UNIVERSITE PARIS XI
UFR SCIENTIFIQUE D'ORSAY

THESE
présentée
pour obtenir

le grade de DOCTEUR EN SCIENCES
DE L'UNIVERSITE PARIS XI ORSAY

par

Olivier BEAUDOUX

Sujet :

Espaces de travail interactifs et collaboratifs : Vers un modèle centré sur les documents et les instruments d'interaction

Soutenue le 29 juin 2004 devant la Commission d'examen

M. Michel BEAUDOUIN-LAFON (directeur de thèse)
M. Alain DERYCKE (rapporteur)
M. Philippe PALANQUE
M. Vincent QUINT
M. Michel RIVEILL (rapporteur)

Remerciements

Je tiens à remercier tous ceux qui m'ont permis, par leur soutien, de réaliser cette thèse et en particulier :

Michel Beaudouin-Lafon, directeur de thèse, pour avoir su me transmettre de sa passion ;

Alain Derycke et Michel Riveill, rapporteurs, pour leurs conseils avisés ;

Philippe Palanque et Vicent Quint, membres du Jury, pour leur écoute attentive ;

L'ESEO, mon employeur, pour m'avoir permis de réaliser ce travail ;

L'équipe InSitu, pour son dynamisme et sa pluridisciplinarité ;

Emmanuelle et Alice, pour leur soutien interactif et collaboratif.

Table des matières

Introduction

Nous élaborons et développons dans cette thèse un modèle dédié à la conception d'espaces de travail interactifs et collaboratifs centrés sur les documents et les instruments. La construction d'un tel modèle est motivée par un triple constat effectué sur les systèmes actuels.

Premier constat – Les environnements interactifs actuels utilisent largement la notion d'application alors que, du point de vue l'utilisateur, les objets d'intérêt sont principalement des documents. La nature composite des documents oblige alors les utilisateurs à jongler entre les applications afin d'éditer les différentes parties d'un document. Cette bascule d'une application à l'autre induit une discontinuité de l'interaction. Il existe des techniques d'adaptation des applications afin de pallier à ces insuffisances, telle que l'utilisation de "plug-ins" ou d'une approche basée sur OLE / OpenDoc. Cependant, elles ne font que repousser le problème sans vraiment le prendre en considération. En particulier, elles ne remettent pas en cause le principe même de la notion d'application.

Deuxième constat – L'interaction dans les systèmes actuels s'articule autour du tandem clavier + souris, réduisant ainsi sa richesse potentielle à des stéréotypes. Par exemple, l'implantation des techniques d'interaction bimanuelle, des "toolglasses" ou de la reconnaissance de traces, est souvent difficile et reste à la charge des applications. Pourtant, de telles techniques ont montré leur intérêt et ne sont proposées, dans le meilleur des cas, qu'au travers de boîtes à outils indépendantes. Ceci ne facilite pas leur intégration dans un espace interactif.

Troisième constat – De nombreuses techniques liées à la collaboration inter-personnelle ont été mises en œuvre dans des collecticiels. Elles permettent la collaboration dans des styles variés, en temps réel ou en temps différé, par le partage d'espaces de travail et de documents. Cependant, les systèmes d'exploitation ne proposent pas de services dédiés à la collaboration si bien que la construction des collecticiels nécessite un effort important. De plus, cette absence de services ne favorise pas l'intégration des techniques de collaboration au niveau de l'espace de travail. Pourtant, une telle intégration suscite un intérêt certain. Par exemple, l'application NetMeeting intègre différents outils de collaboration tels qu'un tableau blanc partagé et d'un salon de discussion ("chat"). Enfin, la collaboration sur un même écran ("single display groupware") est simplement ignorée bien que ce mode de communication possède des aspects très pertinents.

Proposition

Nous proposons une nouvelle approche des espaces de travail interactifs et collaboratifs qui vise à contrer le triple constat énoncé ci-dessus. Cette approche est formalisée par le modèle conceptuel appelé DPI (Document, Présentation, Instrument). Dans ce modèle, les applications disparaissent au profit des trois composants de base, le document, la présentation et l'instrument. Ces composants communiquent les uns avec les autres à l'intérieur de l'espace de travail en fonction des actions et besoins de l'utilisateur.

Le modèle DPI fournit un modèle de composant qui est instancié en un modèle de document basé sur XML, en un modèle de présentation basé sur l'idée de graphe de scène, ainsi qu'en un modèle d'interaction basé sur l'interaction instrumentale.

Ce modèle est affiné pour pouvoir assurer la collaboration. Le verrouillage de propriétés est introduit afin de pouvoir garantir la cohérence de l'interaction en "single display groupware" et en interaction bimanuelle. Le modèle permet également la collaboration à distance en temps réel ou en temps différé. Il précise à cet effet les notions de partage des composants D, P et I par une approche basée sur la réplication et sur la restitution de la conscience mutuelle.

Résultats

Le modèle de composant proposé permet de construire les trois entités D, P et I de manière relativement indépendante et reste très générique. Il est cependant dédié à DPI et ne vise pas l'interopérabilité avec d'autres modèles.

Une boîte à outils expérimentale "OpenDPI"[1] a été développée afin de tester et d'évaluer le modèle. Un ensemble de mini-scénarios y est élaboré afin d'illustrer la mise en œuvre d'instruments divers dans des situations précises. Cette boîte à outils regroupe environ 250 classes soit quelques 18000 lignes de code Java. Elle permet d'élaborer des instruments complexes et des documents simples dans un contexte éventuellement multi-utilisateurs.

Les perspectives de ces travaux consiste à utiliser cette boîte à outils afin de construire un espace de travail basé sur DPI pour un ensemble de tâches suffisamment conséquent. Une telle réalisation permettrait d'évaluer la pertinence de notre modèle en situation réelle. D'autre part, le modèle DPI sera affiné en s'appuyant sur des modèle de document et d'affichage standards.

Structure de la thèse

La thèse est structurée en six chapitres.

Nous situons en premier lieu le contexte de nos travaux. Le chapitre 1 présente les caractéristiques des principaux systèmes interactifs et collaboratifs de la littérature. La mise en évidence de leurs caractéristiques est essentielle pour l'élaboration de notre modèle. Le chapitre 2 fournit les différents fondements retenus tant du point de vue de l'interaction que du point de vue de la collaboration. Ils définiront les principes essentiels du modèle DPI.

Nous développons en deuxième lieu le modèle DPI à partir des caractéristiques et des fondements mis en évidence dans les deux premiers chapitres. Le chapitre 3 aborde la description du modèle d'un point de vue théorique. Les principes fondamentaux sont expliqués et illustrent les relations entre les trois composants de base que sont les documents, les présentations et les instruments. Le chapitre 4 présente la réalisation pratique du modèle théorique. Les différents mécanismes mis en œuvre dans la boîte à outils "OpenDPI" y sont explicités. Différents "mini-scénarios" de conception sont proposés puis leur implantation précisée dans l'annexe A. Ils permettent d'illustrer des aspects spécifiques de DPI et d'affiner le modèle initialement présenté.

Nous analysons en troisième et dernier lieu le problème de la collaboration et, en particulier, des espaces de travail collaboratifs. Nous montrons comment le modèle DPI, dont la

[1] http ://www.eseo.fr/~obeaudoux/opendpi

facette interactive a été présentée dans les chapitres 3 et 4, propose une facette collaborative permettant d'aborder des situations de coopération variées. Le chapitre 5 présente le cas particulier de la collaboration locale et son analogie avec l'interaction bimanuelle. Il propose une solution au problème des actions simultanées basée sur un verrouillage à grain fin et il définit la notion de cohérence de l'interaction. Le chapitre 6 finalise la thèse en présentant le cas de la collaboration à distance. La facette collaborative du modèle DPI y est abordée par le biais d'un scénario de conception illustrant l'idée de "place". La notion de point de partage est ensuite présentée et illustre comment il est possible de partager les composants D, P et I dans des contextes variés, de garantir la cohérence des données et de fournir une conscience mutuelle de l'activité des utilisateurs.

Nous concluons la thèse par une synthèse de l'approche proposée, une discussion sur les apports et les limites du modèle DPI, puis une mise perspective de nos travaux.

Chapitre 1

Évolution des systèmes interactifs et collaboratifs

Contents

Nous faisons dans ce chapitre un état de l'art des systèmes interactifs et collaboratifs que nous avons jugé pertinents vis-à-vis de notre problème. Le but en est de découvrir les caractéristiques essentielles de tels systèmes afin qu'elles puissent être prises en compte par notre modèle. Elles serviront à définir les fondements de DPI qui seront abordés dans le chapitre 2. Cet état de l'art sera également l'occasion de dresser un bilan sur les environnements interactifs utilisés de nos jours et de faire le constat qu'ils prennent peu (voire ne permettent pas de prendre) en considération les avancées faites dans le domaine de l'Interaction Homme-Machine. Ce constat constitue l'une de nos principales motivations.

Le chapitre est organisé autour de cinq sections. La section 1.1 dresse un constat sur l'évolution des systèmes interactifs depuis le Xerox Star. Un tour d'horizon des différentes techniques d'interaction et de visualisation est effectué dans la section 1.2. Il nous permet d'identifier les caractéristiques jugées pertinentes que notre modèle devra prendre en compte. Notre approche étant centrée sur l'idée de document, nous effectuons une description des systèmes et espaces de travail centrés sur les documents dans la section 1.3. Nous présentons plus particulièrement le système OpenDoc qui possède les caractéristiques les plus proches de celles de notre modèle. Enfin, les aspects relatifs à la collaboration sont présentés dans la section 1.4 en étudiant, d'une part, les applications collaboratives et, d'autre part, les espaces de travail collaboratifs. La section 1.5 reformule notre constat et reformule alors non principales motivations.

1.1 Constat

1.1.1 Le Xerox Star

L'ancêtre de nos actuels micro-ordinateurs est incontestablement le Xerox Star. A peine vingt cinq ans après, la puissance de calcul de nos machines s'est considérablement accrue. Les cartes graphiques permettent d'atteindre une grande précision et une importante rapidité de rendu. Les périphériques d'entrée utilisateur deviennent plus variés étendant ainsi le registre de l'interaction au delà de ce qui est permis avec le classique duo clavier - souris. Le progrès technologique est ainsi incontestable.

Nous allons illustrer dans les points qui suivent le peu (voire même l'absence) d'évolution des interfaces Homme-Machine de nos systèmes actuels depuis le Xerox Star.

La rétrospective du Star faite par Johnson et al. (1989) synthétise les différents aspects qui caractérisent ce système. La figure 1.1 montre d'emblée que l'interface du Star est basée sur la métaphore du bureau ("desktop"), métaphore que nous retrouvons sur la quasi totalité des systèmes actuels. Le bureau informatique ressemble ainsi à un bureau réel qui contient diverses fournitures et équipements. Il représente l'espace de travail donnant accès aux projets courants et à diverses ressources. L'écran affiche des icones d'objets familiers tels que des documents, des dossiers, une corbeille, des boîtes aux lettres entrantes et sortantes. Cette approche encourage les utilisateurs à voir les objets du bureau en tant qu'objets physiques. Il est par exemple possible de déplacer ces objets d'un point à l'autre du bureau afin d'organiser son contenu, comme nous le faisons sur un bureau réel. Ce qui est pertinent ici, du point de vue de cette thèse, tient dans le fait que le bureau, ou plus généralement l'idée d'espace de travail, devient l'élément central du système interactif. Sa conception définit *globalement* la façon dont les utilisateurs vont interagir avec le système puisque le bureau constitue son principal point d'entrée. Par ailleurs, l'utilisation intensive des icones va dans le sens d'une pensée "orientée objet" de l'interface (section 2.1.2) et encourage la manipulation directe de ces objets (section 2.1.1). Les périphériques d'entrée du Star sont ceux que nous retrouvons classiquement aujourd'hui sur nos systèmes, à savoir un clavier et une souris (figure 1.2). Cependant, le Star simplifie considérablement l'utilisation de son interface par le concept de commandes *génériques*, lesquelles sont accessibles par le pavé situé

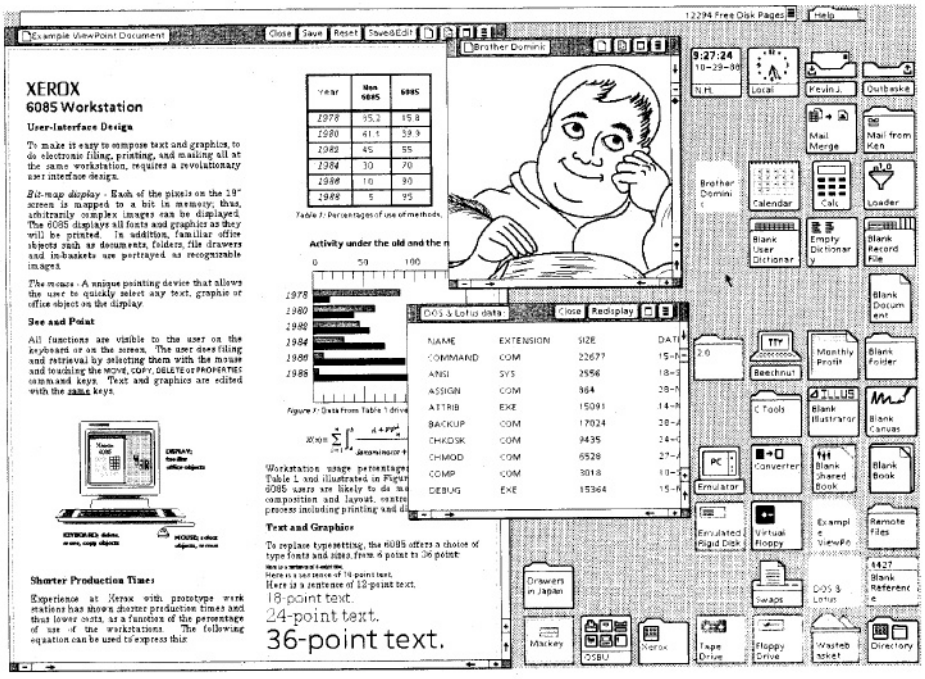


FIG. 1.1 – Interface graphique du Xerox Star – extrait de (Johnson et al., 1989)

à la gauche du clavier (figure 1.2). Une commande générique est une commande qui peut s'appliquer à un large éventail d'objets. Le Star définit six commandes génériques : déplacer ("move"), copier ("copy"), ouvrir ("open"), effacer ("delete"), afficher les propriétés ("prop's") et copier les propriétés ("same"). Ces commandes sont effectivement activées sur un objet en utilisant *conjointement* le pavé gauche du clavier et la souris. Par exemple, pour copier un objet, l'utilisateur sélectionne l'objet à copier avec la souris, appuie sur la touche "copy" du pavé gauche et indique la destination de la copie de l'objet avec la souris. La commande ainsi que l'interaction qui permet son déclenchement restent identiques quel que soit l'objet à copier : il peut s'agir d'un document que l'utilisateur copie depuis le bureau, ou une portion de texte copié à l'intérieur d'un document ou d'un document à l'autre. Chaque type d'objet interprète ainsi à sa manière la commande générique à réaliser. Le choix des commandes génériques doit se faire de manière précise. Il permet de réduire de manière non négligeable le nombre de commandes requises par le système. Par exemple, nous pourrions penser que la commande "imprimer" est générique car elle peut s'appliquer à presque tous les types d'objet. Cependant, cela peut être considéré comme une erreur de conception puisque cette commande peut-être factorisée avec la commande "déplacer" : le déplacement de l'objet vers une imprimante effectue implicitement la commande d'impression de l'objet par l'imprimante visée. Nous reprenons cette notion de généricité dans le modèle de l'interaction instrumentale (section 2.1.4). D'autre part, l'utilisation conjointe du pavé gauche du clavier et de la souris est une forme, certes limitée, d'interaction bimanuelle (section 1.2.1.3).

