IAMInteractor, est le suivant :

```
public void setCenterLocation(double x, double y) {
   // mise à jour du centre
```

```
    this.center = new Point2D.Double (x,y);
    //construction et envoi de message de modification d'attribut
    setCenterLocationToAction ();
}
```

Détaillons setCenterLocationToAction implémentée dans la classe IAMInteractor :

```
public void setCenterLocationToAction() {
    //Création du message en précisant le type d'action
    String s = beginMethod ("setCenterLocation");
    //Spécification des paramètres de l'action
    s = addParameter (s, "x",""+center.x);
    s = addParameter (s, "y",""+center.y);
    s = endMethod (s);
    //Envoi de l'action
    sendAction (s);
}
```

La méthode beginMethod construit le début du message comprenant le tag <action>, l'identifiant unique de l'interacteur, son type et l'appel de méthode à répercuter sur l'interacteur effectif. addParameter concatène à la suite du message les paramètres additionnels. endMethod termine le message avec le tag <action>. La méthode sendAction d'un IAMInteractor procède à l'appel de la méthode sendAction du Mapper de l'application. Cette méthode parcourt la liste des surfaces S présentes dans la LogicalTopology de A et envoie le message à chacune d'elles.

Au final, l'instruction :

```
maFenetre.setCenterLocation (100, 150);
```

se traduit par la construction du message :

```
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow»
method=«setCenterLocation» x=«100» y=«150»></action>
```

qui est envoyé à chaque surface S utilisée par l'IAMApp 129.87.31.43 via le canal de communication établi entre cette IAMApp et S.

Chaque WindowManager de surface ainsi sollicité :

- retrouve l'IE de I à partir de l'identifiant d'interacteur logique (Cf. le rôle de la structure IAMAppEffectiveInteractors),
- appelle sur IE la méthode correspondant à l'action décrite dans le message (ici, setPosition avec les paramètres 100 et 150 modifiés par la transformation affine que le Mapper de A avait envoyé à

l'apparition de la surface gérée par ce WindowManager.

Création d'un interacteur logique

Reprenons l'exemple de l'interacteur maFenetre créé par une IAMApp comme suit :

```
// Création d'une IAMapp
IAMApp myIamapp = new IAMApp ();
// création d'une fenêtre centrée sur le point (100, 100)
// de taille 300 par 200
IAMWindow maFenetre = new IAMWindow
                    (myIamapp, 100, 100, 300, 200);
```

Le code du constructeur de d'IAMWindow est le suivant :

```
public IAMWindow (IAMApp iamapp, double centerx,
          double centery, double width, double height) {
   // appel du constructeur d'IAMInteractor
   super (iamapp);
   // initialisation des attributs
   this.setCenterLocation (centerx, centery);
   this.setSize (width, height);
   this.setRotation (0);
}
```


Le constructeur de la classe IAMInteractor :

- utilise le paramètre iamapp pour trouver son Mapper M ;
- demande à M de s'occuper de la projection du nouvel interacteur (par appel de la méthode addInteractor sur le Mapper) ;
- appelle constructorToAction qui crée le message XML de création de l'interacteur et l'envoie aux surfaces utilisées par iamapp.

Pour notre exemple, le message de création a le format suivant :

```
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow»
method=«IAMWindow»></action>
```

Il décrit une demande d'action de création (le nom de la méthode est le nom de la classe) d'un interacteur fenêtre (la classe de l'objet est IAM-Window) identifié par 129.87.31.43_4.

Une fois la demande de création faite, le constructeur de la classe IAMWindow initialise les attributs de l'interacteur. Ces initialisations sont des modifications d'attributs qui procèdent comme en 6.1.2. Pour notre exemple, les messages d'action émis par le mapper d'iamapp sont les suivants :

<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«IAMWindow»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setCenterLocation» x=«100» y=«100»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setSize» width=«300» height=«200»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setRotation» rotation=«0»></action>

Arrivée d'une surface S

Ici, l'IAMApp A se met à utiliser S alors que son espace logique est déjà peuplé d'interacteurs. S, nouvelle arrivée, doit construire à la volée les IE en conformité avec l'état actuel des interacteurs I de A.

Pour cela, il convient, pour chaque interacteur I de A de :

- demander une création comme en 6.1.3 (sauf que la création ne doit atteindre que S, non pas toutes les surfaces qui connaissent dejà I) ;
- collecter l'etat actuel de I ;
- envoyer cet état comme en 6.1.2 (sauf que la demande de modification d'attributs ne doit atteindre que S)

Ce processus est défini dans la méthode reconstruct de la classe IAMInteractor. Pour atteindre S de manière sélective, la méthode sendAction d'un interacteur est définie comme suit :


```
protected void sendAction (String action) {
   if (!reconstructMode)
      this.mapper.sendAction (this, action);
   else if (reconstructMode)
      reconstructActions+=action;
}
```

La méthode reconstruct de la classe IAMInteractor est définie comme suit :

```
public String reconstruct () {
   this.reconstructMode = true;
   this.reconstructActions = this.constructorToAction ();
   Method[] methods = getClass().getMethods();
   for (int i=0; i<methods.length;i++){
      Method m = methods[i];
      if (m.getName().endsWith("ToAction") &&
         !m.getName().startsWith("constructor")){
         try{
            m.invoke(this,null);
         } catch(Exception e){
            e.printStackTrace();
```

```
            }
          }
        }
        reconstructMode=false;
        return reconstructActions;
    }
```

La méthode reconstruct commence par indiquer que l'interacteur logique fait l'objet d'un processus de reconstruction. Dans ce mode, le mapper de l'application ne sera pas sollicité et les messages de modification ne seront pas diffusés à toutes les surfaces (Cf. le code de la methode SendAction). On commence par construire la description de l'interacteur en appelant constructorToAction. Puis, grâce au mécanisme d'introspection de Java, on appelle toutes les méthodes suffixées par «ToAction» (exception faite de la méthode constructorToAction). On crée ainsi itérativement la description de l'interacteur dans la chaîne reconstructActions (concaténation effectuée dans la méthode sendAction de l'interacteur). Enfin, pour terminer, l'interacteur est remis dans le mode normal et la chaîne reconstructActions est fournie en retour. Pour l'exemple de notre interacteur mafenetre, cette chaîne est :

<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«IAMWindow»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setCenterLocation» x=«100» y=«100»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setSize» width=«300» height=«200»></action>
<action id=«129.87.31.43_4» class=«IAMInteractor.IAMWindow» method=«setRocation» rotation=«0»></action>

Pour le programmeur d'interacteur, le mécanisme de description d'un interacteur logique et celui de création des interacteurs effectifs correspondants sont transparents car ils sont encapsulés dans les classes IAMInteractor et IAMEffectiveInteractor. Le programmeur doit respecter la décomposition des méthodes set* en deux parties :

- mise à jour des attributs modifiés,
- et appel d'une méthode set*ToAction.

Par ailleurs, pour définir une nouvelle classe d'interacteur, on doit écrire les méthodes set*ToAction mais aussi le code de la classe de l'interacteur effectif correspondant. Comme nous n'utilisons pas de chargeur de classe, le code des interacteurs effectifs doit être présent sur les différentes plates-formes utilisées par I-AM. Aussi, l'une des améliorations possibles de notre travail consiste à utiliser un chargeur de classe et à générer automatiquement le code des interacteurs effec-

tifs en partant du code des interacteurs. Le développement des interacteurs effectifs est facilité par le mécanisme d'introspection de Java.

VII.6.2. INTERACTEUR EFFECTIF

Tout interacteur effectif hérite de la classe IAMEffectiveInteractor qui hérite à son tour des services graphiques de classe JComponent de Java. Dans son implémentation actuelle I-AM propose les classes d'interacteurs effectifs suivantes : IAMEffectiveWindow, IAMEffectivePanel, IAMEffectiveText, et les interacteurs effectifs pour le rendu des curseurs souris et l'état du clavier, les IAMEffectivePointerInstrument et IAMEffectiveInputTextInteractor. Nous verrons le rôle de ces deux derniers dans la section 7.

Une IAMEffectiveWindow est douée de capacité de rotation (d'elle-même et de son contenu), elle possède une barre de titre et des zones d'actions permettant à l'utilisateur de manipuler la fenêtre : changement de taille, mise en rotation, minimisation et maximisation. Toutes ces capacités sont fournies par un de ses composants de classe WMWindow. La méthode paint de la classe WMWindow assure ce service :

```
public void paint(Graphics g){
   Graphics2D g2d = prologue(g);
   super.paint(g2d);
   //appelle paintComponent paintBorder paintChildren
   g = epilogue(g2d);
}

public Graphics2D prologue(Graphics g){
   Graphics2D g2d = (Graphics2D)g;
   g2d.setTransform(transform);
   return g2d;
}

public Graphics epilogue(Graphics2D g2d){
   g2d.setTransform(new AffineTransform());
   //on rend un graphics sans transformation
   return (Graphics)g2d;
}
```

La méthode prologue applique au «graphics» la transformation affine afin que les fils de la fenêtre se dessinent correctement. Les fils se dessinent (corps de leur méthode paintComponent) en utilisant le graphics modifié sans connaître cette modification (rotation, mise à l'échelle, positionnement à l'écran). Après affichage des fils, le graphics est remis d'aplomb pour permettre un affichage correct des autres fenêtres.

Si le programmeur veut afficher une image dans une fenêtre voici le code simplifié de la méthode paintComponent de la classe IAMEffectiveWindow reflétant cette action :

```
public void paintComponent(Graphics g){
   if (img != null)
         g.drawImage(img,0,0,(int)width,(int)height,this);
}
```

On remarque que pour afficher l'image, le programmeur dessine l'image à la position (0, 0) de la fenêtre avec pour taille, la taille de la fenêtre. Il n'y a donc aucune référence à une quelconque modification du Graphics pour le programmeur d'interacteur effectif. Pourtant ce code a pour effet de créer une fenêtre disposée à la bonne rotation et à la bonne échelle.

L'utilisation de WMWindow facilite la programmation d'interacteurs effectifs. En effet, le développeur n'a pas à se soucier des paramètres de la projection des interacteurs. Il doit simplement développer les interacteurs effectifs en supposant que les coordonnées logiques définissent leurs véritables coordonnées dans l'espace effectif. La programmation d'interacteur effectif ressemble donc à la programmation de composant graphique Java usuel à une différence près : il n'est pas possible pour des raisons internes à Java de réutiliser les composants graphiques de base pour cause de problèmes de «repaint abusif avec perte du contexte graphique courant». Il faut réécrire les interacteurs en s'inspirant de l'interface de programmation des interacteurs Swing de Java.

Cette différence entraîne un surcoût de programmation qui explique que nous n'ayons développé que très peu d'interacteurs.

VII.6.3. GESTION DES ÉVÉNEMENTS

Nous avons vu en 4.4:

- que l'InstrumentsManager d'une plate-forme P crée un proxy d'instrument par instrument (que celui-ci soit natif ou exotique) ;
- que ce proxy sert de lien entre :
  - une abstraction d'instrument, un IAMInstrument qui reçoit des IAMEvents (abstractions d'événements natifs) et
  - sa représentation observable, un IAMInstrumentInteractor ;
- que cet IAMInstrumentInteractor est créé par une application particulière qui couvre l'ensemble des surfaces de l'espace interactif.