L'utilisation d'une approche métaphorique induit naturellement que les objets purement informatiques soient masqués aux utilisateurs autant que faire se peut. Dans un système basée sur la métaphore du bureau, les utilisateurs manipulent principalement des fichiers et les applications ne sont qu'un moyen d'arriver à cette fin : l'objet d'intérêt est donc principalement le document. Ainsi les utilisateurs "ne lancent pas un éditeur de texte", mais "ouvrent un document texte". Le système du Xerox Star connaît le type de chaque fichier

FIG. 1.2 – Les périphériques d'entrée du Star – extrait de www.digibarn.com/ friends/ curbow/ star/keyboard

et notifie alors l'application dédiée à ce type quand l'un d'entre eux doit être ouvert. En procédant de la sorte, le concept d'application est masqué alors que celui de document est mis au premier plan. La visualisation iconique des fichiers informe l'utilisateur sur le type du fichier, alors que le nom du fichier est renseigné par le label associé à l'icone. L'organisation par dossiers et noms de fichiers est complétée par une organisation *spatiale* : un fichier, ainsi que tout objet accessible associé à un icone, peut être placé indifféremment dans un dossier ou directement à un endroit spécifique du bureau, comme il est fait dans un bureau réel. Les concepteurs du Star mettent en avant le fait que l'application de première importance est l'éditeur de document. Les autres applications existent principalement en tant qu'outils ayant des rôles variés mais dont la destination finale est presque toujours un document – mais qui peut parfois n'intéresser que le système et non les utilisateurs. En conséquence, la plupart des applications sont *intégrées* dans l'éditeur de document et opèrent à l'intérieur de cadres réservés dans les documents. Les applications qui n'y sont pas intégrées fournissent un support pour le transfert de leurs données dans les documents. Ceci préfigure d'une manière là encore assez surprenante les systèmes centrés sur les documents tels que OpenDoc (section 1.3).

Enfin, le concept de ressources distribuées est largement présent dans l'interface du Star. L'idée de pouvoir connecter une station de travail personnelle à un réseau local et d'y attacher diverses ressources telles que des serveurs de fichiers, des serveurs de bases de données ou d'imprimantes, a été concrétisée par le Star. Dans les systèmes qui l'ont précédé, l'architecture était soit "stand alone" (ordinateur personnel) soit centralisée (terminaux reliés à un serveur jouant à la fois le rôle de serveur d'applications et de serveur de fichiers). La combinaison des ordinateurs personnels et des réseaux locaux permet de profiter de l'avantage des deux approches en s'affranchissant, du moins partiellement, de leurs inconvénients. Mais cela va également jouer un rôle qui n'était peut-être pas celui attendu : la possibilité d'utiliser un ordinateur personnel et donc un espace de travail personnel à des fins de collaboration. Ainsi l'accès à un serveur de fichiers permet dans une certaine mesure de partager des documents, l'accès à un serveur de base de données permet de travailler conjointement sur un référentiel commun de données, et l'utilisation du courrier électronique permet de collaborer d'une manière plutôt informelle. L'aspect collaboratif joue de nos jours un rôle plus que prépondérant, comme il sera discuté dans la section 1.4.

1.1.2 Les systèmes actuels

Il est intéressant de comparer les interfaces actuelles avec celles du Star afin de caractériser leurs évolutions. Nous avons choisi les deux systèmes d'exploitation les plus populaires et représentatifs de cette évolution, à savoir Windows et MacOS

Tout d'abord, les systèmes actuels s'appuient tous sur la métaphore du bureau (figure 1.3). Concernant Windows, c'est à partir de la version Windows95 que cette métaphore est ap-

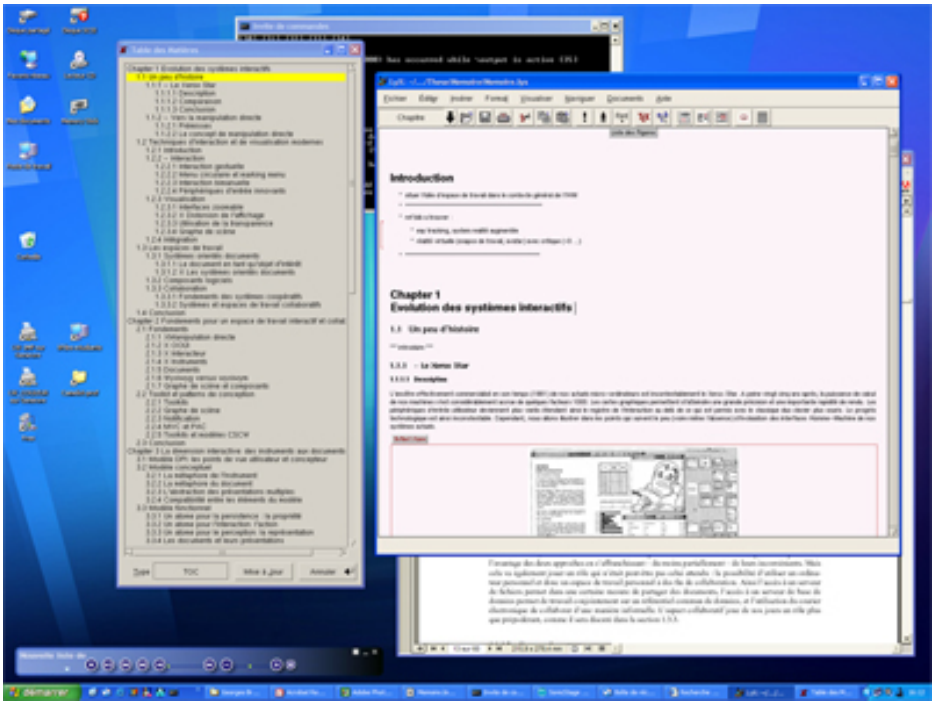
parue. Les versions antérieures telles que Windows3.1 étaient radicalement orientées application : l'espace de travail était constitué de groupes d'applications accessibles par leurs fenêtres associées, ces groupes contenant des icones représentant les applications. L'interaction au travers de l'espace de travail était ainsi limitée au strict nécessaire, *i.e.* au lancement d'une application ! Les versions de MacOS antérieures à MacOS X étaient très largement inspirées de l'interface du Star. Un constat un peu surprenant : Windows XP et MacOS X utilisent un "tableau de bord", c'est à dire un élément du bureau qui contient principalement les points d'accès aux applications au travers des icones qui permettent leur lancement et des "boutons" permettant d'accéder aux applications ouvertes et leurs fenêtres associées. Ceci est sûrement assez révélateur d'une tendance à mettre l'application au premier plan, au détriment d'une approche centrée sur les documents.

Tout comme dans le Star, nos systèmes actuels utilisent le classique tandem clavier - souris. Cependant, l'utilisation de commandes génériques se limite typiquement aux commandes "copier /coller" et "ouvrir". Elle s'est étrangement réduite à l'idée de "modificateurs". Un modificateur est une touche du clavier qui permet de modifier le comportement par défaut du geste "glisser-déposer". Typiquement, la touche "ctrl" permet de spécifier une copie plutôt qu'un déplacement simple. Cependant, l'utilisation des modificateurs n'est pas nécessairement homogène entre l'espace de travail et les différentes applications, et n'est pas toujours très simple à comprendre par l'utilisateur car un modificateur peut agir différemment selon le contexte. Globalement, le gain apporté par les commandes génériques a disparu. Ceci est révélateur de la tendance actuelle de nos systèmes qui consiste à fournir toujours plus de fonctionnalités d'une version d'une application à la suivante. L'approche du Star était davantage basée sur une *factorisation* des fonctionnalités quand leur nombre s'accroît, approche qui reste possible en considérant le concept de généricité des commandes. L'activation des commandes génériques par le pavé gauche du clavier reflétait l'efficacité de l'interaction bimanuelle. Notons que English et al. (1967) illustrent l'intérêt de l'interaction bimanuelle basée sur l'utilisation simultanée du clavier par la main non dominante et de la souris par la main dominante. Les systèmes actuels ne mettent en œuvre – ni ne permettent facilement de mettre en œuvre – les techniques d'interaction bimanuelle. Pourtant l'intérêt majeur de l'interaction bimanuelle est l'efficacité de l'interaction, en permettant par exemple d'atténuer sensiblement les va-et-vient entre le clavier et la souris.

L'application reste aujourd'hui l'élément central dans les interfaces graphiques alors que le document constitue presque toujours l'objet d'intérêt principal. En particulier, la notion de document composite ne reprend pas le principe spécifié par le Star, *i.e.* l'intégration des applications dans "l'éditeur de document". Ceci amène les systèmes actuels à proposer plusieurs manières de composer un document. Par exemple, l'utilisateur travaillant sous Windows XP peut insérer dans un document texte un graphique de plusieurs manières : par un copier/coller du graphique depuis l'application d'édition du graphique vers le traitement de texte, par une importation du fichier qui contient le graphique et qui a été préalablement exporté par l'éditeur du graphique (dans un format que l'utilisateur devra choisir), ou par insertion d'un "objet OLE[1]" qui permet de travailler avec l'éditeur graphique à l'intérieur même de la fenêtre du traitement de texte. Si la dernière solution rend un peu plus transparentes les applications, les deux premières solutions ne vont pas dans le même sens. Dans tous les cas, l'utilisateur devra choisir l'une des trois méthodes en fonction de ses propres connaissances et des possibilités et contraintes fournies par les applications concernées. Un autre point qui illustre l'orientation application des systèmes actuels concerne le nommage des fichiers : le nom du fichier contient une "extension" située à la fin du nom, cette extension servant à identifier le type du document et donc à spécifier indirectement quelles applications peuvent être utilisées pour éditer ou visionner le document.

Les systèmes actuels s'intègrent largement dans des réseaux. L'intégration d'ordinateurs personnels au sein d'un réseau est souvent préférée à l'utilisation d'une architecture client /

[1] Object Linking Embedding



(a) WIndowsXP

(b) MacOS X

FIG. 1.3 – Interfaces graphiques des systèmes d'exploitation

serveur pure (à base d'un serveur connecté à plusieurs terminaux), ceci pour des raisons de confort pour l'utilisateur. Les applications de type "collecticiel" apparaissent depuis quelques années et modifient peu à peu la façon dont les ordinateurs personnels sont utilisés : elles rendent possible la collaboration médiatisée par l'ordinateur (Shneiderman, 1998, chapitre 14). Cependant, nous pouvons être surpris par le fait que les systèmes d'exploitation ne proposent pas de services pour les collecticiels, ces derniers étant alors réalisés par les applications elles-même. Un exemple significatif : la gestion de courrier électronique est présente au niveau de l'espace de travail du Star par le simple fait que des icones "courrier entrant" et "courrier sortant" peuvent y être positionnées, donnant ainsi un accès simple au courrier électronique – sans avoir à lancer l'application dédiée. En particulier, pourquoi les messages entrant et sortant ne sont-ils pas considérés comme des documents à part entière et les boîtes aux lettres comme des dossiers ? Cela provient probablement du peu de considération portée à l'idée de document ainsi qu'à l'idée d'espace de travail, et d'une pensée toujours centrée sur le concept purement informatique d'application.

1.1.3 Conclusion

Les descriptions précédentes mettent en lumière le paradoxe suivant : si les performances des machines actuelles se sont largement accrues pendant ces 25 dernières années (niveau matériel), les interfaces Homme-Machine proposées n'ont guère évoluées (niveau logiciel). A certains égards, nous pouvons même constater qu'elles ont baissé en qualité, *i.e.* en ce qui concerne la simplicité et le côté intuitif de leur utilisation. Ce qui est pour le moins surprenant, c'est que les performances matérielles, difficilement prévisibles il y a vingt cinq ans, ne servent finalement que très peu les interfaces actuelles. La quasi incapacité à utiliser d'autres périphériques que le tandem clavier / dispositif de pointage est à cet égard tout à fait significatif. Par exemple, l'incapacité à utiliser deux dispositifs de pointage distincts rend complexe la mise en œuvre des techniques d'interaction bimanuelle. Un autre exemple concerne l'utilisation d'un stylet conjointement à une tablette graphique : elle est possible dans les espaces de travail actuels mais n'induit pas nécessairement plus d'attrait que l'utilisation d'un dispositif de pointage plus traditionnel telle que la souris. En effet, la seule différence réside dans le fait que la position du stylet est une position absolue (*i.e.* la position du stylet par rapport à la tablette graphique) et non relative (*i.e.* le déplacement de la souris). Par contre, l'utilisation de ce même stylet dans une *application* de dessin telle que Painter Classic offre des possibilités qu'une simple souris ne saurait fournir : la tablette graphique mesure non seulement la position absolue du stylet (deux coordonnées X et Y), mais aussi sa position angulaire (trois angles Ax, Ay et Az donnant à la position angulaire du stylet par rapport aux trois axes X, Y et Z) ainsi que la pression exercée sur la pointe du stylet. Un utilisateur averti peut alors réaliser des œuvres picturales d'une manière assez expressive qu'il serait difficile d'égaler en utilisant une souris classique. Le fait que le progrès matériel n'ait principalement bénéficié qu'aux applications et non aux espaces de travail constituera une revendication quant aux espaces de travail centrés sur les documents et non sur les applications (section 1.3).

Nous pouvons ainsi constater que le gain matériel ne profite que (trop) peu à l'espace de travail lui-même, et profite finalement *principalement* aux applications. Ceci illustre que les systèmes actuels sont centrés sur les applications.

Le constat que nous faisons ici constitue, d'une certaine manière, la motivation principale de cette thèse : réfléchir sur les fondements d'une interface graphique – ou espace de travail – de nouvelle génération qui permettrait de prendre en considération les techniques d'interaction issues du monde de la recherche ainsi que les différents schémas de collaboration offerts par les applications développées par les laboratoires de recherche universitaires ou industriels. Notons que ce travail a déjà été amorcé, pour la facette interaction, au travers du développement de l'application d'édition de réseaux de Pétri colorés, CPN 2000

(Beaudouin-Lafon & Lassen, 2000). Cette application peut-être vue comme un espace de travail dédié à des tâches très spécifiques liées aux réseaux de Pétri colorés et non une application qui s'intègre à un espace de travail pré-existant.

1.2 Techniques d'interaction et de visualisation

Différentes techniques d'interaction et de visualisation fournissent des alternatives complémentaires de celles des systèmes WIMP[2] actuels. Nous effectuons dans cette section un rapide et non exhaustif tour d'horizon des techniques que nous avons jugé comme étant les plus significatives des interfaces "post-WIMP". Nous montrons également que, si ces techniques sont pertinentes, leur utilisation conjointe n'est pas aisée.

1.2.1 Interaction

1.2.1.1 Interaction gestuelle

Le système SketchPad utilisait déjà le tracé de formes géométriques à main levée *via* le stylet optique. La forme tracée était analysée afin de substituer au tracé une forme géométrique régulière. Cette reconnaissance de tracés était donc une interaction gestuelle permettant d'effectuer des actions de construction d'objets graphiques. Il est possible d'étendre le champs d'un tel principe de telle sorte que des actions autres que construire des objets graphiques puissent être déclenchées par la reconnaissance du tracé, *i.e.* du "geste".

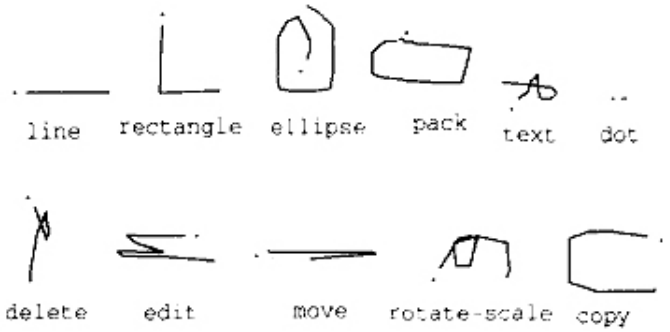


FIG. 1.4 – Un exemple de commandes gestuelles – extrait de (Rubine, 1991)

Le système GRANDMA a été utilisé pour développer un Gesture-based Drawing Program (GDP) proposé par Rubine (1991) et permet de spécifier des actions sur les objets par le biais de gestes. Un geste a une *signature* caractéristique de l'action associée ; ils sont d'abord physiquement "tracés" sur l'écran *via* le dispositif de pointage (souris ou stylet), analysés par le système puis, lorsqu'une signature est reconnue, la trace est effacée de l'écran et l'action associée est invoquée sur l'objet cible par la trace. La figure 1.4 donne un exemple d'un vocabulaire de signatures permettant la construction et l'édition de formes géométriques diverses.

L'interaction à base de gestes, ou interaction gestuelle, est intuitive de par l'aspect naturel lié de la gestuelle. Elle entre tout à fait dans le cadre de la manipulation directe puisque les commandes sont "dessinées" directement sur les objets d'intérêt. De plus, l'interaction gestuelle peut être tout à fait significative pour certains types de document, comme des partitions musicales (Rubine, 1992).

[2] Window, Icon, Menu, Pointer

1.2.1.2 Menu circulaire et “marking menu”

Une autre technique qui utilise l’interaction à reconnaissance de gestes – plus précisément de “marques” – concerne les éléments d’interaction précis que sont les “marking menus”, extension des menus circulaires.

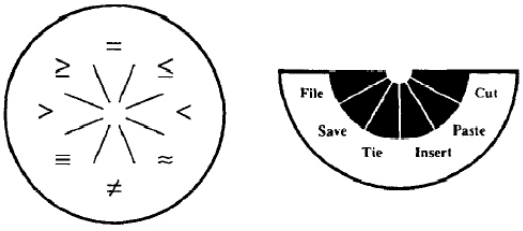


FIG. 1.5 – Exemples de menus circulaires – extrait de (Callahan et al., 1998)

Les menus circulaires (“pie menus”, figure 1.5) sont des menus qui se différencient des menus linéaires par leur géométrie : les éléments constitutifs sont répartis circulairement plutôt que linéairement. Callahan et al. (1998) montrent que l’utilisation de ce type de menu apporte un gain de rapidité et minimise les erreurs de sélection, tout en restant subjectivement équivalent aux menus linéaires traditionnels.

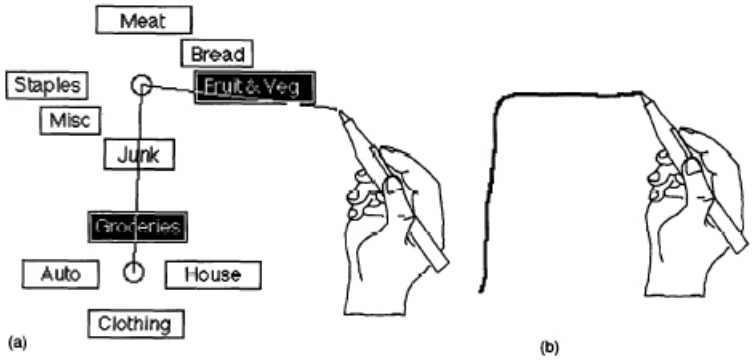


FIG. 1.6 – Exemples de “marking menu” – extrait de (Kurtenbach & Buxton, 1993)

Les “marking menus” sont une extension des menus circulaires au niveau de l’interaction (Kurtenbach & Buxton, 1993). Lorsque la sélection d’un élément de menu est amorcée, le système attend un bref laps de temps avant d’afficher ledit menu. Si l’utilisateur attend que ce laps de temps soit écoulé, le menu s’affiche et la sélection de l’élément (et éventuellement des sous-éléments dans le cas des menus hiérarchiques) s’effectue normalement (figure 1.6-a). Toutefois, si l’utilisateur effectue le geste complet de sélection du ou des éléments pendant ce laps de temps, le ou les menus ne s’affichent pas et le système tente de reconnaître le geste. Si le geste correspond effectivement au chemin de la sélection d’un élément (la “marque”), cet élément de menu est activé (figure 1.6-b). Cette approche est intuitive car elle n’impose pas d’apprendre un “raccourci” différent de l’interaction habituelle et permet de passer au mode expert simplement en accélérant le geste, ce qui est naturel.

Les “marking menus” sont de plus intéressants car ils conservent l’esprit d’une interaction directe (les menus sont contextuels) et restent homogènes avec l’interaction gestuelle. Remarquons cependant que l’utilisation conjointe de l’interaction gestuelle et des “marking menus” doit être *consistante* et non ambiguë.

1.2.1.3 Interaction bimanuelle

Guiard (1987) propose une réflexion puis une modélisation de la chaîne kinesthésique de l'action bimanuelle de l'humain. L'auteur illustre le fait que l'homme utilise principalement, dans les actes de sa vie courante, ses deux mains de manière simultanée et complémentaire, et non une seule main. Ceci illustre l'intérêt de mettre en œuvre cette forme d'interaction au niveau des interfaces utilisateur. Cet intérêt renforce la possibilité de satisfaire aux principes de la manipulation directe :

– L'action bimanuelle étant inhérente à l'humain, ce dernier peut maîtriser plus précisément le système.
– L'efficacité de l'utilisateur est nécessairement accrue (imaginons par exemple que nos actions quotidiennes soient réalisées avec une seule main...).
– L'action physique induite renforce le plaisir d'utiliser le système et la confiance de l'utilisateur à en rester maître.
– Cela enrichit le "vocabulaire" des actions physiques.
– Certaines opérations peuvent alors être plus rapides et efficaces.

L'aspect de non symétrie de la chaîne kinesthésique est important, et surtout l'aspect corollaire de la complémentarité des deux mains. Notamment, la main non dominante (*i.e.* la main gauche pour un droitier) possède davantage un rôle lié au positionnement de l'objet du contexte de l'action alors que la main dominante (*i.e.* la main droite pour un droitier) effectue une action (fine) sur un objet du contexte.

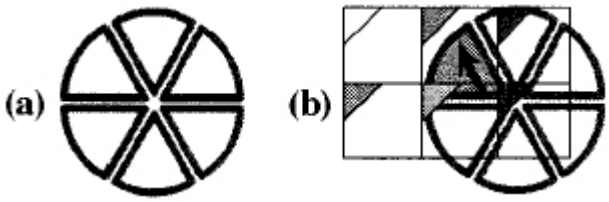


FIG. 1.7 – Un exemple de "toolglass" – extrait de (Bier et al., 1993)

Cet aspect de l'interaction bimanuelle est mis à profit par les "toolglasses" (Bier et al., 1994). Cette technique d'interaction s'inspire de la métaphore des "traceurs" des dessinateurs industriels tels que les trace-cercles ou les trace-lettres. Le "toolglass" est manipulé par la main non dominante qui le positionne au dessus de l'objet ciblé. La main dominante effectue alors une interaction appelée "clic au travers" ("click-through") qui applique l'une des actions disponibles sur la toolglass vers l'objet ciblé. Par exemple, une toolglass "palette de couleurs" permet d'appliquer une couleur (figure 1.7) : la toolglass (b) est amené au-dessus de l'objet (a) par la main non dominante (typiquement *via* un trackball) et le curseur est positionné à la fois sur la partie de l'objet ciblé et sur la couleur à lui appliquer (typiquement *via* une souris). Un "clic au travers" permet alors d'appliquer la couleur voulue sur l'objet pointé. L'efficacité qu'apporte cet outil d'interaction vient du fait que l'utilisateur spécifie au travers d'un geste bimanuel unique l'objet cible *et* le paramètre associé à l'action (une couleur dans l'exemple). Cette technique a été évaluée, pour une tâche particulière consistant à créer différents segments colorés reliant des points qui apparaissent progressivement sur l'écran, comme offrant un gain d'efficacité de 40 % (Kabbash et al., 1994).

Kurtenbach et al. (1997) fournissent un exemple d'application basée sur l'interaction bimanuelle. Ils illustrent les avantages de l'utilisation conjointe de la transparence, de l'interaction bimanuelle et d'une tablette graphique. L'application est une extension du logiciel de dessin artistique *StudioPaint* pour laquelle les principes interactifs mis en place dans

l'application de prototypage *T3* ont été repris (aux fonctionnalités compatibles près). Cette application montre l'intérêt d'*intégrer* différentes techniques d'interaction pour la réalisation d'un ensemble de tâches défini (section 1.2.3).

1.2.1.4 Périphériques d'entrée

Les systèmes actuels n'ont guère évolué depuis le Star en ce qui concerne les périphériques d'entrée utilisateur : il s'agit presque toujours d'un clavier et d'une souris. Il est pourtant intéressant de pouvoir étendre la richesse de l'interaction en proposant des périphériques plus évolués. Un exemple récent illustre ce propos : la molette de nos souris actuelles a étendu l'efficacité de certaines interactions génériques. Par exemple, elle donne la possibilité de faire défiler un document à l'intérieur de sa fenêtre, de faire un zoom avant ou arrière, ou bien régler le volume de sortie audiophonique, selon des interactions plus directes. Il n'est en effet plus nécessaire d'interagir avec un "widget" spécifique, telle qu'une barre de défilement pour le défilement du document.

Le trackball sensitif et la souris sensitive de Hinckley & Sinclair (1999) sont des améliorations des classiques "trackball" et souris à molette. Ces deux dispositifs de pointage apportent une détection du toucher : le trackball sensitif détecte tout effleurement de la boule du trackball, et la souris sensitive détecte l'effleurement de ses différentes parties constitutives (la région en contact avec la paume de la main, celle en contact avec le pouce, la molette et le bouton gauche). La détection de l'effleurement permet alors de prévoir l'*intention* de l'utilisateur et donc d'*adapter* l'interface en conséquence. Par exemple, les barres à outils situées sous les barres de menu de la grande majorité des applications sont affichées en permanence, tendant ainsi à prendre une place excessive par rapport à l'objet d'intérêt – typiquement un document. L'utilisation d'une souris sensitive permet d'éliminer ce problème en n'affichant les barres à outils que lorsque la souris est effleurée. Les "toolglasses" peuvent également bénéficier de cette amélioration : ils ne sont affichés que lorsque le trackball sensitif lié à la main non dominante est effleuré.

FIG. 1.8 – Le système PHANToM – extrait de http ://www.sensable.com

Les dispositifs précédents permettent des extensions intéressantes des interactions offertes par les espaces de travail et leurs applications. Ils visent à accroître les capacités offertes à l'utilisateur pour interagir avec les objets de l'interface. Par contre, ils ne fournissent pas de retour d'information – outre le retour visuel ou/et sonore de l'interface elle-même – sur l'état de l'action en cours par exemple. Des dispositifs tels que le joystick à retour de force, la souris " tactile" à retour de vibration (souris Logitech "iFeel"), ou le stylet à retour de force (interface "haptic", figure 1.8) vont dans ce sens : ils fournissent une information

de retour (feed-back) physique typiquement sous la forme d'une réaction mécanique de type force réactive ou vibratoire. Cette approche fournit une information kinesthésique qui peut être mise à profit pour naviguer dans l'interface et en percevoir son état, ainsi que pour affiner l'interaction sur un objet de l'interface. Par exemple, Miller & Zeleznik (1998) proposent un retour de force pour la manipulation des éléments de l'interface de X-Window en simulant la perception des bords des fenêtres, des menus et des icones, comme si ces derniers possédaient une fine épaisseur.

1.2.2 Visualisation

1.2.2.1 Interfaces "zoomables"

La métaphore du bureau[3] est intéressante du fait que le travail est principalement organisé, dans la vie réelle, autour de la notion de bureau. Cependant, cette métaphore limite naturellement le champ spatial offert aux utilisateurs. Cette limite peut être surmontée par l'utilisation d'une interface "zoomable", c'est-à-dire une surface à deux dimensions semblable à celle du bureau, mais sur laquelle il est possible de se déplacer (action de type "pan" pour panoramique) ainsi que de s'en s'éloigner ou de s'en rapprocher (action de type "zoom"). Perlin & Fox (1993) ont créé le premier modèle pour implémenter des interfaces zoomables avec le système Pad. La surface du "Pad" est un plan infini à deux dimensions dans lequel l'utilisateur navigue par une interaction de type "pan & zoom" : il peut s'y déplacer par défilement plan selon les axes X et Y et, éventuellement simultanément, s'y déplacer par zoom avant ou arrière afin d'accéder à l'information disséminée sur la surface. La boîtes à outil de développement d'interfaces zoomables "Pad++" (Bederson & Hollan, 1994) , puis "Jazz" (Bederson et al., 2000) se sont succédées à celle du Pad.

Cette technique de navigation s'avère très efficace pour se déplacer d'un point de départ à un point d'arrivée éloigné (la distance étant mesurée relativement à l'espace "zoomable" et non à l'écran sur lequel est projeté cet espace). Le chemin le plus court (en temps) entre le point de départ et le point d'arrivée n'est plus une ligne droite mais correspond à une interaction consistant à effectuer un zoom arrière ("zoom out"), un déplacement panoramique ("pan") puis un zoom avant ("zoom in") (Furnas & Bederson, 1995). La loi de Fitts s'avère toujours vérifiée à l'intérieur d'un espace "zoomable", et donc pour des distances de navigation nettement plus importantes que celles offertes par les espaces traditionnels (Guiard et al., 2001).

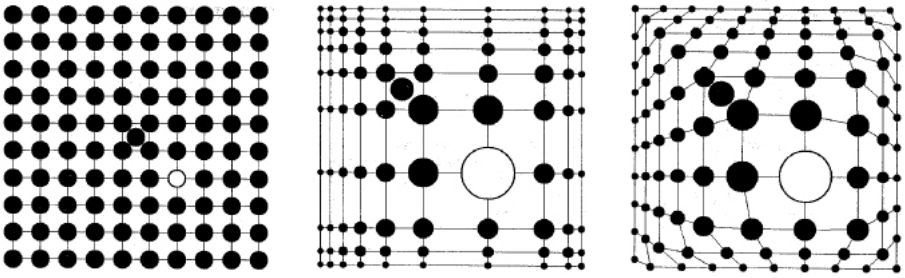
La technique du "pan & zoom" est intéressante car elle illustre bien comment une technique de visualisation peut impliquer un changement assez radical de l'interaction, et qu'il est ainsi difficile de séparer les techniques d'interaction des techniques de visualisation.

1.2.2.2 Distorsion de l'affichage

L'ensemble des techniques de distorsion de l'affichage, si elles changent moins radicalement l'interaction que la technique du "pan & zoom", n'en demeure pas moins intéressant. Ces techniques permettent principalement de visualiser une information particulière à l'intérieur d'un espace de dimensions réduites et contenant un nombre d'informations important. Nous présentons ici très sommairement deux techniques de distorsion, le Fisheye et de défilement stationnaire.

Le "Fisheye" est un objectif utilisé en photographie et qui permet la prise de photographies en très grand angle. L'utilisation d'un tel objectif produit inévitablement un effet de

[3] Le terme de bureau fait référence ici au mot anglais "desktop", c'est à dire au bureau en tant que meuble, par opposition au bureau (en anglais "office") qui fait référence à la pièce ou lieu de travail.



(a) Graphe d'origine (b) Transformation cartésienne (c) Transformation polaire

FIG. 1.9 – Distorsion par Fisheye – extrait de (Sarkar & Brown, 1992)

distorsion : l'image est grossie vers son centre et devient rétrécie sur sa périphérie. L'idée d'utiliser un Fisheye pour la visualisation d'information permet ors d'effectuer des zooms sur une partie de l'information sans pour autant perdre le *contexte* (Furnas, 1986). Il s'agit en quelque sorte d'un zoom non linéaire. Sarkar & Brown (1992) présentent deux types de transformations Fisheye : la transformation cartésienne et la transformation polaire. La figure 1.9 montre un graphe symétrique (a) pour lequel ces deux transformations ont été respectivement appliquées : une transformation cartésienne (b) centrée sur le nœud blanc, puis une transformation polaire (c) équivalente.