C'est ainsi qu'une souris se voit représentée par un proxy d'instrument liant une instance de la classe IAMRelativePointerInstrument et une instance de la classe IAMRelativePointerInstrumentInteractor. Il en va de même pour le clavier. Dans cette section, nous décrivons en détail le

traitement des événements souris et clavier au moyen de l'exemple de la figure 69

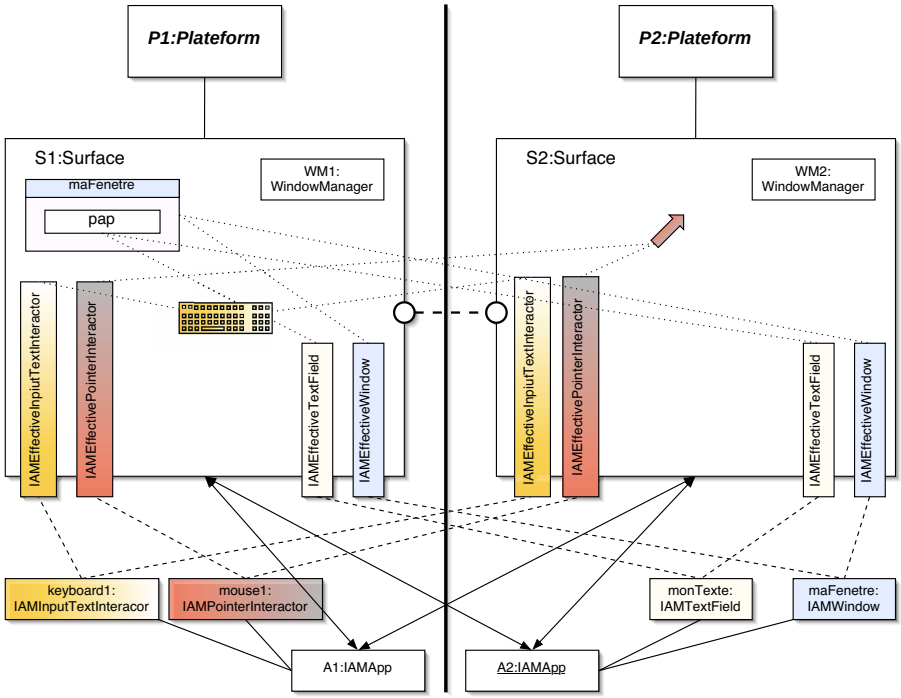

Figure 69. Gestion des évènements dans I-AM.

Deux plates-formes P1 et P2 gèrent les surfaces S1 et S2 respectivement. Ces surfaces sont couplées. Un clavier Keyboard1 et une souris Mouse1 sont connectés à P1. Sur P1 tourne l'application A1 propriétaires des interacteurs logiques représentant l'état de ces instruments. Du fait du couplage de S1 et S2, ces interacteurs logiques ont un IE sur chacune des deux surfaces. Sur P2 tourne l'application A2 qui a créé la fenêtre maFenetre dans laquelle s'affiche le champ texte monTexte. Ce champ texte visualise le texte «pap». Du fait de la liaison entre S1 et S2, cette fenêtre et ce champ texte ont un correspondant effectif sur chaque surface.

Gestion des événements souris

Lorsque l'utilisateur déplace la souris Mouse1 de (dx, dy), le message natif java est abstrait (par le WindowManager de S1) en IAMEvent et remis à l'IAMRelativePointerInstrument de la souris, puis à son proxy sur P1. Le message reçu par le proxy est le suivant :

<action id=«128.78.54.56_12» class=«IAMInstrument.IAMPointerInstrument» method=«moved» dx=«-4» dy=«3»></action>

Le message dénote un déplacement de souris de 4 pixels vers la gauche et de 3 vers le bas dans le repère de la surface de référence de P1 c'est-

à-dire S1. Cet événement est produit par l'instrument de pointage numéro 12 connecté à la plate-forme 128.78.54.56. À la réception de ce message, le proxy effectue auprès de l'interacteur logique de cet instrument l'appel suivant : pointerInteractor.moved(-4, 3);

L'exécution de cette méthode doit conduire aux résultats suivants : mettre à jour les IE correspondants sur S1 et S2 en sorte que l'utilisateur puisse constater le déplacement du curseur, mais envoyer l'événement à une seule des surfaces en sorte que si le curseur se trouve au-dessus d'un interacteur d'application (par exemple une fenêtre), cet interacteur le reçoive une seule fois. Ce qui donne :

```
public void moved (int dx, int dy) {
   setCenterLocation(this.center.x+dx,this.center.y+dy);
   String s = beginMethod("moved");
   s = addAttr(s, "x",""+this.center.x);
   s = addAttr(s, "y",""+this.center.y);
   s = endMethod(s);
  super.sendActionOnce(s);
}
```

La méthode setCenterLocation procède au déplacement de tous les IE correspondant à l'interacteur logique de la souris selon le processus décrit en 6.1. Puis un message XML dénotant un événement «moved» est construit et envoyé par sendActionOnce. Ce message est envoyé par le mapper de A1 à une seule des surfaces de l'espace interactif : la surface de référence de A1, soit S1. Si, par exemple, maFenêtre se trouvait sous le curseur, et si le message de déplacement était envoyé aux deux surfaces S1 et S2, chaque IE de maFenêtre recevrait l'événement et le ferait suivre à maFenetre ce qui pourrait conduire à des incohérences.

Notons que le mapper de A1 pourrait aussi choisir la surface sur laquelle le curseur est visible. Cependant ce choix poserait deux problèmes. Le premier concerne la gestion des événements notamment dans le cas où un événement «pressed» serait reçu sur S1 et l'événement «released» sur S2. Ce cas introduit une incohérence dans l'état des interacteurs effectifs difficilement gérable. Le second problème est que ce calcul introduit une latence non négligeable dans le cas d'un mouvement continu de la souris. Pour éviter ces désagréments, l'événement est donc envoyé à une seule surface : la surface de référence de la plate-forme locale. Cela suppose que les interacteurs existent sur toutes les surfaces de l'espace logique ce qui est heureusement le cas. Cela explique en partie le choix d'une réplication totale des interacteurs sur chaque surface.

En recevant les messages décrivant le déplacement du centre du curseur, chaque surface procède au déplacement du curseur effectif. La surface qui reçoit le message d'action dont le champ method est moved

le donne à son window manager (ici le WindowManager). Ce dernier regarde si un IE se trouve sous la coordonnée décrite par l'événement. Si c'est le cas, il transforme l'événement pour qu'il soit défini dans l'espace de coordonnées de cet IE. Une fois cette transformation effectuée, il donne l'événement à l'IE sous la forme d'une observation. En recevant celui-ci l'interacteur effectif le fait suivre à son interacteur logique. L'interacteur logique le reçoit et traite les callbacks associées à ce type d'événements. Ce message d'observation est de la forme :

<observation id=«128.78.54.56_22» class=«IAMInteractor.IAMEffectiveTextFieldInteractor» source=«128.78.54.56_12» method=«moved» dx=«-3» dy=«2»></observation>

Il s'agit d'un message d'observation car il correspond à l'observation d'une action de l'instrument représenté par l'interacteur 128.78.54.56_12 sur l'interacteur champ texte effectif 128.78.54.56_22. Chaque événement contient l'identifiant de l'interacteur de l'instrument source de l'événement. Par simplification, on peut dire qu'il s'agit de l'identifiant de l'instrument lui-même puisque cet interacteur est l'unique représentant de cet instrument. Il n'y a donc pas d'ambiguité possible.

Dans le cas d'un événement «pressed» ou «released» envoyé par la souris et donné à son interacteur, une seule surface S le reçoit. Il est traité par le WindowManager de S et donné à l'interacteur effectif de S situé sous le curseur qui le transmet à son correspondant logique. Supposons que l'utilisateur clique sur le champ texte monTexte dans maFenetre. L'événement natif est reçu le WindowManager de S1 puis transmis à l'IE de monTexte sur S1 après modification des coordonnées de l'événement. Ainsi, le WindowManager donne le focus à cet interacteur effectif pour cet instrument. Ce dernier stocke cette information et transmet celle-ci à l'interacteur logique monTexte qui se met dans l'état «j'ai le focus» pour l'instrument correspondant à l'interacteur 128.78.54.56_12. Ce dernier calcule, grâce à l'événement, la position du point d'insertion et décide de fournir un feedback en mettant son contour en bleu. Pour cela il appelle les méthodes setBorderColor et setInsertPoint. Ces appels de méthode se répercutent sur tous ses interacteurs effectifs qui changent alors d'état.

I-AM étend le concept de focus en permettant à un même interacteur d'avoir le focus pour des instruments de pointage différents. Chaque interacteur logique gère donc une liste d'instruments pour lesquels il a le focus. Cette liste est vide par défaut. À chaque modification de cette liste, l'interacteur logique envoie un message à tous ses IE afin qu'ils aient connaissance de la nouvelle liste. Ces informations de focus locales aux surfaces sont utiles pour la gestion des événements clavier lorsque des couplages existent entre souris et claviers. Lorsqu'une nouvelle surface S3 apparaît dans l'espace logique, un nouvel interac-

teur effectif sera créé sur celle-ci. Dans les données de création, on retrouvera cette liste de focus.

Gestion des événements clavier

Nous allons décrire la gestion des événements clavier en prenant comme exemple le clavier Keyboard1.

Lorsque l'utilisateur appuie sur une touche de Keyboard1, un message décrivant cet événement est envoyé de cet instrument à son proxy sur P1. Pour le clavier il y a trois types d'événements : «pressed», «released» et «typed». Si l'utilisateur appuie sur la touche «a», afin que le texte «papa» apparaisse dans monTexte, alors le message suivant arrive au proxy :

<action id=«128.78.54.56_13» class=«IAMInstrument.IAMInputTextInstrument» method=«pressed» type=«401» modifiers=«0» keyCode=«65» keyChar=«a»></action>

Ce message décrit le fait que l'instrument 13 de type IAMInputTextInstrument géré par la machine 128.78.54.56 envoie un événement pressed sur la touche "a". Les messages concernant les événements "pressed" et "released" sont construits de la même façon. En recevant ce message le proxy appelle sur l'interacteur correspondant au clavier la méthode pressed avec les paramètres suivants : inputTextInteractor.pressed (401, 0, 65, «a»);

Cet événement doit être envoyé à une seule surface pour les raisons expliquées plus haut. Le corps de la méthode pressed de la classe IAMInputTextInteractor procède à la création d'un message décrivant cette action et à l'envoi de ce message à une seule surface :

```
public void pressed (int type, int modifiers,
   int keyCode, String keyChar) {
      String s = beginMethod("pressed");
      s = addAttr(s, "type",""+type);
      s = addAttr(s, "modifiers",""+modifiers);
      s = addAttr(s, "keyCode",""+keyCode);
      s = addAttr(s, "keyChar",keyChar);
      s = endMethod(s);
      super.sendActionOnce (s);
}
```

Ce message est reçu par une instance de la classe IAMEffectiveInputTextInteractor sur la surface S1. Cet IE se charge de transmettre cet événement au WindowManager de S1. L'événement clavier doit être donné à l'interacteur effectif qui a le focus pour l'instrument de pointage couplé avec le clavier si un tel interacteur existe. Le WindowManager de S1 a donc besoin de connaître les couplages définis entre les

différents instruments. Comme S1 connaît Mouse1 et Keyboard1 car elle affiche ces deux interacteurs effectifs, elle peut questionner Mouse1 pour savoir si elle est couplée avec Keyboard1. Pour pouvoir répondre à cette question Mouse1 doit connaître les différents couplages de la souris de P1.

I-AM gère par défaut un couplage entre les instruments de pointage et de saisie de texte connectés à la même plate-forme. Cette information est stockée dans l'interacteur logique associé à chaque instrument. Ainsi Mouse1 sait qu'il est couplé avec Keyboard1 et vice-versa. Cette information de couplage est donnée à leurs interacteurs effectifs grâce à la méthode giveCouplingToAction. Ainsi dès qu'un interacteur effectif instance de la classe IAMEffectiveInstrumentInteractor est crée il connait ces informations. De plus, il est prévenu quand ces informations de couplage sont mises à jour. Tout ce mécanisme est encapsulé dans la classe IAMInstrumentInteractor dont les classes IAMInputTextInteractor et IAMPointerInteractor sont des spécialisations.