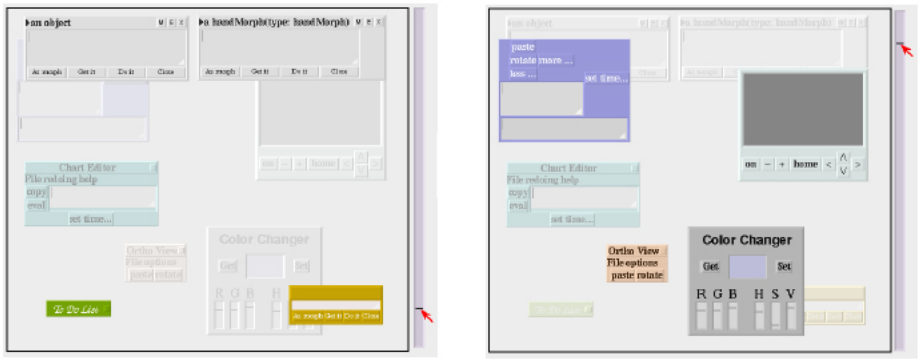


FIG. 1.10 – Défilement stationnaire basé sur l'atténuation de couleur – extrait de (Smith & Taivalsaari, 1999)

Le défilement, généralement réalisé par le biais d'un composant de type barre de défilement, permet de faire défiler le contenu d'un espace afin de se focaliser sur une portion particulière de cet espace. Un tel défilement consiste alors à déplacer, relativement à l'écran, tous les éléments présents dans l'espace. Cette technique permet d'afficher un espace plus grand que l'écran sur lequel il s'affiche. Cependant, elle a pour principal défaut de faire perdre le contexte général au profit de la portion de l'espace effectivement affichée. Smith & Taivalsaari (1999) ont défini la technique de défilement stationnaire qui permet de faire "défiler" le contenu d'un espace sans changer sa position relative à l'écran. Le défilement stationnaire consiste à *substituer* la modification de la position des éléments de l'espace par une modification d'une autre grandeur visuelle de ces éléments : la barre de défilement n'agit alors plus sur ces positions mais sur la grandeur de substitution. La figure 1.10 montre un défilement stationnaire basé sur l'atténuation des couleurs des éléments de l'interface. A chaque élément est associé une "hauteur" qui est visuellement représentée par une atténuation des couleurs : lorsque la barre de défilement "s'éloigne" d'un élément, la

couleur de ce dernier est atténuée d'une manière proportionnelle à l'éloignement.

1.2.2.3 Fenêtrage

Les systèmes de fenêtrage actuels restent basés sur les principes présents dans le Star : les fenêtres peuvent se déplacer, être dimensionnées et se superposer. Cependant, d'autres approches peuvent compléter cette décomposition de l'affichage en fenêtres superposables.

Kramer (1994) propose une alternative aux fenêtres superposables : les "translucent patches". Un translucent patch est une zone non nécessairement rectangulaire et transparente qui permet l'organisation de représentations présentes sur l'espace de travail. L'utilisateur effectue une sélection d'une zone de l'espace afin de créer un patch qui contiendra alors ce qui a été sélectionné. Les différents "patches" peuvent être déplacés librement sur l'espace et se superposer, être agrandis, détruits ou dissous (le contenu revient dans l'espace de travail). L'aspect innovant de cette approche réside principalement dans le fait que l'information n'est plus "enfermée" dans un cadre strict, mais peut être prise depuis un endroit, modifiée, comparée par superposition, etc. Cette approche va dans un sens assez semblable à la suppression de la notion d'application qui conduit à associer une fenêtre à une application. Elle étend également le champ des systèmes orientés documents qui, s'ils réduisent la taille du cadre au niveau des éléments constitutifs des documents (texte, graphiques, tableau, etc.), ne les font pas disparaître. Dans l'approche proposée dans cette thèse, nous définirons un grain de l'affichage très fin et qui n'enferme pas nécessairement une représentation dans un cadre rigide.

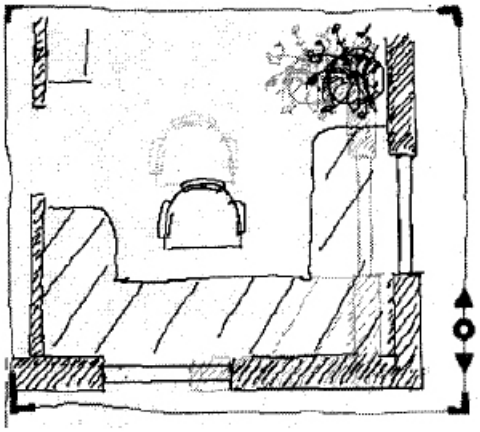
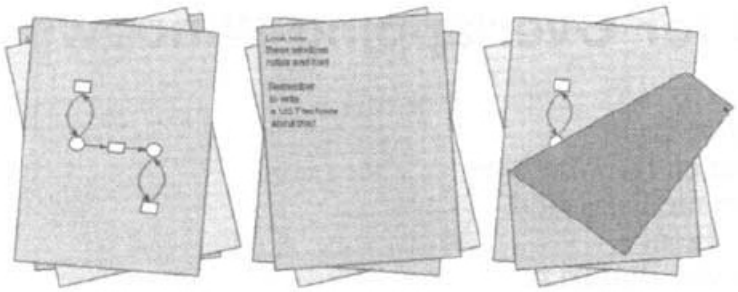


FIG. 1.11 – Affichage de l'historique d'un "patch" – extrait de (Genau & Kramer, 1995)

L'étude de Harrison et al. (1995) illustre l'intérêt d'utiliser la transparence dans les interfaces graphiques d'un point de vue intentionnel (non uniquement visuel). L'idée centrale est de jouer sur la transparence des objets afin d'améliorer la focalisation de l'attention de l'utilisateur sur certains d'entre eux et de mettre en arrière plan ceux qui en perturberaient la perception. Le système basé sur les "translucent patches" a été étendu afin de "voir" l'historique de contenu d'un patch en utilisant un effet de transparence (Genau & Kramer, 1995). La figure 1.11 illustre cet effet et montre le côté intuitif d'une telle approche, le contexte "passé" étant affiché par superposition et transparence, et le focus étant conservé sur l'état courant du croquis.

Beaudouin-Lafon (2001) fournit plusieurs techniques visant à améliorer la gestion du positionnement des fenêtres superposées. Par exemple, les fenêtres peuvent être regroupées dans un groupe de fenêtres à onglets, chaque onglet affichant complètement son label

(a) Zoom sur onglet



(b) Piles irrégulières

FIG. 1.12 – Deux techniques d'affichage de fenêtres empilées – extrait de (Beaudouin-Lafon, 2001)

lorsque le pointeur de la souris se trouve au-dessus (figure 1.12-a). Elles peuvent également être regroupées en piles irrégulières (effet de rotation) et peuvent également être tournées temporairement afin de visualiser la fenêtre suivante (figure 1.12-b).

Les fenêtres élastiques de Kandogan & Shneiderman (1996) ont été conçues selon trois principes : une organisation hiérarchique des fenêtres en groupe, des opérations sur des groupes de fenêtres et une répartition de fenêtres sur l'écran sans superposition. Le regroupement hiérarchique des fenêtres est indiqué en changeant graduellement la couleur du bord de chaque fenêtre en fonction de son niveau hiérarchique. De plus, les fenêtres sont spatialement reparties en fonction de cette hiérarchie. Les opérations tel que le dimensionnement, l'ouverture, la fermeture ou la maximisation, peuvent être appliquées un à groupe de fenêtres, facilitant alors la manipulation de la hiérarchie des fenêtres. Enfin, les fenêtres (d'un groupe) sont dites élastiques car elles sont dimensionnées d'une manière proportionnelle à la dimension du groupe.

Ces différents exemples illustrent que les systèmes de fenêtrages habituels peuvent être sensiblement améliorés tout en restant basés sur les fenêtres superposées. Cependant ces techniques imposent des contraintes au système gérant l'affichage, notamment la possibilité d'appliquer des transformations et des couleurs transparentes aux objets de l'interface.

1.2.3 Intégration

Les techniques d'interaction, de visualisation et de fenêtrage montrent leur intérêt dans différents contextes. Leur utilisation conjointe semble alors pertinente dans la conception d'un espace de travail de nouvelle génération. Cependant, le choix des techniques à retenir effectivement parmi un panel assez large n'est pas trivial : l'interface proposée doit rester globalement consistante, ce qui n'est pas immédiat puisque les différentes techniques mentionnées plus haut ont toutes été pensées *indépendamment* les unes des autres.

Le premier exemple d'intégration des différentes techniques pré-citées concerne l'extension des principes interactifs mis en place dans le prototype d'application "T3" à l'application StudioPaint (Kurtenbach et al., 1997). L'application ainsi étendue utilise comme dispositifs d'entrée un "puck" pour la main non dominante, *i.e.* une souris se déplaçant

sur une tablette graphique et munie d'un petit viseur transparent permettant une visée précise, conjointement à un stylet pour la main dominante et se déplaçant sur la même tablette graphique que le "puck". Les interactions *via* ces périphériques permettent, d'une manière directe, le déplacement, la rotation, le dimensionnement, et le zoom. Elles mettent en jeu les techniques d'interaction tels que les "marking menus", des "toolglasses", les guides curvilignes transparents ("sweeps") le tout dans un contexte graphique utilisant largement la transparence. L'utilisation de toutes ces techniques permet d'atteindre les trois buts initialement fixés :

1. Maximiser l'espace pris par les objets d'intérêt (dessins et graphiques artistiques) par rapport aux objets de l'interface graphique (widgets).
2. Éviter que l'utilisateur soit contraint de dévier son attention visuelle des objets d'intérêt.
3. Accroître les degrés de manipulation et de confort des périphériques d'entrée (utilisation de la position angulaire des dispositifs de pointage par exemple).

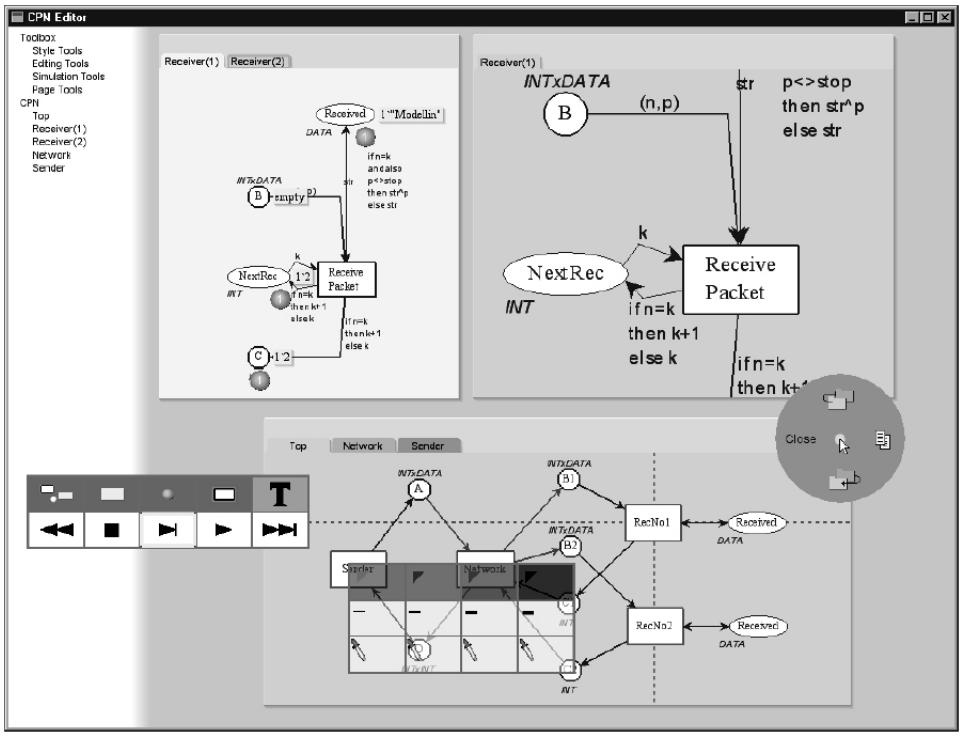


FIG. 1.13 – L'espace de travail de CPN2000 – extrait de (Beaudouin-Lafon & Lassen, 2000)

L'application CPN2000 met en œuvre de nombreuses techniques d'interaction et de visualisation dans un espace de travail dédié à l'édition et la simulation de réseaux de Pétri colorés : guides magnétiques, "marking menus", palettes flottantes, et "toolglasses" (Beaudouin-Lafon & Lassen, 2000). Le style de l'interaction est relativement varié. Par exemple, il est possible d'utiliser les palettes d'outils de manière traditionnelle, soit par une interaction mono-manuelle basée sur l'application d'une commande par un outil, soit de manière bimanuelle en les transformant en "toolglass". De la même manière, une interaction directe bimanuelle permet d'effectuer des actions telles que le dimensionnement et le zoom. L'application est finalement un espace de travail complet utilisant un système de fenêtrage basé sur les classeurs contenant des pages accessibles par leurs onglets.

Ces deux exemples illustrent comment un système peut être construit de manière *cohérente* et *consistante* à partir de techniques d'interaction et de visualisation issues de la recherche. Ces exemples sont cependant abordés sous l'angle applicatif, *i.e.* pour des tâches liées à un type de document prédéfini. L'objet de cette thèse est d'étudier comment de telles techniques peuvent s'intégrer de manière cohérente et consistante au niveau d'un espace de travail *dans son ensemble*. Plus précisément, nous nous intéresserons principalement à définir un modèle qui rende possible l'élaboration de tels environnements interactifs, la conception (ou "design") étant un aspect à considérer *a posteriori*.

1.3 Systèmes et espaces de travail orientés documents

Nous présentons dans cette section les motivations quant à une approche centrée sur les documents. Nous effectuons un tour d'horizon des différents systèmes respectant une telle approche, depuis les plus simples (HotDoc et OOE) jusqu'au plus abouti (OpenDoc).

1.3.1 Tendance

Le fait que le document soit l'objet d'intérêt principal d'un espace de travail interactif a déjà été identifié à l'époque du Star. Dans sa rétrospective du Star, Johnson et al. (1989) souligne que "le document est au centre du monde et qu'il l'unifie"[4]. Le Star est ainsi le premier système majeur qui adapte l'approche centrée sur les documents ; l'idée de l'application centrale dite "éditeur de documents" illustre ce point (section 1.1.1). Cependant, dans leur immense majorité, les environnements interactifs actuels sont fondés sur la notion d'application. Une application est dédiée à la manipulation de données typées, classiquement du texte, des images, des dessins vectoriels, etc. Les systèmes de fenêtrage ont été introduits afin de faciliter le passage d'une application à l'autre, et des techniques de type copier-coller ont été imaginées afin de permettre de transférer le contenu des documents entre plusieurs applications. Cependant, de nombreuses tâches nécessitent l'utilisation conjointe de plusieurs applications. Par exemple, un utilisateur peut être contraint de jongler avec quatre ou cinq applications de manière à créer une page web ou un document technique. Les éditeurs de logiciels utilisent trois stratégies pour réduire la complexité qui en résulte ainsi que la charge cognitive supplémentaire de l'utilisateur.

La première consiste à créer des "mini-applications" à l'intérieur même d'applications plus larges. Par exemple, la suite logicielle Microsoft Office est constituée de trois applications principales, Word, PowerPoint et Excel. Elles incluent respectivement des fonctions de dessin vectoriel mais offrent des fonctionnalités globalement moindres qu'une application dédiée au dessin vectoriel. Le recours à une application dédiée devient alors nécessaire dans le cas de graphiques complexes. De plus, les interfaces utilisateur de ces trois applications sont quelques peu différentes et les formats de données restent incompatibles. Cette approche conduit ainsi à dupliquer les fonctionnalités et les formats de données sans réellement résoudre le problème.