Ainsi, en connaissant l'interacteur effectif de Keyboard1 sur S1, le WindowManager de S1 peut savoir avec quels autres instruments ce clavier est couplé. Il s'aperçoit que Keyboard1 est couplé avec Mouse1. Comme monTexte a le focus pour Mouse1, l'événement est donné à celui-ci par l'intermédiaire de son interacteur effectif sur S1 sous la forme d'une observation. L'interacteur logique monTexte reçoit cette observation, modifie le champ texte en conséquence. Puis, il envoie à ses IE un message d'action demandant l'affichage du texte "papa" et le positionnement du point d'insertion après le quatrième caractère.

VII.6.4. Synthèse du niveau Interacteur d'I-AM

L'exemple nous a permis de voir que les surfaces jouent un rôle central dans I-AM. Sans leur présence, les différentes applications I-AM n'utiliseraient pas pour l'utilisateur le même espace logique unifié. Elles servent d'intermédiaire aux applications et gèrent la cohérence en faisant suivre les actions et les observations aux bons composants. Les événements ne sont pas centralisés dans un serveur comme c'est le cas avec EventHeap [Johanson et al 2002b], ni envoyés à tous les composants par un bus logiciel mais ils sont gérés de manière distribuée.

VII.7. Synthèse

I-AM offre une solution originale au problème du couplage de ressources d'interaction pour la création d'un espace unifié permettant une interaction multi-instrument multisurface. Elle réutilise tous les concepts présentés dans notre ontologie et notamment les trois principales que sont les relations spatiales, le couplage et la projection d'interac-

teurs sur plusieurs surfaces. De plus, I-AM est basée sur les notions d'Observation et d'Action, qui sont l'essence même du rôle de ressource d'interaction présenté au chapitre 2.

Il nous reste à analyser les caractéristiques d'I-AM vis-à-vis des requis du chapitre 6 et à présenter les perspectives de notre travail. Ces aspects font l'objet de notre conclusion.

Chapitre VIII *Conclusion et perspectives*

Notre travail avait pour objectif la réalisation d'un intergiciel capable d'assurer la découverte de ressources de calcul et d'interaction et d'en masquer l'hétérogénéité pour former, du point de vue du développeur, un espace uniforme d'interaction. En réponse, nous proposons I-AM. I-AM permet à un utilisateur de relier plusieurs machines exécutant des systèmes d'exploitation différents (par exemple, MacOS et Windows) et gérant chacune des ressources d'interaction différentes. Du point de vue programmation, ces ressources donnent l'impression d'être gérées par une seule machine : la multiplicité des machines et des systèmes d'exploitation, l'allocation des ressources d'interaction à ces machines, et l'hétérogénéité des machines, des systèmes et des ressources sont transparentes au programmeur, mais accessibles si le besoin s'en fait sentir.

Si les objectifs de cette thèse étaient essentiellement techniques, l'approche adoptée pour aboutir à une solution, s'est appuyée sur les principes d'une interaction où les mondes physiques et numériques fusionnent pour constituer des espaces interactifs. Nous avons montré dans notre ontologie que dans ces espaces, certaines entités physiques servent de ressources d'interaction. Les unes jouent le rôle d'instrument, d'autres celui de surface.

Dans ce dernier chapitre, nous présentons une évaluation d'I-AM au regard des requis logiciels du chapitre 6. Nous présentons ensuite quelques éléments de perspectives à ce travail.

VIII.1. Evaluation d'I-AM

Le chapitre 6 a montré que les solutions logicielles proposées aux problèmes de l'interaction multi-instrument multisurface étaient incomplètes et difficilement intégrables entre elles. Ce constat a motivé la

mise en œuvre d'I-AM présentée de manière très complète au chapitre précédent. Il convient maintenant de faire le bilan de notre réalisation en réutilisant nos critères d'évaluation de la « concurrence ». Nous les rappelons ici : découverte des acteurs et des ressources d'interaction, localisation des entités physiques, modélisation du couplage, migration et distribution d'IHM, architecture distribuée, conformité de la latence, support aux applications patrimoniales, et facilité de programmation.

VIII.1.1. DÉCOUVERTE DES ACTEURS ET RESSOURCES D'INTERACTION

Le logiciel de base qui sous-tend la mise en œuvre d'espaces interactifs doit être capable de découvrir les entités pertinentes. Nous avons vu que cela regroupe trois fonctions :

- Reconnaître,
- Identifier,
- Et détecter la présence, l'arrivée et le départ des acteurs et des ressources d'interaction.

Reconnaissance. I-AM modélise la majorité des attributs et propriétés physiques des surfaces et certaines caractéristiques des instruments, mais ne retient des utilisateurs que leurs identifiants de session. Même si la plupart de ces informations sont tirées de fichiers de configuration, la connaissance de la taille physique des surfaces et de leurs orientations sont des informations inconnues des systèmes concurrents d'I-AM. De lourds efforts de recherche sont encore nécessaires en sorte que tous les objets physiques de notre environnement s'autodécrivent et se découvrent sans intervention humaine.

Identification. Notre infrastructure identifie de manière unique chaque objet d'intérêt. En effet chaque surface, chaque instrument, chaque machine et enfin chaque interacteur a un identifiant unique dans l'espace interactionnel. Ainsi, malgré la distribution totale du système, aucun objet n'est géré en doublon.

Détection de présence. Ce service repose essentiellement sur l'infrastructure des contexteurs qui, par son mécanisme d'abonnement-publication, offre une solution performante au problème de la découverte des ressources d'interaction. L'environnement ainsi découvert par les contexteurs est modélisé dans I-AM par une structure de donnée particulière : la PhysicalTopology (vers chapitre 7). Celle-ci offre une vision unifiée d'un espace d'interaction librement façonnable.

Par ailleurs, nous rappelons que ces services de découverte dénotent la capacité de communiquer, mais ne signifient pas nécessairement la présence physique de l'entité dans l'espace interactif. C'est pourquoi les contexteurs sont aussi utilisés pour véhiculer dans l'infrastructure les informations topologiques (relations spatiales entre entités physiques)

nécessaires à la détermination de la présence tangible ou de l'absence d'une entité.

Ces caractéristiques font d'I-AM une solution plus complète que ses concurrentes. Cependant, nous sommes conscients que de nombreuses améliorations restent à apporter pour offrir une modélisation plus riche du monde réel.

VIII.1.2. LOCALISATION D'ENTITÉS PHYSIQUES

I-AM s'appuie sur une localisation géométrique en 2D 1/2 précise qui permet la distribution d'IHM au niveau du pixel sur plusieurs surfaces tout en limitant les risques de discontinuités visuelles.

L'abstraction géométrique fournie par les contexteurs de liaisons est transformée en structure utilisable pour le couplage de surfaces. Sur ce point, l'annexe 1 donne de plus amples détails sur la localisation et le couplage de surfaces ainsi que sur le rôle des contexteurs de liaisons. On y trouvera également trois prototypes de localisation de surfaces développés dans notre équipe.

Même si notre modélisation est riche, un modèle géométrique plus complexe doit être utilisé si l'on veut intégrer dans I-AM des effecteurs comme l'EveryWhere Display Projector [Pinharez 2001]. Toutefois, un modèle en 3 dimensions posera le problème de point de vue si le système est utilisé par plusieurs utilisateurs. En effet, comment permettre à plusieurs utilisateurs d'utiliser l'information projetée sur des surfaces mobiles orientées de manière quelconque alors qu'ils risquent de ne pas pouvoir lire leur contenu ?

VIII.1.3. MODÉLISATION DU COUPLAGE

Le chapitre 4 a montré que le couplage est un concept incontournable mais mal formalisé. C'est pourquoi avec Barralon, nous imposons comme requis :

- La modélisation explicite du couplage de ressources d'interaction, à raison d'un automate par couplage. En particulier, nous mettons l'accent sur la nécessité de modéliser les couplages instrument-instrument, instrument-surface et surface-surface,
- L'expression explicite auprès de l'utilisateur de l'état courant de chaque automate en sorte que les propriétés de robustesse et de souplesse de l'interaction soient satisfaites.

De ce point de vue, I-AM est de loin la solution la plus complète. Cependant, avec la notion de couplage nous avons mis au jour de nouveaux problèmes que cette seule thèse ne peut suffire à résoudre. En ce qui concerne l'implémentation du couplage, une piste reste à explorer : est-il possible de donner à un seul composant la responsabilité de la

gestion des couplages ? En effet, dans notre solution actuelle chaque type de couplage modélisé est géré par plusieurs composants distincts :

- couplage surface-surface : c'est la responsabilité des LogicalTopologyManagers et des Mappers
- couplage instrument-instrument : les composants responsables sont les InstrumentManagers (qui est couplé avec qui) et les WindowManagers (redirection des événements en fonction de cela)
- couplage instrument-surface : les composants responsables sont les IAMApps des PlatFormManagers (plus précisement leurs LogicalTopologyManagers et leurs Mappers). Ils déterminent sur quelles surfaces les instruments peuvent agir.

Il est possible qu'en l'état le caractère distribué de notre architecture empêche ce regroupement de préoccupations. Cependant, la question mérite d'être étudiée.

VIII.1.4. MIGRATION ET DISTRIBUTION D'IHM

La migration et la distribution de l'IHM sont deux nouveaux services offerts par les IAMInteracteurs. Puisque, pour la projection de l'interface graphique, I-AM tient compte des attributs physiques des surfaces, des relations spatiales, de la description des interacteurs et de la vue logique du programmeur, notre solution offre une migration et une distribution de l'IHM au niveau du pixel. En effet, même avec une rotation, un interacteur peut être placé à cheval sur plusieurs surfaces.

Bien sûr, sans l'utilisation d'une boîte à outils graphique performante offrant la rotation et le facteur d'échelle au niveau du pixel, rien de cela n'est possible. Notre infrastructure étant écrite en Java, nous avons commencé par utiliser la boîte à outil graphique Swing. Cependant, pour des raisons de performances et d'implémentation, nous avons réécrit nos propres composants graphiques à partir d'un canevas openGL. Pour l'instant, cette boîte à outil n'est pas très riche : tout au plus une dizaine d'interacteurs. De ce fait, les IAMInteracteurs ne permettent pas l'écriture de logiciel complexes comme un traitement de texte, mais seulement la création de démonstrateurs de recherche.

Par contre, I-AM innove sur la gestion des instruments et notamment sur la gestion des pointeurs multiples. En effet, la réutilisation en entrée des services de migration et de distribution des interacteurs permet l'utilisation de n'importe quel intrument sur n'importe quelle surface (et donc sur n'importe quel interacteur) quelle que soit sa position dans l'espace unifié.

VIII.1.5. ARCHITECTURE DISTRIBUÉE

I-AM est construite sur une architecture complètement distribuée dans laquelle les contexteurs jouent un rôle de glue. Ainsi, aucune machine n'a de rôle privilégié, ce qui donne une souplesse logicielle en conformité avec la créativité spontanée des humains. Certes, le modèle

XWindow nous a servi de guide tout au long de notre implémentation. Mais, nous avons ajouté de nombreuses fonctionnalités pour atteindre nos objectifs comme la description des interacteurs.

Le chapitre précédent a montré la similitude de certains aspects de notre architecture avec l'achitecture conceptuelle PAC et MVC. Sur ce point il est intéressant de noter que, pour I-AM, les Abstractions et les Présentations sont disséminées sur toutes les machines d'un espace interactif, et que les Mappers et les IAMEffectiveInteractorManagers jouent le rôle de Contrôles. Dans ce modèle d'architecture, les notions d'Action et d'Observation, tirées de notre ontologie, garantissent la cohérence globale du système.