La seconde approche consiste à offrir une architecture ouverte pour les applications permettant à des tierces parties de développer et de commercialiser des extensions appelées "plug-ins". Par exemple, un grands nombre de plug-ins sont disponibles pour l'application de retouche d'image Adobe Photoshop (Gray, 1997), depuis l'extension permettant la création de formes vectorielles jusqu'à celle autorisant la manipulation de modèles 3D simples. Le marché des extensions pour l'application de mise en page QuarkXPress ainsi que pour l'application d'édition de contenus multimédia Macromedia Director est également très actif, et certaines extensions sont plus coûteuses que l'application initiale. Les

[4] The document is the heart of the world, and unifies it.

utilisateurs installent les extensions de manière à spécialiser les applications en fonction de leurs besoins propres, et peuvent utiliser des extensions identiques pour des applications différentes. Cependant, les interfaces des extensions sont souvent mal intégrées dans les applications mères et restent accessibles exclusivement au travers de boîtes de dialogues souvent complexes.

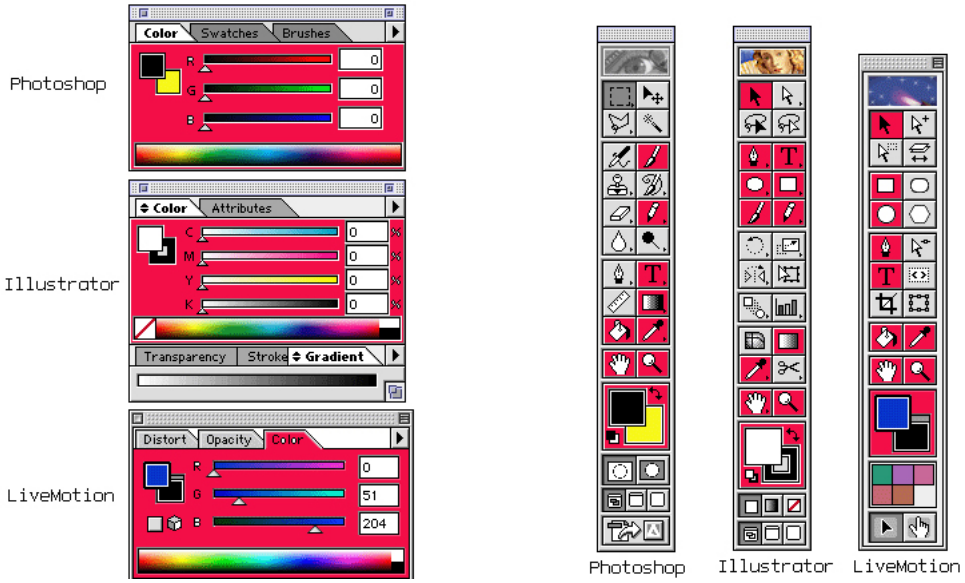


FIG. 1.14 – Similitudes (couleur foncée) entre les interfaces de la suite Adobe

Une troisième approche consiste, pour les éditeurs de logiciels, à créer des suites d'applications dont les interfaces sont compatibles. Cette approche rend l'interaction plus homogène et plus fluide, et le passage d'une application à l'autre est facilité. Le meilleur exemple est la suite des produits Adobe : les applications Photoshop pour la retouche d'image, Illustrator pour le dessin vectoriel, GoLive pour l'édition de sites Web et InDesign pour la mise en page, ont toutes la même présentation de l'interface graphique et des palettes d'outils similaires (figure 1.14). Elles supportent toutes les document en couches et les couches créées par l'une des applications peuvent être importées dans l'une des autres applications. Cependant, cette dernière approche impose à l'utilisateur de travailler avec plusieurs documents à la fois pour réaliser une tâche unique. Par ailleurs, les applications qui n'appartiennent pas à la suite logicielle ne peuvent bénéficier des fonctions qui y sont intégrées.

Le but de ces trois approches est de positionner le document, et non l'application, au centre de l'interaction. Cependant, elles ne parviennent pas réellement à leurs fins car les utilisateurs ont toujours à jongler entre plusieurs applications et plusieurs documents afin de réaliser une tâche. Elles essaient de rendre les applications moins visibles en réduisant leurs écarts, mais la logique générale reste centrée sur l'application. Ce constat constitue une autre motivation forte quant à cette thèse dans laquelle nous proposons un modèle pour la conception d'espaces de travail intégralement centrés sur les documents : l'application disparaîtra réellement.

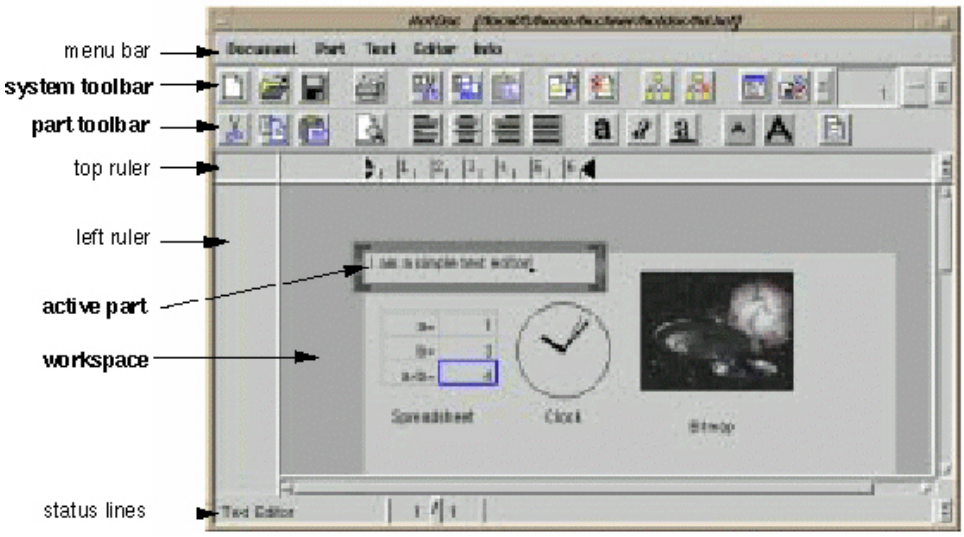


FIG. 1.15 – L'environnement HotDoc – extrait de (Buchner, 2000)

1.3.2 Systèmes orientés document

1.3.2.1 HotDoc

HotDoc est une extension Smalltalk du motif de conception MVC[5] qui permet la mise en œuvre de documents composites (Buchner, 2000). HotDoc propose un environnement (figure 1.15) dans lequel les documents sont structurés en un ensemble de parties (ou "parts") imbriquées sans restriction de profondeur. Chaque partie précise le type de l'objet qu'elle contient : cela peut être du texte simple, du texte stylé, une image, un graphique, ainsi que d'autres parties. Un document est défini par sa partie racine unique qui contient récursivement l'ensemble du document. L'insertion d'une partie se fait alors en précisant le type de la partie ainsi que sa disposition par rapport à la partie parente et aux parties sœurs. Lorsque l'utilisateur manipule l'une des parties d'un document, l'interface s'adapte à cette partie en proposant un menu et une barre d'outils adaptés à son type. Le motif de conception MVC de SmallTalk est étendue de la manière suivante :

- La classe `PartApp` définit le comportement du modèle d'une partie de type donné. Elle définit ainsi une petite application qui gère l'affichage et l'édition de la partie du document.
- La classe `PartView` définit la représentation graphique d'une partie de type donné.
- La classe `PartController` définit comment l'utilisateur interagit avec la partie du document.

L'avantage de HotDoc réside dans sa simplicité, la notion de partie étant gérée par le populaire motif MVC. Le fait que ce modèle soit générique en fait sa force mais aussi sa faiblesse : il est facilement utilisable en tant que motif de programmation général mais ne propose pas de modèle précis pour les documents, pour les vues et surtout pour l'interaction.

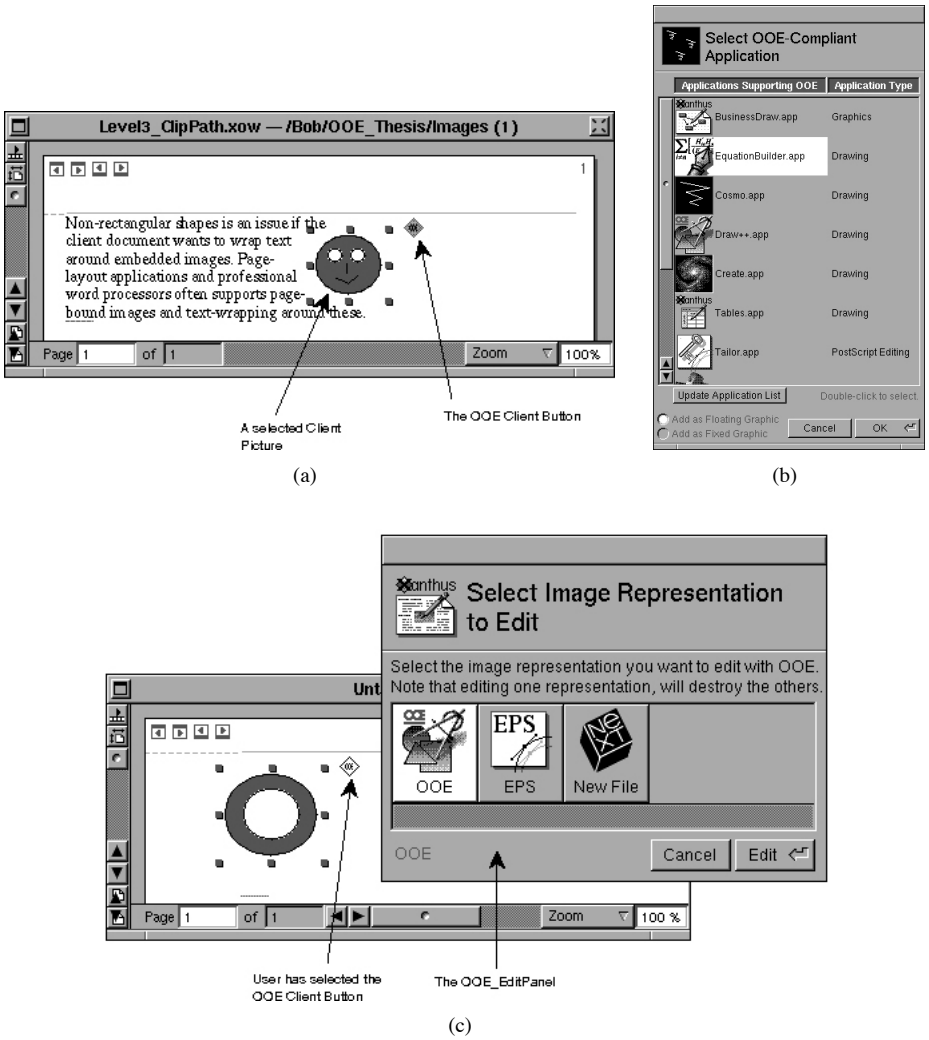


FIG. 1.16 – Édition d'un document composite avec OOE – extraits de (Backlund, 1997)

1.3.2.2 OOE

Le système OOE[6] est une extension du système NextStep qui permet une édition simplifiée des documents composites (Backlund, 1997). L'idée d'OOE est de rendre transparent le transfert des données nécessaires à la composition de documents. Le document, dit document client, contient typiquement du texte à la racine et est gérée par l'application cliente OpenWrite (traitement de texte du NeXT). Il contient des objets intégrés dans le document et éditables (figure 1.16-a). La visualisation de ces objets par l'application cliente est réalisée directement *via* le format "PostScript" qui est le format standard le l'affichage du NeXT[7] : l'application, dite application serveur, qui a servi à éditer l'objet intégré fournit cette représentation PostScript à l'application cliente, comme si elle "imprimait" l'objet dans l'application cliente. Lorsqu'un nouvel objet est intégré au document client, l'utilisa-

[5]"Design pattern" Model-View-Controller

[6]"Open Object Embedding"

[7]Le langage PostScript est normalement réservé à la description vectorielle des documents pour les imprimantes.

teur choisit le type d'objet à insérer le système *via* le sélecteur des applications serveurs (figure 1.16-b). Lorsque l'utilisateur souhaite éditer l'objet intégré d'un document client, il active le bouton "ooe" situé en haut et à droite de l'objet (figure 1.16-a) et choisit d'éditer l'objet au travers de l'application serveur ou au travers de sa représentation PostScript[8], ou d'en créer un nouveau (figure 1.16-c).

L'utilisation d'OOE est ainsi relativement simple et ne change pas beaucoup les habitudes des utilisateurs. L'application demeure toujours l'élément central du système et l'utilisateur bascule simplement d'une application à l'autre afin d'éditer un document composite. Cette approche ne permet pas l'édition *sur place* des documents, c'est-à-dire l'édition des objets intégrés à l'intérieur même de l'application cliente. Le document client est nécessairement de type texte ce qui reste limitatif. De plus, un objet intégré ne peut pas contenir un autre objet intégré ; les applications serveurs et clientes sont donc conçues de manière différente (asymétrie). Globalement, OOE reste encore très orienté application.

1.3.2.3 OLE

Le système OLE[9] est un système développé par Microsoft et permettant l'édition de documents composites par le biais d'une communication inter-applications (Brockschmidt, 1995). Les éléments d'un document OLE jouent deux rôles :

1. Le conteneur : Il définit "l'endroit" qui est le client d'un serveur.
2. Le serveur : Il définit les "choses" qui résident dans les "endroits".

Par exemple, un élément graphique (le client) dans un document texte (le serveur) définit l'endroit du document occupé par l'élément graphique ainsi que son contenu propre. La composition d'un document se fait par des liens entre fichiers ou par une incorporation réelle. L'imbrication est en théorie non limitée mais l'édition sur place n'est possible que pour un seul niveau.

L'approche proposée par OLE est donc centrée sur l'application. Elle reste, comparativement à OOE ou HotDoc, extrêmement complexe à mettre en œuvre. OLE étant un kit de construction d'interfaces utilisateurs uniquement orienté développeur et sans une réelle "philosophie", il est difficile pour les éditeurs de connaître les règles d'ergonomie à appliquer aux applications compatibles OLE. Dans la réalité, ces "lois" sont fixées par les applications Microsoft telles que Word, Access ou Excel, si bien qu'aujourd'hui, il est difficile d'imaginer d'autres applications conteneurs compatibles OLE. Pour ces applications, les interfaces et les interactions restent homogènes et l'utilisation d'OLE reste alors assez continue pour l'utilisateur. Enfin, notons que l'édition sur place s'effectue par une activation à étapes (dite "outside-in") : l'élément client devient éditable une fois qu'il a été "ouvert", ce qui nécessite typiquement un double-clic sur cet élément.

1.3.2.4 OpenDoc

Description Le système OpenDoc développé par Apple est l'une des rares tentatives d'élaboration d'un système réellement centré document, par opposition aux systèmes centrés application (Apple, 1994; Orfali et al., 1996). Dans ce système, l'utilisateur n'a plus à manipuler des applications afin d'éditer des documents : il manipule directement le document.

Le modèle de document est basé sur la métaphore de la feuille blanche : lorsqu'un utilisateur crée un nouveau document, il utilise typiquement un document vierge (la feuille

[8] Utile pour des retouches succinctes lorsque l'application serveur n'est pas présente
[9] Object Linking and Embedding

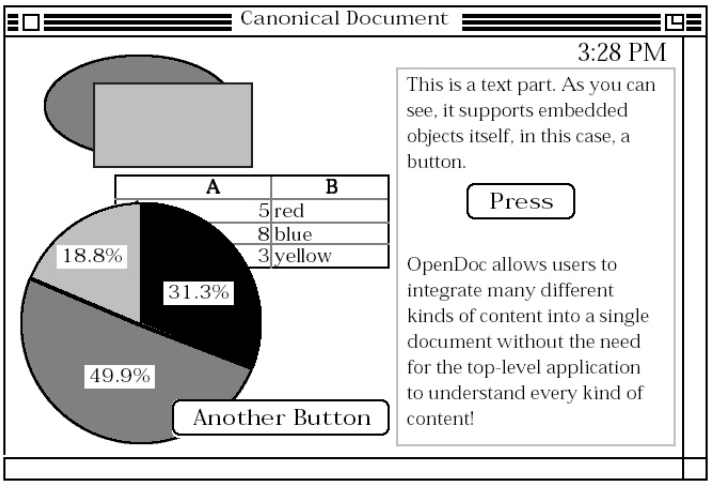


FIG. 1.17 – Exemple de document composite OpenDoc – extrait de (Apple, 1994)

blanche) et y ajoute des éléments de types variés. Ces différents éléments constitutifs sont appelés des *parties* et peuvent être imbriquées les unes dans les autres, sans limite de profondeur. Le document possède systématiquement une partie racine unique qui définit le point d'entrée des différentes parties du document. La figure 1.17 donne un exemple de document composite OpenDoc. La partie racine est un graphique vectoriel qui contient un rectangle et une ellipse. Elle contient une partie "horloge", une partie "tableau" que chevauche la partie "camembert", une partie de type bouton ("Another Button") et finalement une partie textuelle. La partie textuelle contient le texte ainsi qu'une partie de type bouton ("Press"). Chaque partie possède son propre modèle de contenu et de comportement. Par exemple, la partie texte de la figure 1.17 contient du texte sans attributs de style et que l'utilisateur peut sélectionner, copier, coller, supprimer ou modifier. L'ellipse et le rectangle sont des éléments graphiques contenu dans la partie racine du document, lesquels peuvent être déplacés, peints ou dimensionnés.

Les parties d'un document OpenDoc sont gérées par un *éditeur* de partie ou par un *visualisateur* de partie. Les éditeurs et visualisateurs sont les substituts OpenDoc des classiques applications. La distinction entre visualisateurs et éditeurs est intéressante : typiquement, les éditeurs sont payants alors que les visualisateurs sont gratuits. Il est ainsi recommandé aux développeurs de fournir un éditeur *et* un visualisateur pour chaque type de partie mis en œuvre. Les éditeurs et visualisateurs sont responsables de trois aspects essentiels :

1. L'affichage de la partie. Cela concerne aussi bien l'affichage sur écran que l'impression sur papier.
2. La persistance de la partie.
   (a) La partie doit pouvoir être lue à partir d'un enregistrement.
   (b) La partie doit pouvoir être enregistrée.
3. L'édition de la partie. L'éditeur doit gérer les événements utilisateurs et mettre à jour la partie.

Les aspects 1 et 2-a doivent être gérés par les éditeurs et par les visualisateurs. Les aspects 2-b et 3 ne concernent que les éditeurs. L'association d'une partie avec un éditeur ou un visualisateur se fait par le type de la partie. Lorsqu'un document OpenDoc est ouvert, l'affichage initial du document est réalisé en invoquant tous les éditeurs associés à chacune

des parties constitutives du document, association basée sur le type de la partie. Si l'éditeur n'est pas disponible sur la machine, le système cherche un visualisateur potentiel. Si ce visualisateur n'existe pas sur la machine, il devrait être possible de la télé-charger au travers du réseau. Si le visualisateur n'est pas accessible, un cadre grisé est affiché à la place de la partie.

Les paragraphes précédents synthétisent les principes généraux d'OpenDoc. Il existe de nombreuses autres particularités que nous mentionnons ici rapidement. Une première particularité concerne la possibilité d'utiliser des scripts qui agissent sur les parties des documents composites de manière à pouvoir automatiser les actions répétitives. Ceci étend largement le spectre des scripts que nous écrivons dans des "shell" : il est possible de dialoguer directement avec les documents et non exclusivement avec le système d'exploitation. La seconde particularité concerne la collaboration, *i.e.* la possibilité de partager un document. Pour ceci, OpenDoc utilise la notion de version de document ("draft") : lorsque des utilisateurs travaillent sur un document partagé, chacun édite son propre "draft" du document. Les différents drafts peuvent ensuite être utilisés pour mettre à jour le document final. Enfin, la troisième particularité concerne la portabilité : OpenDoc a été conçu – en théorie – pour les plate-formes Macintosh, Microsoft Windows, Motif, OPEN LOOK et OS/2.

Malheureusement, le système OpenDoc n'a pas été commercialisé, ceci en dépit de ses nombreuses qualités et de son aspect très innovant. Il est probable que, si tel avait été le cas, le paysage des environnements interactifs actuels serait très différent.

Comparaison Comparativement aux systèmes OOE, HotDoc et OLE, l'approche OpenDoc a de nombreux avantages :

1. Les applications disparaissent réellement et sont remplacées par des éditeurs ou visualisateurs. La granularité applicative est ainsi réduite.
2. Le système est parfaitement symétrique, contrairement à OOE et OLE. Il n'y a pas d'application "racine" qui fixe le type principal du document : il s'agit ici d'une partie racine.
3. OpenDoc définit une ergonomie de l'interface assez précise à laquelle tout éditeur et visualisateur doit adhérer. L'activation des parties se fait en mode "inside-out" : l'utilisateur interagit directement dans une partie sans avoir à "ouvrir" cette partie (contrairement à OLE).
4. La scriptabilité des parties étend la scriptabilité des applications présente dans MacOS, cette dernière étant généralement pauvre, voire absente, des autres systèmes d'exploitation.
5. La collaboration est abordée par la notion de "draft" qui est implantée dans le modèle des documents OpenDoc.
6. OpenDoc est (ou plutôt visait à être) multi plate-formes.

En vertu de toutes ces qualités relatives, nous soulignons les imperfections suivantes :

1. La granularité des éditeurs et visualisateurs (respectivement des parties) reste assez forte, inférieure cependant à celle des applications (respectivement des fichiers).
2. D'une partie à l'autre (*i.e.* d'un éditeur à l'autre), l'interface utilisateur change. Ce changement est toutefois assez léger, typiquement la barre de menu et certaines barre d'outils changent, mais il existe et reste perceptible.
3. Chaque partie définit son propre format de fichier en respectant le protocole imposé par OpenDoc. Les formats, même s'ils demeurent potentiellement propriétaires, peuvent ainsi être mixés dans un même fichier. Cependant, cela ne va pas dans le sens de formats ouverts (tels que les formats dérivés de XML).

4. La collaboration est abordée sous l'angle très restrictif des "drafts" (édition asynchrone). L'édition en temps réel d'un document partagé (édition synchrone) n'est ainsi pas abordée.
5. Si l'ergonomie liée à OpenDoc semble satisfaisante, aucun modèle précis d'interaction n'est fourni (probablement dans un soucis d'ouverture).

Ces différentes critiques émises à l'égard d'OpenDoc complètent les motivations de cette thèse. Ainsi, nous proposerons un modèle qui tente de combler ces lacunes sans en apporter de nouvelles (chapitre 3 et suivants). C'est pour ces raisons que nous avons fait une analyse plus approfondie d'OpenDoc que des autres systèmes.

1.4 Systèmes et espaces de travail collaboratifs

Engelbart & English (1968) avaient identifiés très tôt l'intérêt des collecticiels ainsi que l'utilisation de la vidéo dans la collaboration à distance. L'émergence des collecticiels s'est produite depuis le début des années 80 et ne cesse aujourd'hui de s'accroître tant dans les entreprises que dans les foyers.

Nous introduisons cette section par un positionnement du problème des collecticiels à l'aide de la matrice de la collaboration. Nous présentons ensuite différents systèmes collaboratifs selon quatre axes : la collaboration en temps réel, la collaboration en temps différé, le partage de l'espace de travail et la médiatisation des espaces.

Le but de cette section est de déduire les principaux "traits" des systèmes collaboratifs que notre modèle devra prendre en compte. Il s'agit en particulier du partage des objets, de la distribution des objets partagés, de la gestion de la concurrence d'accès et de la cohérence de ces objets, ainsi que de la conscience réciproque des actions des utilisateurs.

1.4.1 La matrice des collecticiels

	Même endroit	**Endroits différents**
Même instant	Conversation en face-à-face	Téléphone
Instants différents	Note "Post-it"	Courrier

TAB. 1.1 – La matrice de la collaboration – traduit de (Dix et al., 1998)

La collaboration entre individus a des caractéristiques et des objectifs extrêmement variés. Deux axes essentiels permettent de caractériser cette collaboration : l'axe temporel et l'axe spatial (table 1.1). L'axe temporel précise si la collaboration s'effectue au même instant ou à des instants différents. L'axe spatial précise si la collaboration a lieu au même endroit ou en des endroits différents. Nous pouvons ainsi distinguer quatre manières typiques de collaborer, chacune correspondant à un emplacement dans la matrice de la collaboration :

1. Lorsque la collaboration a lieu au même endroit et au même instant, il s'agit de la très usuelle conversation en face-à-face avec tout ce que cela comporte en terme d'échange : la communication est verbale mais aussi très largement gestuelle.
2. Une conversation peut avoir lieu en des endroits différents par le biais du téléphone. Dans ce cas, la communication n'a plus le caractère gestuel de la conversion en face-à-face et le côté verbal est renforcé par des expressions typiques du téléphone telle que "allô" (ou le simple fait de raccrocher pour couper court à la conversation !).

3. La collaboration peut avoir lieu au même endroit mais à des instants différents. La liste des courses est un exemple pris dans notre vie quotidienne : chaque membre d'une famille peut écrire sur une liste accrochée au mur de la cuisine une course à faire, et ainsi collabore avec la personne chargée de faire de telles courses en fin de semaine.
4. Enfin, le courrier postal est un exemple de communication à des instants différents (entre l'envoi du courrier et la réception de la réponse) et à des endroits différents. Comme pour le téléphone, un vocabulaire est apparu pour cette forme de communication comme les formules de politesse écrites en fin de lettre.

	Un seul lieu de réunion (même endroit)	**Plusieurs lieux pour la réunion** (endroits différents)
Communication synchrone (même instant)	**Interactions en face-à-face** - Tables de conférence avec écrans intégrés - Écrans publics - Outils dédiés ("brainstorming" par exemple)	**Interactions à distance** - Conférence *via* éditeurs partagés - Conférence *via* bureaux partagés - Visioconférence, "chat" - Espaces médiatisés
Communication asynchrone (instants différents)	**Tâches à réaliser** - Salles d'équipe - Écrans de groupe - Logiciel de répartition de tâches - Gestion de projet	**Communication et coordination** - Email - Forum de discussion - "Workflow" - Contrôle de version

TAB. 1.2 – La matrice des collecticiels – traduit de (Johansen, 1988) dans (Baecker et al., 1995)

La matrice de la table 1.1 peut-être reprise pour caractériser et classer les différents types de collecticiels. L'utilisation des collecticiels permet d'étendre le champ de la collaboration de la vie courante. Certains d'entre-eux – le courrier électronique en est un bon exemple – sont devenus partie intégrante de notre vie quotidienne. La table 1.2, dite "matrice de collecticiels", précise quels types de collaboration peuvent être effectivement engagés en utilisant les collecticiels.

Les systèmes dédiés à la *communication asynchrone* (instants différents) permettent la communication ainsi que la résolution de problèmes entre des groupes d'individus. Ces derniers interviennent à divers intervalles de temps et sont en général géographiquement dispersés. Nous pouvons typiquement distinguer quatre types de systèmes asynchrones :

Le courrier électronique – Il permet d'organiser son courrier typiquement en fonction de l'envoyeur et de la date de réception, et éventuellement en fils de discussion. Un message peut être adressé simultanément à plusieurs destinataires permettant ainsi d'engager une collaboration. Le courrier électronique pose un certain nombre de problèmes, tels que la difficulté de suivre le fil d'une collaboration, l'abondance de "spam" (publicité non sollicitée), la gestion difficile d'une base non négligeable de messages, ou encore l'absence d'accusé de réception.

Les forums de discussion – Ils permettent d'entamer des discussions sur un thème propre au forum. Les messages sont organisés par fil de discussion (sujet + date) et sont ainsi hiérarchiquement structurés. La classification, le filtrage et la gestion des messages sont plus naturels que pour le courrier électronique puisque l'accent est mis sur le sujet et donc le fil de discussion, et non sur l'expéditeur.

L' "instant messaging" – Les systèmes d' "instant messaging" permettent à un groupe d'utilisateurs de se rencontrer de manière inopinée, c'est-à-dire non planifiée à l'avance.

Un utilisateur qui souhaite pouvoir être contacté par un membre du groupe auquel il s'est préalablement enregistré a simplement besoin de se "connecter" au groupe. Le contact peut avoir lieu *via* un système de courrier électronique ou du "chat" connexe.

Les systèmes "workflow" (Ellis, 1999) – Ils permettent d'utiliser un mécanisme de messagerie afin de définir, répartir et gérer le flux des tâches lié à une organisation spécifique. De tels systèmes sont centrés sur la notion de "corbeille" : la corbeille d'un utilisateur reçoit l'ensemble des tâches qui incombent à cet utilisateur sous la forme de documents, ces différentes tâches s'inscrivant dans un ensemble plus vaste de tâches hiérarchisées et nécessaires au bon fonctionnement de l'organisme en question.

Le contrôle de version – Les systèmes de contrôle de version permettent de gérer le suivi des différentes versions d'un document ou d'un projet. Dans ces systèmes, les utilisateurs éditent une version du document et valide les modifications une fois l'édition terminée. Les systèmes permettent de plus de retrouver des versions antérieures de documents. Ce type de système est adapté aux situations nécessitant une collaboration très formalisée tel que le développement logiciel (Krause, 2001).

Les systèmes dédiés à la *communication synchrone* (au même instant), parfois qualifiés de systèmes "temps réel", aident les groupes d'utilisateurs répartis à travailler ensemble et en même temps. Nous pouvons typiquement distinguer quatre types de systèmes synchrones :

Les éditeurs partagés (Prakash, 1999) – Ils permettent d'engager des phases de collaboration par le biais de l'édition partagée de documents. Ils assurent la gestion de la concurrence d'accès aux données et doivent fournir un retour d'information quant aux actions de chaque utilisateur ("awareness"). Un éditeur partagé ne fonctionne pas nécessairement sur un système d'exploitation donné mais peut inter-opérer avec différents systèmes d'exploitation.

Les bureaux partagés – L'utilisation de bureaux partagés permet de collaborer d'une manière plus couplée que lors de l'utilisation d'un éditeur partagé. Les éléments des espace de travail peuvent être communs à plusieurs utilisateurs en simulant ainsi à distance le partage de l'écran.

La visioconférence – Elle permet de tenir des réunions à distance engageant ainsi une collaboration à caractère informel. L'idée directrice de tels systèmes est de réduire l'interface entre plusieurs salles de visioconférence distantes, afin de créer l'illusion qu'il s'agit – tant que faire se peut – d'une unique salle de réunion. L'utilisation d'artefacts partagés, tel qu'un tableau blanc partagé, permet de réduire cette distance.

Les espaces médiatisés (Mackay, 1999) – Les systèmes précédents souffrent de ne pouvoir fournir la conscience de "qui est là ?" et "comment puis-je atteindre telle personne ?". Le but des espaces médiatisés (ou "mediaspaces") est de combler cette lacune en proposant d'adjoindre aux systèmes informatiques un système audio-visuel en réseau. Ce système fournit une information audio-visuelle quant à la présence de tel ou tel individu et à son activité.

Les différents types de systèmes collaboratifs présentés ci-dessus correspondent à la colonne "endroits différents" de la matrice de la collaboration (table 1.1). Cependant, ignorer l'utilisation de l'ordinateur en tant qu'outil de collaboration dans un "même endroit" serait un erreur. En particulier, les systèmes basés sur la *co-présence* des individus ("single display groupware") présentent un intérêt certain comme l'illustre l'application KipPad conçue pour l'édition graphique collaborative sur un même écran (Stewart et al., 1999). Assez peu d'engouement existe en ce qui concerne la communication asynchrone au même endroit. Ceci est probablement dû au fait que l'ordinateur reste en général une machine individuelle, ou une machine connectée à un serveur et qui ne joue alors aucun rôle collaboratif. Ce constat pourrait s'atténuer le jour où les systèmes intégreront réellement la collaboration de type "single display groupware".