VIII.1.6. CONFORMITÉ DE LA LATENCE

En architecture distribuée, une mesure rigoureuse de la latence est difficile à réaliser. Toutefois, afin d'améliorer les performances de notre système, nous avons testé de manière indépendante les composants clés de notre infrastructure qui, à nos yeux, pouvaient créer une latence interne ou externe. Les points mis en cause sont les suivants :

- L'envoi de messages sur le réseau comme lors de la communication du Mapper M d'une application A avec une surface S et inversement.
- L'encapsulation et l'analyse des messages au format XML.
- La gestion des files d'évènements (les topologylisteners et les files de messages).
- L'affichage des composants graphiques avec application de zooms et de rotations.

De cette analyse, il ressort deux points critiques :

- la boite à outil graphique AWT de Java est la cause de la plus grande partie de la latence,
- et la gestion des files d'évènements dont l'impact reste négligeable sur la latence totale.

Ainsi, notre effort s'est porté sur l'utilisation d'interacteurs peu gourmands en ressources processeur lorsqu'ils subissent des rotations et des mises à l'échelle. Nous avons encapsulé en Java des interacteurs de la table magique [Bérard 2003] basés sur OpenGL. Ainsi, le rendu graphique n'est plus produit par la boite à outil graphique conventionnelle de Java mais par une extension OpenGl qui utilise la carte graphique et non le processeur. Suite à cette modification, la latence perçue est devenue acceptable, confirmant ainsi notre choix d'implémentation. Pour cela, nous avons fait tester notre infrastructure par une dizaine de personnes.

VIII.1.7. RÉUTILISATION DE L'EXISTANT (LEGACY SYSTEMS)

Sur ce point notre constat était le suivant : il n'est intéressant de réutiliser les applications existantes que si elles tirent pleinement profit des opportunités interactionnelles données par l'interaction multi-instrument et multisurface. Il s'agissait donc de trouver un bon compromis entre le nombre de modifications à effectuer sur les applications existantes et le nombre de nouveaux services offerts, sachant qu'il est souhaitable de minimiser le premier et de maximiser le second.

Ainsi, afin d'offrir un maximum de services d'utilité publique avec un maximum de souplesse, nous avons délibérement choisi de ne pas supporter les applications patrimoniales. Celles-ci étaient à nos yeux trop contraintes pour mettre en évidence le potentiel de l'interaction multi-instrument multisurface. Nous avons donc concentré notre effort sur la facilité de programmation et non sur la réutilisation de l'existant.

VIII.1.8. FACILITÉ DE PROGRAMMATION

Afin de conserver les modèles de programmation existants, nous avons réfléchi à la problématique :

- Qu'est-ce qu'il est convenable d'écrire et qu'est-ce qui ne l'est pas pour le programmeur ? En effet, le but d'I-AM est de cacher la complexité de la création de l'espace unifié tout en laissant le maximum de souplesse. Ainsi, avant de développer I-AM, nous avons d'abord défini toutes les interfaces de programmation.

Au final, il nous semble que notre but a été atteint. Il suffit de voir le code de l'IAMApp exemple du chapitre 7 pour voir que les habitudes de programmation des développeurs ont été conservées au maximum.

Ayant présenté l'adéquation de notre solution vis-à-vis des requis logiciels, nous proposons le tableau récapitulatif suivant :

Systèmes multi-instrument multisurface Notation UDP/C : N-N-N/*	Découverte des acteurs et des ressources d'interaction	Localisation des entités physiques	Modélisation du couplage	Interface migrable et distribuable	Architecture distribuée	Conformité de la latence	Réutilisation de l'existant	Facilité de programmation
Ubit et UMS	-	+	+	+	+	++	-	+
I-AM	+	++	++	++	++	+	-	++

Figure 70. Tableau récapitulatif de l'adéquation d'I-AM vis-à-vis des requis logiciels.

VIII.2. Perspectives

Au chapitre 1, nous avons introduit notre travail avec deux questions :

- Comment gérer la découverte mais aussi l'hétérogénéité des ressources de calcul, de communication et d'interaction ?
- Quels modèles d'interaction proposer? Autrement dit, le paradigme « écran-dispositif de saisie-dispositif de pointage » est-il encore valide ? Qu'offrir au-delà du desktop, puisque le lieu d'interaction est maintenant l'espace. La notion de fenêtre fait-elle encore sens ?

Mes travaux de thèse ont tenté de répondre à la première question plus technique sans pour autant ignorer la deuxième qui insiste plus sur les aspects interactionnels. Ici, nous suggérons quelques pistes en rapport avec la seconde question.

VIII.2.1. VERS UN DESKTOP ++

Les seuls exemples d'interfaces migrables et distribuables que nous avons donnés sont les fenêtres et les interacteurs qu'elles contiennent. Cependant, si l'on veut étendre notre solution pour en faire l'interface du système d'exploitation de demain (le Meta UI de Barralon [Barralon 2004]), il convient d'étudier la place des icônes et des barres de menu et de navigation dans un espace d'interaction unifié.

En effet, si l'espace d'interaction peut évoluer à tout instant (couplages et découplages opportunistes) :

- Quelle est la signification du déplacement d'une icône d'un écran sur un autre ? Simple réorganisation de l'espace de travail, demande de transfert de fichier ou partage d'information ? Comment s'y retrouver et que choisir ?
- Aujourd"hui en haut de l'espace de travail se trouve la barre de menu et en bas ou sur les cotés une barre de navigation. Où va-t-on les mettre lorque l'espace va s'agrandir ou rétrécir ? Quand l'espace deviendra trop grand comment limiter la trajectoire d'interaction pour y accéder au plus vite ?

Toutes ces questions montrent qu'un espace interactif qui évolue dynamiquement risque d'être pour l'utilisateur un espace très instable. De ce fait, les propriétés de souplesse et de robustesse de l'interaction devront être étudiées plus avant. À l'évidence, le problème du Desktop++ ne se limite pas aux seuls emplacement et rôle des icônes et des barres de raccourcis, mais vise à trouver des techniques d'interaction en accord avec le nouveau contexte d'interaction.

De plus, comme I-AM permet l'utilisation conjointe de multiples ressources d'interaction par de multiples utilisateurs, elle pourrait être utilisée pour concevoir un système réellement collaboratif ressemblant à

Dynamo [Izadi et al 2003]. Pour cela, la couverture fonctionnelle d'I-AM doit être étendue pour prendre en compte les notions d'espaces publics, semi-privés et privés, et autoriser l'accès aux ressources d'interaction en fonction du contexte d'usage.

VIII.2.2. PLASTICITÉ DES INTERFACES

I-AM permet l'assemblage dynamique de surfaces afin de former un espace d'interaction plus grand. Dans notre étude, nous avons toujours mis en avant des couplages visant l'unification de l'espace d'interaction. Ainsi, involontairement, en cherchant l'unification des ressources d'interaction au sein d'un même espace, nous avons réalisé une certaine uniformisation.

En effet, nous n'avons utilisé le fait que les surfaces et les instruments pouvaient être hétérogènes voire très hétérogènes et que, dans ces conditions, pendant et après la migration et la distribution de l'IHM entre les surfaces, une plastification devenait nécessaire. Ici le terme plasticité signifie une adaptation de l'IHM au contexte d'usage tout en préservant l'utilisabilité [Thevenin 2001].

Les applications I-AM offrent la gestion de la migration et de la distribution de l'IHM mais donnent aussi avec les TopologyEventListeners, un moyen de contrôler ces services. En effet, il est possible de calculer pour chaque interacteur sa répartition sur l'espace des surfaces (par exemple 50% sur la surface S1, 30 % sur la surface S2 et 20% dans le vide). De plus, pour chaque interacteur, le programmeur peut contrôler sa visibilité sur les surfaces (méthode AuthorizeAccess). Ainsi, un bouton peut être visible sur une surface mais pas sur une autre. On peut aussi imaginer le remplacement du bouton par un autre interacteur ou le déplacement de ce dernier (s'il est plus de 50% dans le vide) suivant la disposition de l'interface. Avec ces services, il est donc possible de définir au-dessus d'I-AM une couche dédiée à la plasticité des interfaces graphiques. L'intérêt d'une telle couche est renforcé si l'on crée un espace interactif sur un ensemble hétérogène de ressources comprenant des murs, des ordinateurs portables et des PDA.

Certes, à l'heure actuelle notre solution est trop gourmande pour être utilisée sur un PDA. Mais, l'évolution rapide des caractéristiques de ce type de dispositif devrait bientôt rendre possible l'utilisation d'I-AM sur PDA. Cependant, il faut bien voir que mettre une fenêtre à cheval entre un PDA et un écran n'a que peu d'intérêt car les bords du PDA sont presque aussi larges que son écran. Ainsi, il est peut être préférable d'envisager une plastification de l'IHM d'une application lors de son déplacement d'un écran usuel vers celui d'un PDA et inversement.

VIII.3. Synthèse

Au bilan, ces travaux de recherche doctorale ont contribué à définir les fondements conceptuels pour le développement d'IHM migrable et distribuable en informatique ambiante.

Tout d'abord, notre ontologie a permis de définir les concepts clés du thème de recherche abordé et d'illustrer son potentiel interactionnel. Parmi les notions mises en avant par nos travaux (action, observation, relations spatiales, projection de l'IHM, ...), le couplage est à nos yeux le concept le plus riche car il va de pair avec la créativité. En effet, c'est en couplant les entités que le système et l'utilisateur définissent de nouveaux services.

Dans la lignée de la vision de Weiser, notre infrastructure logicielle I-AM apporte une première solution au couplage des instruments et des surfaces. Elle offre aux programmeurs et aux utilisateurs la vision d'un espace interactionnel librement configurable. Avec I-AM, l'IHM sort de ses frontières originales (l'écran) pour se répartir dans l'environnement ambiant. Il conviendra naturellement de tester et valider les possibilités offertes par le couplage de ressources. I-AM peut servir de plate-forme pour mener à bien la construction de ces nouveaux desktops.

Annexes

Annexe 1 : Localisation et couplage de surfaces

Dans cette annexe, nous présentons trois techniques d'interaction pour coupler et découpler les surfaces ainsi qu'un feedback qui rend perceptible l'état de ce dernier.

VIII.4. Localisation par proximité : des surfaces équipées de capteurs

Coupler et découpler les surfaces sont, sauf dans de rares exceptions, des actions non pas initiées par I-AM mais par les utilisateurs car ceux-ci ont besoin de contrôler leur espace d'interaction. Nous avons construit une technique d'interaction pour coupler et découpler les surfaces sur le principe de la manipulation directe et la notion de proximité : pour coupler les surfaces l'utilisateur met les surfaces en contact et pour les découpler il les sépare.

Dans cette optique, nous avons équipé les écrans de capteurs IrDa comme sur la figure 71 suivant une idée proche de Dietz et al [Dietz et al 2003]. Les capteurs sont positionnés tout autour de la surface par couples (un émetteur avec un récepteur). Au début, le nombre de couple de capteurs prévus par surface était de 6 ((2 sur la longueur + un sur la largeur) * 2). A terme, nous pensions équiper les surfaces de 16 couples de capteurs ((5 + 3)*2) afin d'augmenter la précision de notre système de localisation.

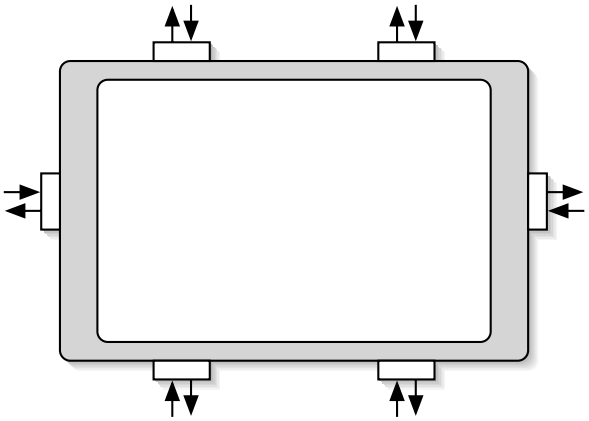

Figure 71. Une surface (Tablet PC) équipée de 6 paires de capteurs IrDa.