1.4.2 Applications collaboratives synchrones

Les collecticiels synchrones, ou temps réel (“real-time groupware”), sont des applications qui permettent typiquement l’édition collaborative de graphiques ou de texte. De telles applications sont apparues principalement au début des années 90 dans les laboratoires de recherche.

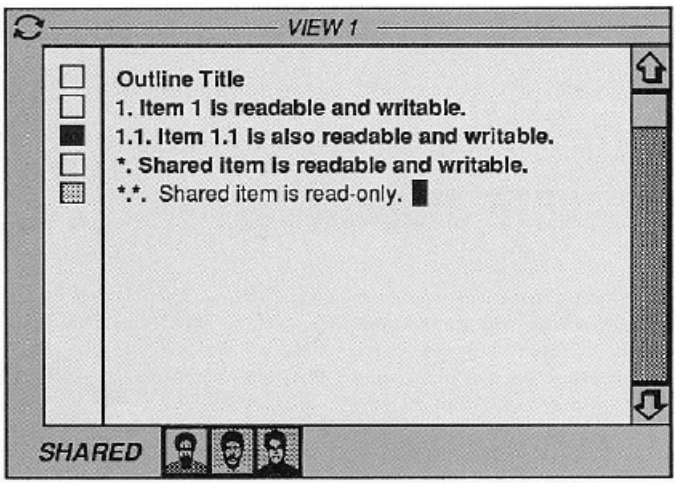


FIG. 1.18 – Exemple de session Grove – extrait de (Ellis et al., 1991)

Le premier type de collecticiels synchrones concerne les éditeurs de texte collaboratifs. L’application GROVE[10] permet l’édition de documents partagés et en particulier l’édition de leur structure (Ellis et al., 1991). Ces documents sont répliqués sur l’ensemble les postes concernés pour des raisons de performance imposée par l’interaction. L’édition du plan d’un document partagé s’effectue au travers d’un éditeur local par les utilisateurs enregistrés (figure 1.18). Pour chaque partie du document et pour chaque utilisateur est défini un droit d’accès (lecture / écriture ou lecture seule) ; les titres de parties sont grisés quand elles ne sont pas accessibles en écriture par l’utilisateur. L’accès aux différentes parties s’effectue *via* les boutons situés sur le bord gauche de la fenêtre de l’éditeur : le bouton est grisé si la partie n’est accessible qu’en lecture, blanc si la partie peut-être éditée et noire si elle est en cours d’édition (*i.e.* ouverte). L’éditeur est, par défaut, basé sur une gestion optimiste de la concurrence d’accès : une partie n’est pas verrouillée lorsqu’elle est ouverte en écriture, le système gérant lui-même la cohérence des différentes répliques (une incohérence peut survenir quand plusieurs actions sont simultanément appliquées sur une partie).

L’application MACE gère l’édition coopérative des fichiers textuels avec une *granularité* fine, du niveau du caractère (Newman & Pelimuhandiram, 1991). Elle utilise le verrouillage des parties de texte, les utilisateurs distants ayant une version à jour du fichier texte lorsque la partie est déverrouillée. L’attrait majeur de MACE tient dans sa capacité à faire varier la granularité du verrouillage : il est possible de visualiser les modifications du texte en temps réel par un verrouillage au niveau des caractères (grain fin), ou bien de travailler d’une manière plus asynchrone en verrouillant une section entière du fichier texte (grain fort).

Le second type de collecticiels synchrones correspond aux éditeur de graphiques (vectoriels ou à main levée).

L’application GroupSketch a été conçue pour permettre à nombre restreint d’individus de collaborer en partageant une *surface de travail* (Greenberg et al., 1992). Les membres du

[10] “GRoup Outline Viewing Editor”

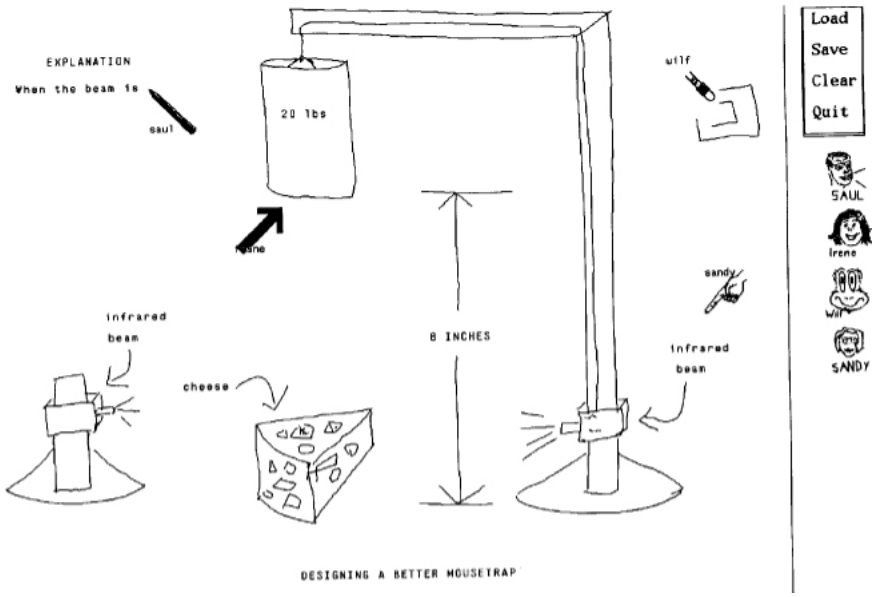


FIG. 1.19 – Exemple de session GroupSketch – extrait de (Greenberg et al., 1992)

groupe échangent des idées en dessinant à main levée sur cette surface (figure 1.19). La conception de GroupSketch met en avant les quatre principes suivants :

1. Les participants partagent une vue identique de la surface de travail. Elle fournit un accès simultané aux participants ainsi qu'une impression de proximité.
2. La surface accepte la communication gestuelle. Les gestes sont visibles par les mouvements des différents curseurs, et maintiennent une relation avec les objets de la surface de travail ainsi qu'avec la communication vocale.
3. L'application incite à créer des artefacts pour exprimer des idées.
4. Elle permet de mixer d'une manière fluide les actions (dessiner) et les fonctions (sauvegarde une information, expression d'une idée).

Cet exemple d'application illustre clairement la notion de WYSIWIS[11] : les membres du groupe perçoivent tous la *même* vue de la surface. Quatre "outils" sont disponibles pour interagir sur la surface de travail : le crayon pour le dessin à main levée et l'édition textuelle, la gomme pour la suppression de graphiques ou du texte, un "télé-pointeur" permettant de montrer du doigt un élément particulier de la scène (la flèche sur la figure 1.19), et enfin le curseur qui indique la position de la "main" de l'utilisateur de manière à avoir conscience des déplacements de chacun sur la surface (la main sur la figure 1.19). Chaque outil possède un label qui précise le surnom de l'utilisateur concerné. GroupSketch est ainsi un bon exemple qui préfigure les surfaces partagées actuelles, comme le tableau blanc que nous retrouvons conjointement à l'application NetMeeting fournie avec Windows XP.

L'application DOLPHIN est un environnement dédié à l'édition collaborative de graphiques (Streitz et al., 1994). Elle permet notamment la réalisation de "brainstorming" pour les membres d'un groupe géographiquement réparti. Ces membres peuvent réaliser des croquis, saisir du texte ou créer des liens entre différentes pages de manière à communiquer leurs idées (figure 1.20). Le système utilise des outils de conférence audiophonique de manière à pouvoir engager des discussions à propos du contenu des documents partagés.

[11] "What You See Is What I See"

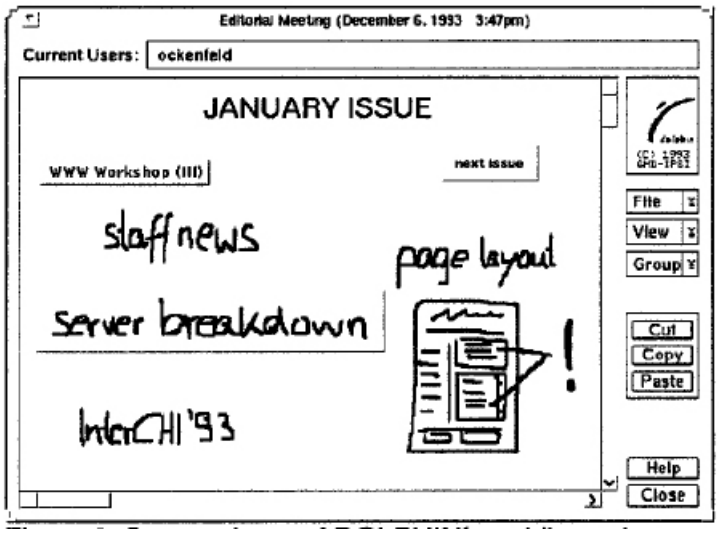


FIG. 1.20 – Exemple de session Dolphin – extrait de (Streitz et al., 1994)

Les “brainstorming” peuvent classiquement utiliser les capacités de réalisation de dessins, mais également les liens hypertexte entre les documents de manière à organiser le fil de la discussion et à favoriser la prise de décision. Les actions sur les objets graphiques (déplacement, dimensionnement) fournissent un retour visuel immédiat. Le système garantit que tous les utilisateurs perçoivent le même état pour les objets partagés même si ces derniers sont manipulés simultanément. Les anomalies qui peuvent apparaître sont réparées par un mécanisme d’annulation des actions concurrentes de manière à les réordonner correctement le cas échéant. Enfin, l’application précise quels utilisateurs accèdent à tels documents partagés, fournissant ainsi un mécanisme qui renforce la perception des actions des différents utilisateurs[12]. Cet aspect est important notamment pour inciter les utilisateurs à éviter *par eux-même* les accès concurrents (gestion optimiste de la concurrence).

1.4.3 Applications collaboratives asynchrones

1.4.3.1 Systèmes basés sur l’annotation

Les applications collaboratives en temps différé, *i.e.* asynchrones (“asynchronous groupware”), sont apparues quelques années avant les premières applications synchrones. Ceci est probablement dû au fait que l’aspect temps réel pose des problèmes techniques encore difficiles à résoudre aujourd’hui : l’interaction nécessite un temps de réponse très faible (non perceptible par l’utilisateur dans l’idéal) généralement difficile à satisfaire dans des infrastructures distribuées au travers de réseaux.

L’application Quilt est un outil de collaboration dédié à l’écriture de documents textuels et qui propose des mécanismes d’annotation, de messagerie, de conférence et de notification (Fish et al., 1988). Il permet le partage de messages textuels et vocaux ainsi que le partage de documents. Les documents peuvent être soit directement modifiés, soit annotés, et leurs historiques permettent de visualiser l’ensemble des modifications effectuées depuis la version initiale du document (figure 1.21). Quilt intègre également la notion de *rôle* : un utilisateur peut jouer à un moment donné un ou plusieurs rôles, tels que l’auteur

[12]Cet aspect essentiel des collecticiels correspond à l’appellation anglophone “awareness”, littéralement conscience réciproque (des actions et intentions des autres utilisateurs).

FIG. 1.21 – Exemple d'utilisation de Quilt – extrait de (Fish et al., 1988)

principal du document ou le critique d'un autre document, ce qui lui confère des droits d'accès précis. Ainsi cette application propose déjà les éléments essentiels des collecticiels asynchrones. En particulier, l'un des éléments essentiel qui apparaît dans Quilt concerne la possibilité d'annoter un document, et ceci de manière récursive (une annotation peut être à son tour annotée). Cette forme de collaboration s'avère très utile et est utilisée dans les systèmes plus modernes, probablement parce qu'elle pré-existe largement dans la vie de tous les jours lorsque nous discutons autour des documents. L'intérêt de l'annotation tient en ce que, tout en y apportant des modifications, le document d'origine reste intact permettant ainsi de conserver le contexte initial. Typiquement, les annotations sont utilisées afin de modifier le document de manière collaborative et par étapes successives.

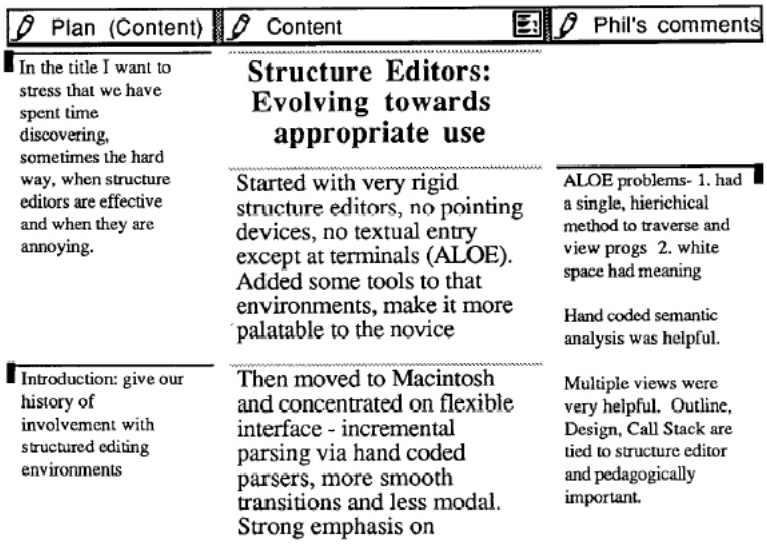


FIG. 1.22 – L'application Prep – extrait de (Neuwirth et al., 1990)

L'application Prep permet l'édition de documents structurés (Neuwirth et al., 1990). De tels documents peuvent être annotés par le biais de colonnes, chaque colonne étant dédiée à un type d'annotation donné. Par exemple, le document de la figure 1.22 définit une colonne centrale pour l'édition du contenu principal du document, alors que la colonne de gauche permet d'annoter le plan et ainsi de justifier la structure choisie pour le document, et la colonne de droite contient les annotations du co-auteur du document. Cette mise en parallèle des annotations est intéressante puisqu'elle suit le fil du document.

1.4.3.2 Systèmes de type "workflow"[13]

Les différentes applications précédentes illustrent l'intérêt de l'annotation dans le processus de collaboration. Cependant, cette technique peut s'avérer rapidement non adaptée à des documents volumineux ainsi qu'à la gestion d'un ensemble vaste de documents (donc relatifs à un projet par exemple). Dans pareils cas, l'utilisation d'un système de type "workflow" s'avère être la solution.

Comme le souligne Ellis (1999), les entreprises sont aujourd'hui amenées à s'appuyer sur une grande variété de systèmes informatiques afin de faciliter et d'améliorer le traitement de l'information. Il existe de nombreux outils informatiques conçus en tant que systèmes d'information personnels (traitement de texte, feuille de calcul, planning de projet, etc.) mais peu d'outils sont conçus pour des groupes de personnes. Le "workflow" est apparu comme l'outil le plus adapté à la gestion et à l'organisation du travail en groupe dans les entreprises. Les systèmes de gestion de type "workflow" sont conçus pour permettent à des groupes de personnes de mener à bien leur processus de travail ; ils définissent pour cela la *connaissance* relative à l'organisation de l'entreprise. Cette approche est ainsi très différente de celle des outils tels que le courrier électronique ou la visioconférence qui ne s'appuient sur aucune connaissance préalable du fonctionnement de l'entreprise et ne sont ainsi pas centrés sur son organisation.