Tous les capteurs d'une même machine sont contrôlés par un microprocesseur relié à un port USB. Cette connexion lui permet de s'alimenter et de communiquer ces données au reste du système. Le microprocesseur exécute un programme qui demande tour à tour à chaque capteur d'émission de publier un message et à chaque capteur de réception d'écouter un possible message émis par un capteur situé en face de lui. Les capteurs d'émission sont réglés pour n'émettre qu'à moins de 5 cm. Le signal émit avec ce type de capteurs est très directionnel, ainsi seul un alignement strict des capteurs permet une localisation relative des surfaces. Bien sûr, le cycle de réception-émission varie aléatoirement pour garantir la mise en relation d'un capteur de réception avec un capteur d'émission lorsqu'ils sont face à face. Toutes ces caractéristiques du système de localisation ainsi créé transforment les diodes IrDa en capteurs de proximité.

Notre prototype permet de localiser les surfaces les unes par rapport aux autres car chaque microprocesseur et ses capteurs associés communiquent avec un contexteur de Liaison. Notre système autorise des rotations de 90, 180 ou 270 degrés. Ici, le contexteur de liaisons encapsule sous la forme de SurfacesLinks les informations que le microprocesseur lui fournit. Il propage ces données dans l'architecture des contexteurs. La figure 72 illustre la mise en relation de certains capteurs situés sur deux surfaces S1 et S2 distinctes que nous donnons en exemple. Cette figure donne un aperçu général du fonctionnement de ce système :

- Au début, les surfaces sont trop éloignées pour que certains de leurs couples de capteurs puissent communiquer (cas A). Ainsi, l'automate du programme exécuté par chaque microprocesseur se trouve dans l'état « recherche de capteurs ». Chaque microprocesseur envoie à son contexteur de liaisons plusieurs messages du type : le

capteur numéro X de la surface SY voit le capteur numéro –1 de la surface –1. (il ne voit donc aucun capteur). Ici, chaque capteur est numéroté et chaque microprocesseur connait l'identifiant I-AM de la surface qu'il gère. Lorsque le contexteur reçoit ces messages, il demande à un autre contexteur les positions respectives de tous les capteurs numérotés de la surface SY concernée. Une fois ces informations connues, il publie plusieurs messages du type : « le point de contact (X1, Y1) de la surface SY est en correspondance avec le point de contact (-1, -1) de la surface –1 » .

- Puis, les surfaces S1 et S2 sont rapprochées jusqu'à ce qu'au moins un couple de capteurs de la surface S1 trouve un couple de capteurs de la surface S2 (cas B). A ce moment là, ces couples de capteurs en vis-à-vis échangent leur identifiant afin que chacun puisse dire à son microcontrôleur « moi, capteur X, je vois le capteur Y de la surface SY ». Puis comme précédemment, ces informations sont données aux contexteurs de liaisons qui les publient au bon format : « le point (X1, Y1) de la surface SX est en contact avec le point (X2, Y2) de la surface SY. Par la suite, chaque IAMApp sera informée de l'existence des liaisons ainsi créées et mettra à jour sa topologie logique. Ici, cela aura pour effet de coupler les surfaces S1 et S2.
- Enfin, les surfaces sont éloignées (cas C). Les capteurs précédemment en contact ne le sont plus. Les contexteurs de liaisons prennent connaissance de la situation et publient alors l'absence de liaisons entre S1 et S2. Pour I-AM, ces deux surfaces sont donc maintenant découplées.

Nous avions expliqué au chapitre 7 que les liaisons étaient monodirectionnelles. La vrai raison est la suivante : pour que la relation spatiale entre deux surfaces n'existent plus, il faut que le capteur de réception de S1 ne voit plus le capteur d'émission de S2 mais aussi que le capteur de réception de S2 ne voit plus le capteur d'émission de S1. Une liaison est donc monodirectionnelle car elle véhicule l'information minimale C1 voit C2 et non l'information C1 voit C2 et C2 voit C1.

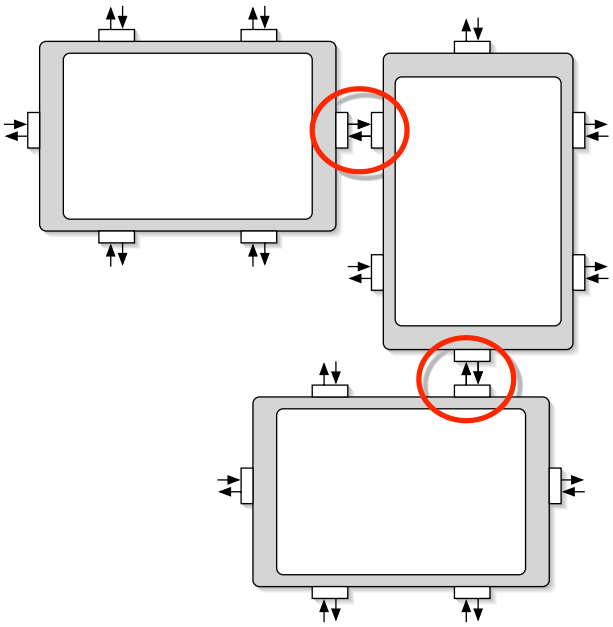

Figure 72. Trois surfaces équipées de capteurs. Les cercles rouges montrent les paires de capteurs en contact.

Nous avons travaillé avec deux étudiants en électronique (3 mois pour le premier et 6 pour le second) pour concevoir ce système de localisation des surfaces. Malheureusement pour diverses raisons, ce système de capteurs n'est toujours pas opérationnel. Nous n'avons donc pas pu utiliser cette technique. Cependant, nous avons créé une simulation logicielle réutilisant les contexteurs de liaisons conçus pour ce prototype afin de mettre en évidence le caractère simple et efficace de notre notion de liaison. Dans la section qui suit, nous décrivons cette simulation logicielle ainsi qu'un feedback particulier qui permet de rendre observable le couplage des surfaces. Il est important de noter que ce feedback proposé par Barralon [Barralon et al 2004] est indépendant de la technique d'interaction utilisée pour le couplage. Il fonctionne donc avec les trois techniques présentées ici et pourrait être utilisé avec d'autres.

VIII.5. Une simulation logicielle : le gestionnaire de la topologie des surfaces

La technique précédente est basée sur l'hypothèse forte qu'une surface (un écran) peut être déplacée facilement à la seule force des bras. En situation de mobilité, cette hypothèse semble raisonnable mais elle a besoin d'être vérifiée dans plusieurs contextes d'utilisation réalistes. Afin d'informer l'utilisateur que sa demande de couplage a été com-

prise par le système et que le couplage est maintenant opérationnel, Barralon propose de donner un feedback à l'endroit où les utilisateurs ont la plus grande chance d'avoir leur focus d'attention c'est-à-dire sur les surfaces elles-mêmes. De plus, ce feedback doit exprimer la fonction que le couplage crée : étendre la surface d'affichage principale. Comme on peut le voir sur la figure 73, un ruban bleu autour des écrans couplés délimite l'espace d'affichage logique rendu ainsi accessible.

Les écrans que nous utilisons ne sont pas encore équipés de capteurs. C'est pourquoi nous simulons les contacts physiques et/ou la proximité à l'aide d'une application appelée SurfacesConfigurator. Ce logiciel est actuellement utilisé par un utilisateur magicien d'Oz qui copie les actions des utilisateurs quand ils mettent en contact les surfaces. Bien sûr, cette application peut aussi être utilisée par les vrais utilisateurs du système s'ils le désirent.

Le SurfacesConfigurator peut s'exécuter sur n'importe quelle machine de l'environnement pourvue qu'elle soit reliée en réseau avec celles qui utilisent I-AM. En effet, rien n'oblige cette machine à être elle même équipée d'un PlatformManager. Les figures suivantes (de figure 73 à figure 78) montre un scénario qui illustre la technique d'interaction que nous voulions réalisée à l'aide de capteurs de proximité.

Figure 73. Etat initial du scénario. Deux machines utilisent l'infrastructure I-AM. Leurs écrans ne sont pas couplés : sur chacun on distingue un bord bleu qui renforme leur contour.

Ce feedback sur les surfaces est produit par des composants appelés TopologyRenderer. Chaque WindowManager a le sien et lui confie l'affichage du feedback sur les surfaces locales. Chaque TopologyRenderer utilise les événements logiques d'une IAMApp A pour produire le contour bleu délimitant les frontières de l'espace unifié. Ce feedback évolue donc dynamiquement en fonction de l'évolution de la topologie logique de A. En effet, le TopologyRenderer utilise les événements topologiques qu'il reçoit en tant que LogicalTopologyEventListener pour créer ou détruire les passerelles entre les surfaces.

La figure 73 correspond à l'état initial du scénario. On y voit deux machines utilisant l'infrastructure I-AM pour lesquelles leurs écrans sont découplés : le ruban bleu fait le tour complet de chaque surface. En parallèle, l'interface utilisateur du SurfacesConfigurator montre que deux écrans ont été découverts (voir figure 74). Pour découvrir les surfaces d'une grappe, notre logiciel utilise l'infrastructure des contexteurs (il demande à connaître toutes les surfaces de l'environnement comme le font les IAMApps). Comme le montre la figure 74, les surfaces sont représentées par des rectangles de différentes couleurs dont la taille et les bords sont proportionnels à la réalité. Une flèche à l'intérieure de chaque rectangle dénote l'orientation de la surface dans l'espace physique. De même, dans le coin haut gauche de chaque rectangle est affiché l'identifiant unique de la surface représentée. Les rectangles peuvent être déplacés et tournés à l'aide de la souris pour modéliser l'orientation et la position des surfaces dans le monde physique.

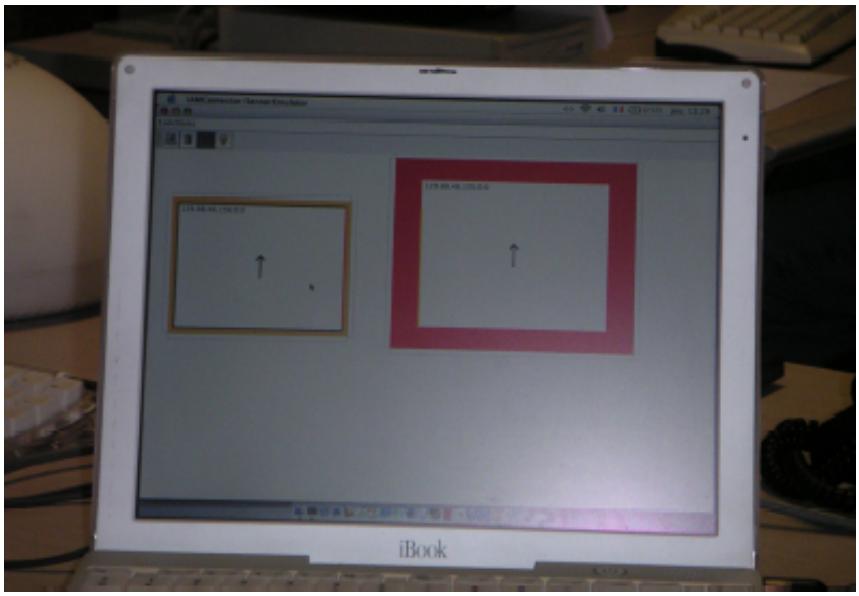

Figure 74. L'interface graphique du SurfaceConfigurator. Ici, l'espace interactif est composée de deux surfaces decouples.

Sur la figure 75, l'utilisateur (ou le magicien) rapproche les deux rectangles et les lie verticalement. Quand l'icône "ordinateur" du menu est sélectionnée, le SurfaceConfigurator crée les liaisons adéquates et les publient à l'aide du contexteur dédié à cette fonction : le SurfacesLinkContextor. A ce moment là, les deux surfaces sont couplées car tous les applications I-AM vont découvrir la relation spatiale associée à la liaison ainsi crée. Les TopologyRenderer des deux surfaces sont avertis et produisent le feedback approprié. Ce feedback, visible sur la figure 76, concrétise le passage qui existe maintenant entre les deux surfaces.

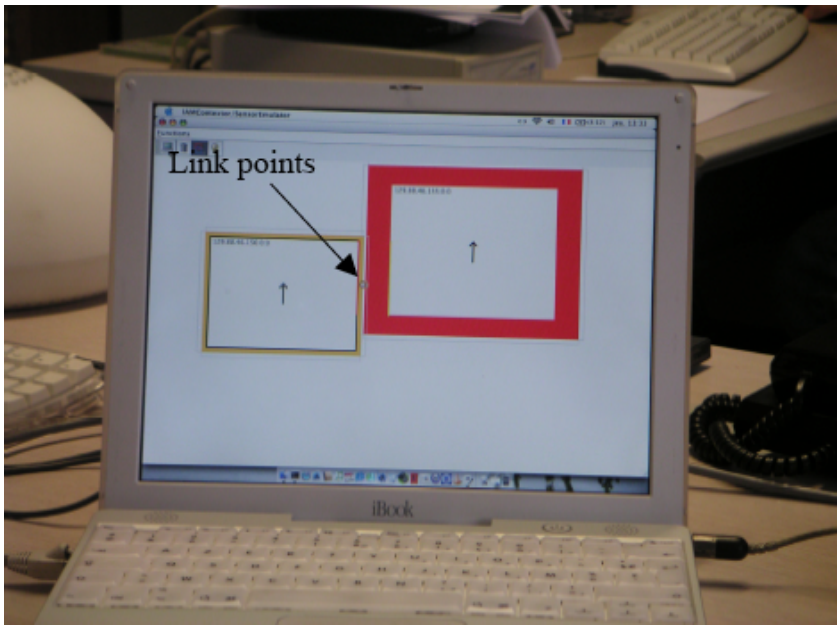

Figure 75. Simulation de l'action "mettre les deux écrans en contact".

Sur la figure 76, les deux écrans sont maintenant couplés et le ruban bleu entoure maintenant l'espace logique rendu ainsi accessible. Sur les bords externes des surfaces le ruban reste inchangé ; par contre au centre on peut observer un passage par lequel les interacteurs peuvent migrer tout en préservant leur observabilité. La figure 77 montre une fenêtre "Gloss" à cheval entre les deux écrans.

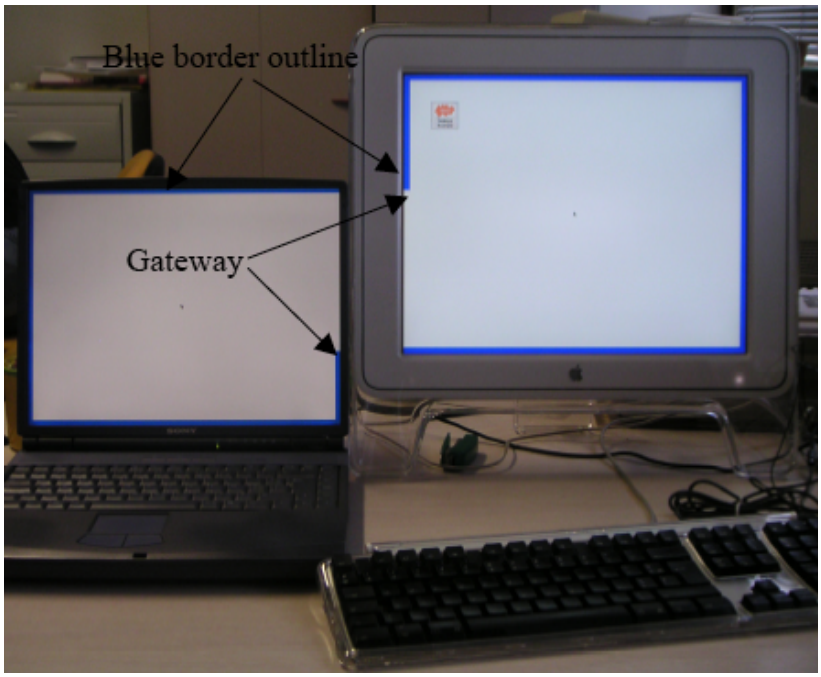

Figure 76. Les deux écrans sont couplés pour créer un espace d'interaction plus grand. Le ruban bleu denote l'espace ainsi rendu accessibles aux interacteurs pour leurs affichages.

Figure 77. Une fenêtre I-AM transite par le passage crée entre les deux surfaces après leur couplage.

Sur la figure 78, l'utilisateur a supprimé le lien entre les deux surfaces. Le résultat obtenu est le découplage de deux écrans. La fenêtre "Gloss" qui était à cheval disparait de l'un des deux écrans car ce dernier n'étend plus son espace d'affichage. Bien sûr, si après le découplage la fenêtre n'était plus visible sur aucune surface, elle serait automatique-

ment repositionnée par le window manager. Les régles qui gouvernent la projection des interacteurs sont, on le rappelle, présentées dans la section 5.2 du chapitre 7.

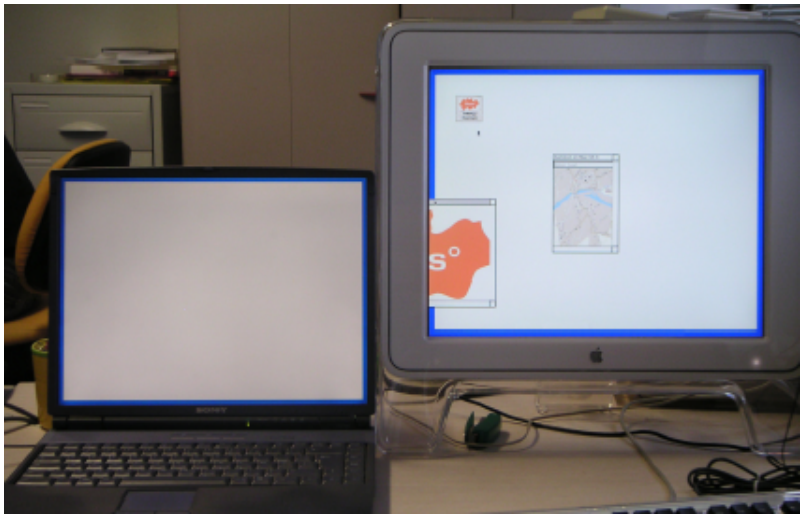

Figure 78. Les deux écrans sont découplés. Ils jouent maitenant le rôle de deux surfaces indépendantes.

I-AM n'est pas fortement limitée par le nombre de surfaces couplées. Ainsi, une mosaïque d'écrans peut facilement être construite afin de créer un très large espace d'affichage. Cependant, pour être honnête nous n'avons pas encore testés de configuration au delà de 5 machines (3 mac et 2 pc).

VIII.6. Une technique d'interaction sans capteur physique

Sans I-AM, les deux surfaces ne sont ni couplées, ni couplables (état 1 de l'automate de couplage). Lorsque les PlatformManager du PC et du Macinstosh ont publié leurs ressources d'interaction, les deux surfaces deviennent couplables (état 4 de l'automate). Cet état est rendu observable par la présence d'un halo bleu qui entoure chacun des écrans : chaque halo délimite un espace d'affichage. Nous avons imaginé une technique d'interaction intégrée à I-AM permettant le couplage par gestes synchronisés façon SynTap [Rekimoto et al 2003 (1)], mais améliorant la prédictibilité de l'effet du couplage lorsqu'elle est utilisée avec le feedback donné par le TopologyRenderer.

Comme le montre la figure 79 a), l'utilisateur, une souris dans chaque main, amène les curseurs des souris dans le halo des bords à joindre. En appuyant simultanément sur les boutons des souris, les bords de

jointure pâlissent, montrant le passage qui s'ouvrira entre les deux écrans au relâché des boutons. La position du curseur dans le halo désigne de manière directe et dynamique un point de liaison de la surface. En maintenant les boutons enfoncés, jouant sur les positions respectives des points de liaison, l'utilisateur peut ajuster la taille souhaitée de la passerelle et donc la forme du futur espace unifié (figure 79 b). La passerelle est opérationnelle au relâchement simultané des deux boutons. Le halo s'ouvre au niveau de la passerelle pour traduire la continuité de l'espace d'affichage formé par le couplage des deux surfaces.

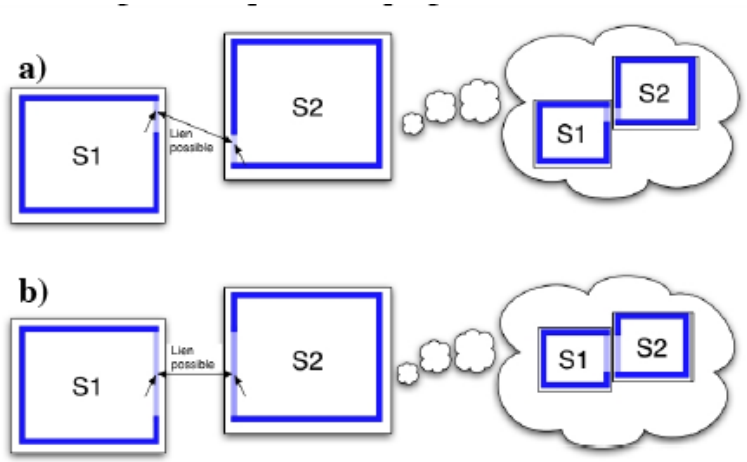

Figure 79. Ajustement dynamique des points de liaison entre surfaces et évaluation a priori de l'espace unifié potentiel.

Pour le découplage, une icône apparaît 2 secondes lorsque le curseur d'une souris passe par la passerelle et que cette souris n'est pas couplée à un interacteur. Un « press » sur l'icône permet de savoir si les surfaces sont découplables et ce que deviendront les frontières (les halos) si le découplage est effectué. Si le « release » a lieu sur l'icône alors que les surfaces sont découplables, le découplage est effectif.

VIII.7. Conclusion

Les contexteurs de liaisons permettent d'intégrer dans I-AM tous types de capteurs de localisation. Nous venons de voir 3 exemples représentatifs de la souplesse offerte par cette solution technique. La dernière technique d'interaction nous semble la solution la plus intéressante car c'est une solution simple, facilement intégrable au WindowManager de chaque plate-forme, qui ne nécessite pas d'équipement particulier contrairement à la première. Bien sûr, il nous reste à évaluer cette technique sur un pannel représentatif d'utilisateurs et à analyser si elle

permet une meilleur appréhension de l'espace unifié construit par couplage de surfaces. En perspective, nous pouvons aussi imaginer l'encapsulation des accéléromètres utilisés par [Hinckley 2003] et des capteurs LED de [Dietz et al 2003] par des contexteurs de liaisons tout en gardant le rendu visuel du TopologyRenderer proposé par Barralon.

Bibliographie

VIII.8. Références littéraires

[Barralon et al 2004]
Barralon, N. Lachenal, C. Coutaz, J. Couplage de ressources d'interaction. IHM2004. Namur, Belgique.Pages 13-20.

[Barralon 2004]
Barralon, N. Meta UI : vers un Desktop++. Rencontre jeunes chercheurs en Interaction Homme-Machine. 2004.

[Baudisch et al 2003]
Baudisch, P., Cutrell, E., Robbins, D., Czerwinski, M., Tandler, P., Bederson, B., Zierlinger, A. Drag-and-Pop and Drap-and-Pick: Techniques for Accessing Remote Screen Content on Touch- and Pen-Operated Systems. INTERACT'03, Rauterberg et al. Eds., IOS Press Publ., IFIP, 2003, Pages.57-64.

[Baudish et al 2004]
Baudisch, P., Cutrell, E., Hinckley, K., and Gruen, R. Mouse Ether: Accelerating the Acquisition of Targets Across Multi-Monitor Displays. In CHI 2004 Extended Abstracts (short paper), Vienna Austria, April 2004, pp. 1379-1382.

[Baümer et al 1997]
Bäumer, D. Riehle, D. Siberski, W et Wulf, M. The Role Object Pattern. PLoP 1997. Septembre 1997. Allerton Park Monticello, Illinois, USA.

[Bederson et Hollan 1994]
Bederson, B. B., Hollan, J. D. (1994). Pad++: A Zooming Graphical Interface for Exploring Alternate Interface Physics. In Proceedings of User Interface and Software Technology (UIST 94) ACM Press, Pages 17-26.

[Beaudoin-Lafon 2000]
Michel Beaudouin-Lafon. Instrumental interaction: an interaction model for designing post-WIMP user interfaces. Conference on Human Factors in Computing Systems. SIGCHI The Hague, The Netherlands. Pages: 446 - 453. 2000.

[Bérard 1999]
Bérard, F. Vision par ordinateur pour l'interaction homme-machine fortement couplée. Thèse de l'université Joseph Fourier. 1999.

[Bérard 2003]
Bérard, F. The Magic Table: Computer-Vision Based Augmentation of a Whiteboard for Creative Meetings. In IEEE workshop on Projector-Camera Systems, Nice, France, 2003.

[Bier et Freeman 1991]
Bier, E. Freeman, S. MMM: a user interface architecture for shared editors on a single screen. Proceedings of the 4th annual ACM symposium on User interface software and technology UIST'91. Hilton Head, South Carolina, United States. Pages: 79 - 86. 1991.

[Bier et al 1993]
Bier, E., Stone, M., Pier, K., Buxton, W., DeRose, T. Toolglass and Magic Lenses: The See-Through Interface. In Proceedings of SIGGRAPH 93, 1993, Pages.73-80.

[Blattner et al 1989]
Blattner, M. , Sumikawa, D. , and Greenberg, R. Earcons, icons : their structure, common design principles. In Proc. HCI'89, 1989.

[Borkowski et al 2003]
S. Borkowski, O. Riff, J.L. Crowley. Projecting rectified images in an augmented environment ProCams Workshop, 2003.

[Brumitt et Shafer 2000]
Brumitt, B. et Shafer, S. Better Living Through Geometry. CHI Workshop on Situated Interaction in Ubiquitous Computing, April 2000. Also submitted to Springer Journal of Personal Technologies.

[Brumitt et al 2002]
Brumitt, B., Meyers, B., Krumm, J., Kern, A., and Shafer, S., EasyLiving: Technologies for Intelligent Environments", Handheld and Ubiquitous Computing, September 2000.

[Campbell et Maglio 2003]
Campbell, C et Maglio, P. Segmentation of Display Space Interferes with Multitasking. INTERACT'03, Rauterberg et al. Eds., IOS Press Publ., IFIP, 2003, Pages.575-582.

[Cadoz 1994]
Cadoz, C. Le geste canal de communication homme-machine : la communication«instrumentale. Sciences Informatiques - Numéro Spécial: Interface Homme-Machine, vol. 13, no. 1, pp. 31-61, 1994.

[Card 83]
Card, S. Moran, T. Newell, A. The Psychology of Human-Computer Interaction" Lawrence Erlbaum Associates, 1983.

[Card et al 1990]
Card, S. Mackinlay, J. Robertson, G. The design space of input devices. Conference on Human Factors in Computing Systems SIGCHI 1990 Seattle, Washington, United States. Pages 117-124. 1990.

[Chatty 1994]
Stéphane Chatty. Extending a graphical toolkit for two-handed interaction. Proceedings of the 7th annual ACM symposium on User interface software and technology UIST'94. Marina del Rey, California, United States. Pages 195 - 204. 1994.

[Coutaz et al 2002]
J.Coutaz, C.Lachenal, G.Rey. Initial reference framework for multi-surface interaction. Livrable D17. Projet FET GloSS. 2002. http://iihm.imag.fr/projects/Gloss/gothenburg.html

[Coutaz et al 2003 (1)]
Coutaz, J., Lachenal, C., Dupuy-Chessa, S. Ontology for Multi-Surface Interaction. In Proceedings of Interact'03, Zürich, Switzerland, 2003.

[Coutaz et al 2003 (2)]
Coutaz, J. Lachenal, C. Barralon, N. Rey, G. Final examples of interaction techniques using multiple interaction surfaces. Livrable 20, Projet FET GloSS, 2003.

[Deflin 2001]
Deflin E., Weill A., Conkar V. Communicating Clothes: Optical Fiber Fabric for a New Flexible Display, Proc. Avantex Symposium ,13-15 Mai 2002, Frankfurt.

[Dietz et al 2003]
Dietz, P.H.; Yerazunis, W.S.; Leigh, D.L. Very Low Cost Sensing and Communication Using Bidirectional LEDS. International Conference on Ubiquitous Computing. UbiComp 2003, Octobre 2003.

[Edwards et Grinter 2001]
Edwards, K., Grinter, R. At Home with Ubiquitous Computing : seven challenges. In Proceedings of Ubicomp'01, Atlanta, Georgia, 2001, Pages.256-272.

[Fishkin et al 1999]
Fishkin, K.P., Gujar A., Harrison B.L., Moran T.P., Want, R. Embodied User Interfaces for Really Direct Manipulation. In Communication of the ACM, 1999, Pages.74-80.

[Fitzmaurice et al 1995]
Fitzmaurice, G. Ishii, H. et Buxton, W. Bricks: Laying the Foundations for Graspable User Interfaces. ACM conference on Computer-Human Interaction. CHI 1995.

[Flury et Privat 2003]
Flury, T. Privat, G. An infrastructure template for scalable location-based services. France Télécom R&D, DIH/OCF. SOC Grenoble, France. 2003.

[Furnas et Bederson 1995]
Furnas, G. W., & Bederson, B. B. (1995). Space-Scale Diagrams: Understanding Multiscale Interfaces. In Proceedings of Human Factors in Computing Systems (CHI 95) ACM Press, Pages. 234-241.

[Gram et Cockton 1996]
Gram, Ch., Cockton, G. (Eds.). Design Principles for Interactive Software. Chapman & Hall, London, 1996.

[Hardenberg et Bérard 2001]
Hardenberg, C. Bérard, F. Bare-Hand Human-Computer Interaction, in ACM workshop on Perceptive User Interfaces (PUI 2001), Orlando, Florida.

[Harrison et al 1998]
Harrison, B. Fishkin, K. Gujar, A. Mochon, C. Want, R. Squeeze me, hold me, tilt me! An exploration of manipulative user

interfaces. Conference on Human Factors in Computing Systems. SIGCHI. Los Angeles, California, United States. Pages: 17 - 24. 1998

[Hightower et al 2002]
Jeffrey Hightower, Barry Brumitt, and Gaetano Borriello, The Location Stack: A Layered Model for Location in Ubiquitous Computing. Proceedings of the 4th IEEE Workshop on Mobile Computing Systems & Applications (WMCSA 2002), (Callicoon, NY), Pages 22-28, Juin 2002

[Hinckley et al 1998]
Hinkley, K., Czerwinski, M., Sinclai, M. Interaction and Modeling Techniques for Desktop Two-Handed Input. In Proceedings of of the ACM UIST'98 Symposium on User Interface Software and Technology, San Francisco, California, 1998, Pages 49-58.

[Hinckley 2003]
Hinckley, K. Synchronous Gestures for Multiple Persons. In Proceedings of the ACM UIST 2003 Symposium on User Interface Software and Technology, Vancouver, Canada, 2003, Pages 149-158.

[Hinckley et al 2004]
Hinckley, K., Ramos, G., Guimbretiere, F. Baudisch, P., and Smith, M. Stitching: Pen Gestures that Span Multiple Displays. In Proceedings of AVI 2004, Gallipoli, Italy, May 2004, pp. 23-31.

[Hoffmann et Scott]
Frank Hoffmann, James Scott. Location of Mobile Devices Using Networked Surfaces. Proceedings of the Fourth International Conference on Ubiquitous Computing. UbiComp 2002. Göteborg, Sweden, LNCS 2498, Springer-Verlag, September 2002.

[Holmquist et al 2001]
Holmquist, L.E., Mattern, F., Schiele, B., Alahuhta, P., Beigl M., Gellersen, H.W. Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In Ubicomp 2001, Atlanta, Georgia, 2001, Pages 116-122.

[Hourcade et Bederson 1999]
Hourcade J., Bederson B. Architecture and Implementation of a Java Package for Multiple Input Devices (MID). 1999. http://www.cs.umd.edu/hcil/mid/

[Ishii 1990]
Ishii, H. TeamWorkStation: Towards a Seamless Shared Workspace. Proceedings of CSCW '90, ACM SIGCHI and SIGOIS, Los Angeles, CA, Oct. 7-10, 1990, Pages 13-26.

[Ishii et Ullmer 1997]
Ishii, H., and Ullmer, B. Tangible Bits: Towards Seamless Interfaces between People, Bits and Atoms. In Proc. CHI'1997, 1997, Pages 234-241.

[Izadi et al 2003]
Izadi S., Brignull, H. Rodden, T. Rogers , Y. Underwood, M. Dynamo: a public interactive surface supporting the cooperative sharing and exchange of media. The 16th annual ACM symposium on User interface software and technology, UIST 2003. Vancouver, Canada, ACM Press. 2003.

[Jacob et al 2002]
Jacob R., Ishii H., Pangaro G. Patten J. A Tangible Interface for Organizing Information using a Grid. ". Proc. of the ACM conf. On Human Factors in Computer Human Interaction CHI2002, ACM, 2002, Pages 339-346.

[Jiang et Steenkiste 2002]
Changhao Jiang, Peter Steenkiste. A Hybrid Location Model with Computable Location Identifier for Ubiquitous Computing. Ubicomp 2002.

[Johanson et al 2002a]
B. Johanson, G. Hutchins, T. Winograd, et M. Stone. PointRight: experience with flexible input redirection in interactive workspaces. Proceedings of the 15th annual ACM symposium on User interface software and technology UIST 2002. Paris, France. Pages 227-234.

[Johanson et al 2002b]
Johanson, B. Fox, A. Winograd, T. The Interactive Workspaces Project: Experiences with Ubiquitous Computing Rooms. IEEE Pervasive Computing Magazine 1(2), April-June 2002

[Johanson et Fox 2002]
Brad Johanson et Armando Fox. The Event Heap: A Coordination Infrastructure for Interactive Workspaces. Proceedings of the Fourth IEEE Workshop on Mobile Computing Systems and Applications. Page 83. 2002.

[Kellogg 2002]
Kellogg, S.B. Brian, D.F. Chi Jui, R.L. Ritchie, A. The "mighty mouse" multi-screen collaboration tool. Proceedings of the 15th annual ACM symposium on User interface software and technology. Paris, France. UIST 2002. Pages 209-212.

[Koleva et al 2000]
Koleva, B. Schnädelbach, H. Benford, S. Greenhalgh, C. Traversable interfaces between real and virtual worldsConference on Human Factors in Computing Systems archive. Proceedings of the SIGCHI conference on Human factors in computing systems. The Hague, The Netherlands. 2000. Pages: 233 - 240.

[Krishna et Luca 1995]
Krishna A.B. Luca C. Migratory applications. Proceedings of the 8th annual ACM symposium on User interface and software technology. Pittsburgh, Pennsylvania, United States. UIST 1995. Pages 132-142.

[Krueger 1990]
Krueger, M. W. “Artificial Reality II”. Addisson Wesley Publishing, 1990.

[Lecolinet 2003a]
Lecolinet, E. Pointeurs multiples: étude et implémentation. Actes Journées Francophones sur l'Interaction Homme-Machine IHM 2003. Pages 134-141. Caen. ACM Press 2003.

[Lecolinet 2003b]
Lecolinet, E. A molecular architecture for creating advanced GUI. Proceedings of the 16th annual ACM symposium on User interface software and technology. Vancouver, Canada. Pages 135-144.

[Lee et al]
Lee, B. Ballagas, R. Stone, M. Designing for Latency: In Search of the Balance Between System Flexibility and Responsiveness. Workshop Ubicomp 2003.

[Lyytinen et Yoo 2002]
Lyytinen, K. and Yoo, Y. Issues and Challenges in Ubiquitous Computing. Communications of the ACM, 45(12), Pages 62-65.

[Mackinlay et Heer 2004]
Mackinlay, J. D. and Heer, J. Wideband Displays: Mitigating Multiple Monitor Seams. Proceedings of the Human Factors in Computing Systems Conference CHI2004, Vienna, Austria.

[MacLean et McKerlie 1995]
MacLean, A. et McKerlie, D. Design Space Analysis and Use-Representations In J.M. Carroll (ed). Scenario-Based Design: Envisioning Work and Technology in System Development. Wiley, New-York, 1995.

[Mantei et al 1991]
Mantei, M. Baecker, R. Sellen, A. Buxton, W. Milligan, T. Wellman, B. Experiences in the use of a media space. Conference on Human Factors in Computing Systems : Reaching through technology. New Orleans, Louisiana, United States. Pages: 203 - 208. 1991.

[Matsushita et Rekimoto 1997]
Matsushita, N., Rekimoto, J. Holo Wall: Designing a Finger, Hand, Body, and Object Sensitive Wall. In Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology, Alberta, Canada, 1997, Pages 209-210.

[Maynes-Aminzade et al 2002]
Maynes-Aminzade, D. et al, Techniques for Interactive Audience Participation. ICMI 2002 Pittsburgh, Pennsylvania. 2002.

[McCall 1977]
McCall, J. Factors in Software Quality. General electric Eds, 1977.

[Moran et Carroll 1996]
Moran, T. P., Carroll, J. M. (eds.) Design Rationale: Concepts, techniques, and use. Lawrence Erlbaum. 1996.

[Myers et al 1998]
Myers B., Stiel H., Gargiulo R. Collaboration Using Multiple PDAs Connected to a PC. Proceedings Computer Supported Collaborative Work, CSCW98, Seattle, WA. Pages 285-294. 1998.

[Myers et al 2001a]
Brad Myers, Scott E. Hudson, et Randy Pausch, Past, Present and Future of User Interface Software Tools. edition John M. Carroll. HCI In the New Millennium. New York: ACM Press, Addison-Wesley, 2001. Pages 213-233

[Myers et al 2001b]
Myers B.A., Peck, C.H., Nichols J., Kong. D, Miller. R. Interacting At a Distance Using Semantic Snarfing (2001).

[Myers et al 2002]
Myers, B. Bhatnagar, R. Nichols, J. Peck, C. Kong, D. Miller, R. Long, C. Interacting At a Distance: Measuring the Performance of Laser Pointers and Other Devices. CHI 2002.

[Norman 1999]
Norman, D. Affordance, Conventions, and Design. In the May 1999 issue of Interactions, 1999, Pages.38-43.

[Normand 1992]
Normand, V. Coutaz, J. Unifying the Design and Implementation of User Interfaces through the Object Paradigm. ECOOP '92: Proceedings of the European Conference on Object-Oriented Programming. Utrecht, the Netherlands. 1992.

[Olsen et Nielsen 2001]
Olsen, D. Nielsen T. Laser pointer interaction. Conference on Human Factors in Computing Systems SIGCHI. Seattle, Washington, United States. Pages: 17 - 22. 2001.

[Pinharez 2001]
Claudio Pinhanez. The Everywhere Displays Projector: A Device to Create Ubiquitous Graphical Interfaces Proc. of Ubiquitous Computing 2001 (Ubicomp"01), Atlanta, Georgia, September 2001.

[Piper et al 2002]
Piper, B. Ratti, C. Ishii, H. Illuminating clay: a 3-D tangible interface for landscape analysis. Conference on Human Factors in Computing Systems archive. SIGCH. Minneapolis, Minnesota, USA 2002. Pages 355 - 362.

[Pook 2001]
Stuart Pook. Interaction and Context in Zoomable User Interfaces. Doctoral thesis, École Nationale Supérieure des Télécommunications, Paris, France, June 2001.

[Rekimoto 1996]
Rekimoto, J. Tilting operations for small screen interfaces. Proceedings of the 9th annual ACM symposium on User interface software and technology UIST'96. Seattle, Washington, United States. Pages: 167 - 168. 1996.

[Rekimoto et Saitoh 1999]
Rekimoto, J., Saitoh, M. Augmented Surfaces: A spatially continuous work space for hybrid computing environments. In Proceedings of ACM CHI'99 Conference on Human Factors in Computing Systems, Pittsburg, Pennsylvania, 1999, Pages 378-385.

[Rekimoto et al 2001]
Rekimoto, J., Ullmer, B., Oba, H. DataTiles: A Modular Platform for Mixed Physical and Graphical Interactions. In Proceedings of ACM CHI'01 Conference on Human Factors in Computing Systems, Seattle, Washington, 2001, Pages 269-276.

[Rekimoto et al 2003 (1)]
Rekimoto, J., Ayatsuka, Y., Kohno, M. SyncTap: An Interaction Technique for Mobile Networking. In Proceedings of MOBILE HCI 2003, Udine, Italy, 2003, Pages 104-115.

[Rekimoto et al 2003 (2)]
Rekimoto, J., Ayatsuka, Y., Kohno, M., Oba, H. Proximal Interactions: A Direct Manipulation Technique for Wireless Networking. In Proceedings of Interact'03, Zürich, Switzerland, 2003, pp.511-518.

[Rey et Coutaz 2004]
G.Rey, J. Coutaz. Le Contexteur : Capture et distribution dynamique d'information contextuelle. Mobilité et Ubiquité, ACM Publication, 2004, Pages 131-138.

[Richardson et al 2003]
Richardson, B. Leydon, K. Fernstrom, M. and Paradiso, J. Z-Tiles: Building Blocks for Modular, Pressure-Sensing Floorspaces. CCCT 2003.

[Schneiderman 87]
Schneiderman, B. Designing the user interface: Strategies for effective human-computer interaction, Addison-Wesley, 1987.

[Schuckmann et al 1996]
Schuckmann, C. Kirchner, L. Schummer, J. and Haake, J. Designing object-oriented synchronous groupware with COAST. In Proceedings of the ACM 1996.

[Shen et al 2004]
Shen, C.; Vernier, F.D.; Forlines, C.; Ringel, M., "DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction", ACM Conference on Human Factors in Computing Systems (CHI), April 2004.

[Stewart et al 1999]
Stewart, J. Bederson, B. Druin, A. Single display groupware: a model for co-present collaboration. Conference on Human Factors in Computing Systems SIGCHI. Pittsburgh, Pennsylvania, United States. Pages: 286 - 293. 1999

[Streitz et al 1999]
Streitz, N., Geibler, J., Holmer, T., Konomi, S., Müller-Tomfelde, C., Reischl, W., Rexroth, P., Seitz, P., Steinmetz, R. i-Land: An interactive Landscape for Creativity and Innovation. CHI 1999, Pages 120-127.

[Strietz et al 2001]
Streitz N., Tandler P. , Müller-Tomfelde C. , Konomi S. Roomware: Towards the Next Generation of Human-Computer Interaction based on an Integrated Design of Real and Virtual Worlds. In Human-Computer Interaction in the New Millenium, Carroll J. (Ed.), Addison-Wesley, 2001, Pages 553-578.

[Szalavári et Gervautz 1997]
Zsolt Szalavári, Michael Gervautz. The Personal Interaction Panel: A Two-Handed Interface for Augmeted Reality. Paru dans Computer Graphics Forum, 16, 3 EUROGRAPHICS'97, Budapest, Hungary, Pages 335-346, Septembre 1997.

[Tan et al 2003]
Tan, D. Gergle, D., Scupelli, P., Pausch, R. With Similar Visual Angles, Larger Displays Improve Spatial Performance. In Proc. Computer Human Interaction CHI 2003, ACM Publ., Fort Lauderdale, April 5-10, 2003, Pages 217-224.

[Tan et Czerwinski 2003]
Tan, D., Czerwinski, M. Effects of Visual Separation and Physical Discontinuities when Distributing Information across Multiple Displays. INTERACT'03, Rauterberg et al. Eds., IOS Press Publ., IFIP, 2003, Pages 252-255.

[Tandler 2001]
Tandler, P. Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogenous Devives. In Proc. of the 3rd International Conference on Ubiquitous Computing. UbiComp 2001. Atlanta Sept. Pages 96-115.

[Tandler et al 2001]
Tandler, P., Prante, T., Müller-Tomfelde, C., Streitz, N., Steinmetz, R. ConnecTables: Dynamic Coupling of Displays for the Flexible Creation of Shared Workspaces. In Proceedings of the ACM UIST'01 Symposium on User Interface Software and Technology, Orlando, Florida, 2001, pp.11-20.

[Thevenin 2001]
Thevenin David. Adaptation en Interaction Homme-Machine : le cas de la plasticité. Thèse de l'université Joseph Fourier de Grenoble. 2001.

[Ullmer et Ishii 1997]
Ullmer, B., Ishii, H. The metaDESK: Models and Prototypes for Tangible User Interfaces. In Proceedings of the ACM UIST'97 Symposium on User Interface Software and Technology, Alberta, Canada, 1997, Pages.223-232.

[Vogt et al 2004]
Florian Vogt, Justin Wong, Barry A. Po, Ritchie Argue, S. Sidney Fels, Kellogg S. Booth. Exploring Collaboration with Group Pointer Interaction. Computer Graphics International CGI'04. Crete, Greece. Pages 636-639. 2004.

[Want et al 2002]
Want, R. Pering, T. Danneels, G.Kumar, M. Sundar, M. and Light, J. The Personal Server: Changing the Way We Think about Ubiquitous Computing. Ubicomp 2002.

[Weiser 1991]
Weiser, M. The computer for the 21st century. Scientific American, 1991. Pages 94–104.

[Wellner et al 1993]
Wellner, P. Mackay, W, Gold, R. Computer-Augmented Environments: Back to the Real World - Introduction to the Special Issue. Volume 36, Number 7, Juillet 1993. Pages 24-26.

[Yee 2003]
Ka-Ping Yee. Peephole displays: pen interaction on spatially aware handheld computers. CHI 2003. Pages: 1-8. 2003.

VIII.9. Référence internet

[Apple]
http://www.apple.fr/

[Fame]
http://isl.ira.uka.de/fame/index.html

[FiCom]
http://www.fibercomputing.net/

[I-vibration]
http://www.i-vibrations.com/

[SmartBoard]
http://www.smarttech.com/

[Smart-Its]
http://smart-its.teco.edu. EU Initiative The Disappearing Computer, IST-2000-25428

[UPnP]
http://www.upnp.org/

[ZeroConf]
http://www.zeroconf.org/