Un système de type "workflow" est construit autour de deux composantes essentielles :

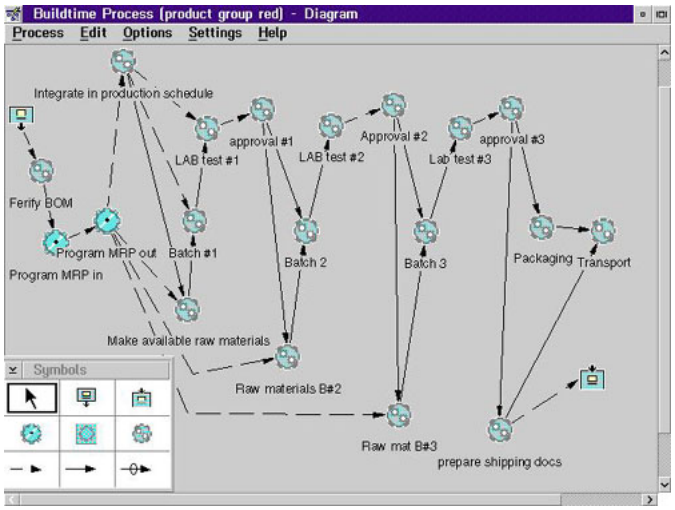
1. **Le composant de modélisation** – Il permet aux administrateurs ainsi qu'aux analystes de définir les procédures et les activités de l'entreprise, de les analyser, de les simuler et de les assigner à différentes personnes (répartition par rôle). Il permet de plus une analyse statique après coup et donc un ajustement du modèle. Ce composant est généralement appelé "module de spécification" (ou "build-time system").
2. **Le composant d'exécution du workflow** – Il consiste à présenter une interface homme / machine aux différents acteurs du processus ainsi qu'à faire tourner le "noyau" du système en permanence. Ce composant gère ainsi la coordination et la réalisation des différentes tâches définies pour le processus dans son ensemble. Il est généralement appelé "module d'exécution" (ou "run-time system").

Du point de vue des utilisateurs, il s'agit d'exécuter différentes tâches dans un ordre clairement défini, de manière séquentielle et/ou parallèle. Les différentes tâches parviennent typiquement jusqu'à l'utilisateur *via* une corbeille qui contient de manière ordonnée les différents documents relatifs à une tâche. Le modèle général des "workflow" est défini sous forme de recommandations par le "Workflow Management Coalition" (Hollingsworth, 1995).

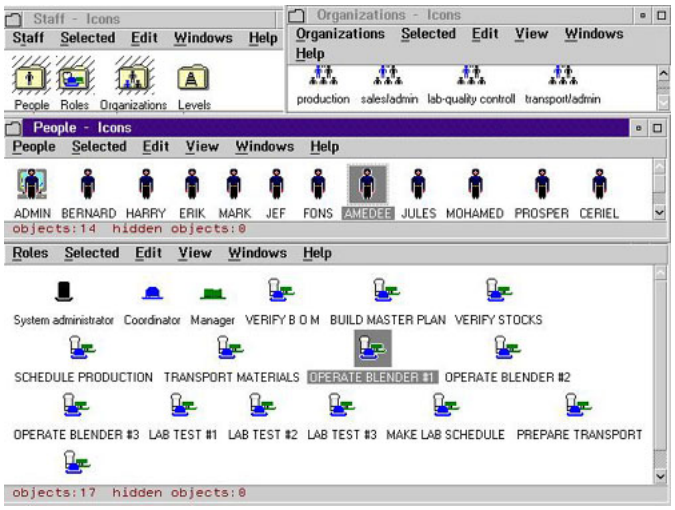
Le système workflow développé par IBM est le produit FlowMark commercialisé en 1994 (Ellis, 1999). Ce logiciel distingue clairement les deux composants "build-time system" et "run-time system". La figure 1.23-a montre un exemple de processus spécifié par un diagramme d'activité articulé autour d'actions et de transitions, et la figure 1.23-b illustre différents éléments que ce système distingue.

Le système Action Workflow fournit un modèle de workflow innovant centré sur les notions philosophiques de Heidegger et sur des aspect théoriques de la linguistique (Flores et al., 1988). L'idée directrice est que les interactions ou conversations sont composés d'actes de communication à interpréter par les personnes qui les perçoivent, mais qui restent cependant assujettis à de mauvaises interprétations (Ellis, 1999, page 44). Le linguiste Searle (1969) suggère l'idée qu'il existe un nombre fini de catégories d'actes de communication ; ce constat est utilisé par le système Action Workflow afin d'éviter leurs mauvaises interprétations.

[13] Le terme "workflow" est difficilement traduisible. Il sera donc utilisé comme anglicisme par la suite .



(a) Spécification d'un processus



(b) Objets d'un processus

FIG. 1.23 – Modélisation avec FlowMark – extrait de http :// www.infochain.be/ framesource/ _isostrm.html

Enfin, l'approche par "but" est un aspect intéressant des workflow qui a été notamment initié par le système expérimental Polymer (Croft & Lefkowitz, 1988). Cette approche tient compte du facteur humain suivant : les personnes ne suivent pas chaque étape d'une procédure formellement spécifiée mais prennent en considération le but des tâches qui leur incombent et font ce qui leur *semble* bon afin d'atteindre ce but.

L'intégration de l'aspect "workflow" dans le modèle que nous tentons de mettre en œuvre sort du cadre de cette thèse. Cependant, nous sommes convaincus des deux points suivants :

1. L'aspect "centré sur les documents" de notre modèle est un point qui devrait faciliter l'intégration de l'aspect "workflow" dans un espace de travail puisque le "workflow" consiste pour une grande part en la circulation de documents (nous pourrions parler de "document flow").
2. L'intégration de différentes approches de la collaboration dans notre modèle (synchrone *et* asynchrone par exemple) devrait permettre de prendre en compte les facteurs humains mentionnés plus haut. Par exemple, intégrer une approche de communication synchrone par un système de type "mediaspace" à une approche de type "workflow" pourrait aller dans ce sens.

1.4.4 Espaces collaboratifs

1.4.4.1 Espaces partagés

Les systèmes collaboratifs mentionnés précédemment, qu'ils soient de type synchrone ou asynchrone, sont typiquement des applications. Ils sont conçus pour réaliser un ensemble de tâches précis, généralement soit de manière synchrone, soit de manière asynchrone, mais rarement les deux à la fois. Ils sont ainsi conçus indépendamment les uns des autres ce qui rend difficile, voire impossible, leur interopérabilité. Ceci entre, à notre sens, en contradiction avec la nécessité de "complémentariser" les différentes approches de la collaboration. Typiquement, il est souhaitable qu'une application asynchrone puisse offrir certains services synchrones, et vice-versa. Par exemple, l'annotation devient un moyen de collaboration efficace quand elle est adjointe à un système de communication temps réel. Ce besoin d'intéropérabilité, et donc d'intégration, se fait ainsi largement ressentir dans les diverses applications précitées. Un autre exemple significatif concerne les workflow : s'ils permettent de structurer le travail en groupe dans les entreprises en proposant les composants de modélisation et d'exécution, ils ont systématiquement besoin d'autres applications dédiées, par exemple, à l'édition de documents. Notons enfin que cette non-interopérabilité influence également la qualité de l'interaction : les utilisateurs sont contraints de jongler entre des applications qui proposent rarement une interface de même ergonomie.

Ce constat constitue une motivation supplémentaire quant à cette thèse. Nous pensons qu'il est effectivement nécessaire d'amener les éléments liés à la collaboration au niveau de l'espace de travail lui-même autant que faire se peut. Par une telle approche, l'interopérabilité entre les différents composants interactifs et collaboratifs de l'espace de travail deviendrait nettement plus naturelle. Le fait de tenter de *faire disparaître l'application* des espaces de travail au profit des instruments d'interaction va de plus dans le sens naturel de la collaboration : dans notre vie courante, nous collaborons à l'intérieur d'un espace physique "partagé", ou en utilisant des instruments de communication tel que le téléphone ou le courrier.

L'utilisation de la métaphore de la "pièce" a déjà été mise en œuvre dans un espace de travail non collaboratif (Henderson & Card, 1986). Son utilisation a été initialement motivée par la nécessité de disposer d'un espace de travail plus grand que les écrans d'ordinateur, en proposant alors une alternative complémentaire quant à la gestion des fenêtres superposables. La première approche consiste à fournir un espace de travail "virtuel", c'est à dire

un espace plus grand que celui imposé par les dimensions de l'écran. La seconde approche proposée par Henderson & Card (1986) consiste à proposer plusieurs espaces de travail, ou "pièces", ces espaces pouvant être connectés les uns aux autres, à l'instar des pièces réelles qui sont "connectées" par des portes. Les auteurs argumentent le fait que nous organisons les objets relatifs à une tâche donnée de manière spatiale. Ainsi, dans leur approche, les documents et les outils relatifs à un projet donné sont disposés dans une "pièce" dédiée au projet. Les différentes pièces sont reliées entre-elles par un réseau d'interdépendance qui permet de se déplacer d'une pièce à l'autre. Cependant, ce maillage peut conduire à des relations entre les différentes pièces relativement complexes et n'est généralement pas utilisé par les autres types d'espace partagé. Notons toutefois que la métaphore de la pièce demeure pertinente et reste un élément central des espaces de travail collaboratifs (Roseman & Greenberg, 1996).

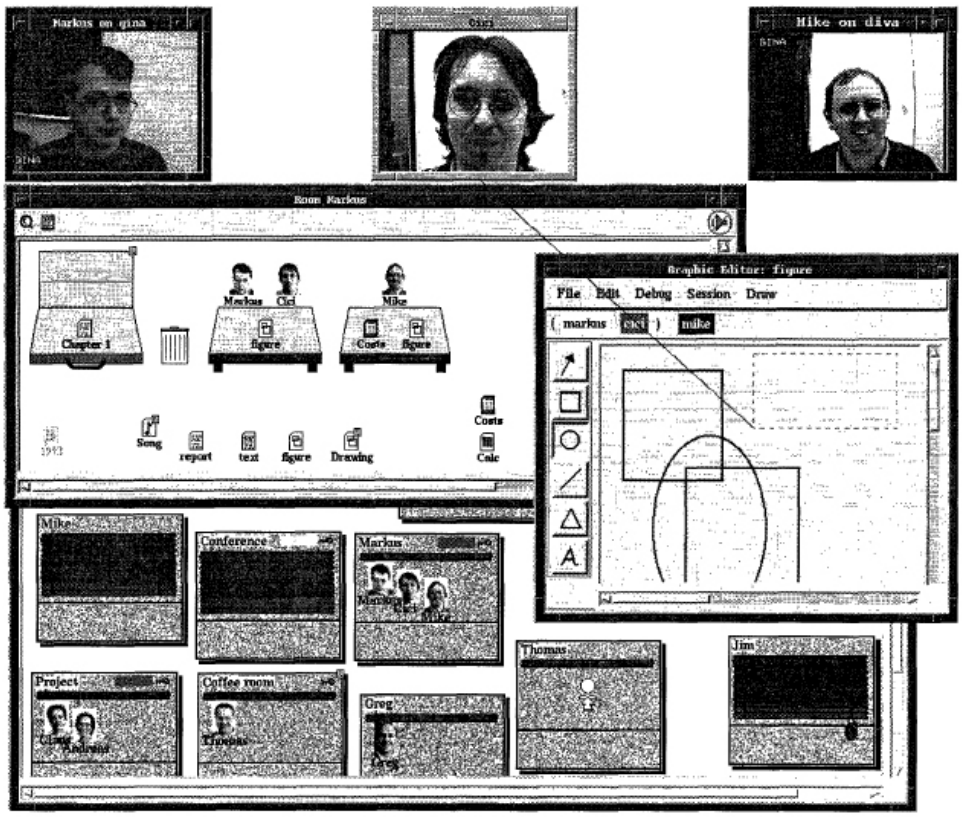


FIG. 1.24 – Un espace partagé avec DIVA – extrait de (Sohlenkamp & Chwelos, 1994)

Le système DIVA utilise d'une manière complémentaire les métaphores de la pièce *et* du bureau (au sens du "desktop"), et propose un certain nombre d'outils permettant de collaborer dans ces espaces (Sohlenkamp & Chwelos, 1994). La figure 1.24 montre un exemple de pièce partagée qui contient deux bureaux. Chaque utilisateur dispose d'un accès à la pièce commune (fenêtre "Room Markus" située au milieu de la figure 1.24) laquelle contient des documents et des bureaux partageables en temps réel. De plus, cette pièce affiche quels sont les utilisateurs qui ont joint la pièce et précise sur quels bureaux ils travaillent. Les outils de collaboration permettent le dialogue avec les différents participants (fenêtres situées en haut de la figure 1.24) ainsi que l'édition synchrone de documents (fenêtre "Graphic Editor : figure"). L'un des intérêts de DIVA est de clairement différencier l'ensemble des objets partageables des espaces de travail partageables : les premiers sont renseignés par le

contenu de la pièce, tandis que les seconds sont relatifs aux bureaux que la pièce contient. Cette idée nous semble intéressante et va, d'une certaine façon, dans le sens de la distinction entre place et espace (Harrison & Dourish, 1996) : les places sont des lieux (logiques) qui n'ont pas nécessairement un espace physique associé, alors que les espaces sont des lieux physiques. Nous reprendrons cette idée lorsque nous aborderons la notion de place publique (chapitre 6).

Le système TeamWave implémente très clairement la métaphore de la pièce commune (Greenberg & Roseman, 2002). La figure 1.25 synthétise une grande partie des composants de ce système :

– La fenêtre (a) fournit l'ensemble des pièces communes accessibles à la communauté et affiche l'identité des utilisateurs qui y sont présents. Un utilisateur entre dans une pièce en la sélectionnant dans la liste et prend en considération l'état de la porte :

  1. Ouverte : incite à entrer dans la pièce.
  2. Entre-ouverte : il est possible d'y entrer.
  3. Fermée : il est préférable de ne pas y entrer.
  4. Fermé à clé : il n'est pas possible d'y entrer.

– La fenêtre (b) indique quels utilisateurs ont joint au moins l'une des pièces de la communauté. Un double clic sur la photographie de l'une des personnes permet d'accéder à une fiche de renseignements sur la personne (fenêtre (c)), permettant ainsi d'engager une collaboration plus étroite.
– La fenêtre principale affiche la pièce partagée "TeamWave Demo" sur l'écran de Carl. Un nombre importants d'éléments collaboratifs y figurent :
  – Les photographies (h) montrent que Carl est en train de partager la pièce "TeamWave Demo" avec deux autres utilisateurs, Saul et Mark.
  – Les différents éléments de la pièce (espace "virtuel" plus grand que l'écran) sont visualisés par une "vue radar" (d).
  – Chaque utilisateur perçoit les curseurs (s) des autres utilisateurs ("télé-pointeurs") renforçant ainsi la conscience des actions qu'ils réalisent ("awareness").
  – Différents outils sont disponibles. Le "chat" (o) permet un dialogue écrit quasi-synchrone. Des crayons de différents couleurs (n) peuvent être utilisés sur le tableau blanc (m), ce dernier constituant le support de base des éléments de la pièce.
  – Différentes mini-applications (ou "applets") peuvent être utilisées sur l'espace partagé (e, i, j, k, l, p, q). Ces mini-applications rappellent les "part" d'OpenDoc.
  – Afin d'attirer l'attention des utilisateurs présents dans la pièce commune, un utilisateur peut activer la cloche (g) qui retentit alors sur l'ensemble des postes des dits utilisateurs.
  – Il est possible d'importer des documents dans son espace personnel (t).
  – Le statut de l'accès à la pièce peut être modifié en jouant sur l'état de la porte (f).
  – Enfin, il est possible d'accéder à d'autres pièces par des portes (r).

L'exemple de TeamWave est ainsi très significatif de ce qu'il est possible de construire à partir de la simple métaphore de la pièce commune. Il intègre un ensemble très vaste d'éléments préalablement identifiés par la recherche dans le domaine des collecticiels. Il permet la collaboration en temps réel au travers d'un espace *physique* partagé. Il illustre de plus l'intérêt de documents composites, l'espace étant ici considéré comme un document composite dont les différentes parties sont partagées. Cependant, nous pensons que son principal défaut est d'être lié à la notion d'espace physique partagé et, en conséquence, n'est pas adapté à la notion d'espace logique, *i.e.* de place sans espace physique[14]. Nous tenterons de combler cette lacune par le concept des "places publiques" (chapitre 6).

[14] Une liste de diffusion ("mailing list") est un exemple de place partagée sans espace physique.

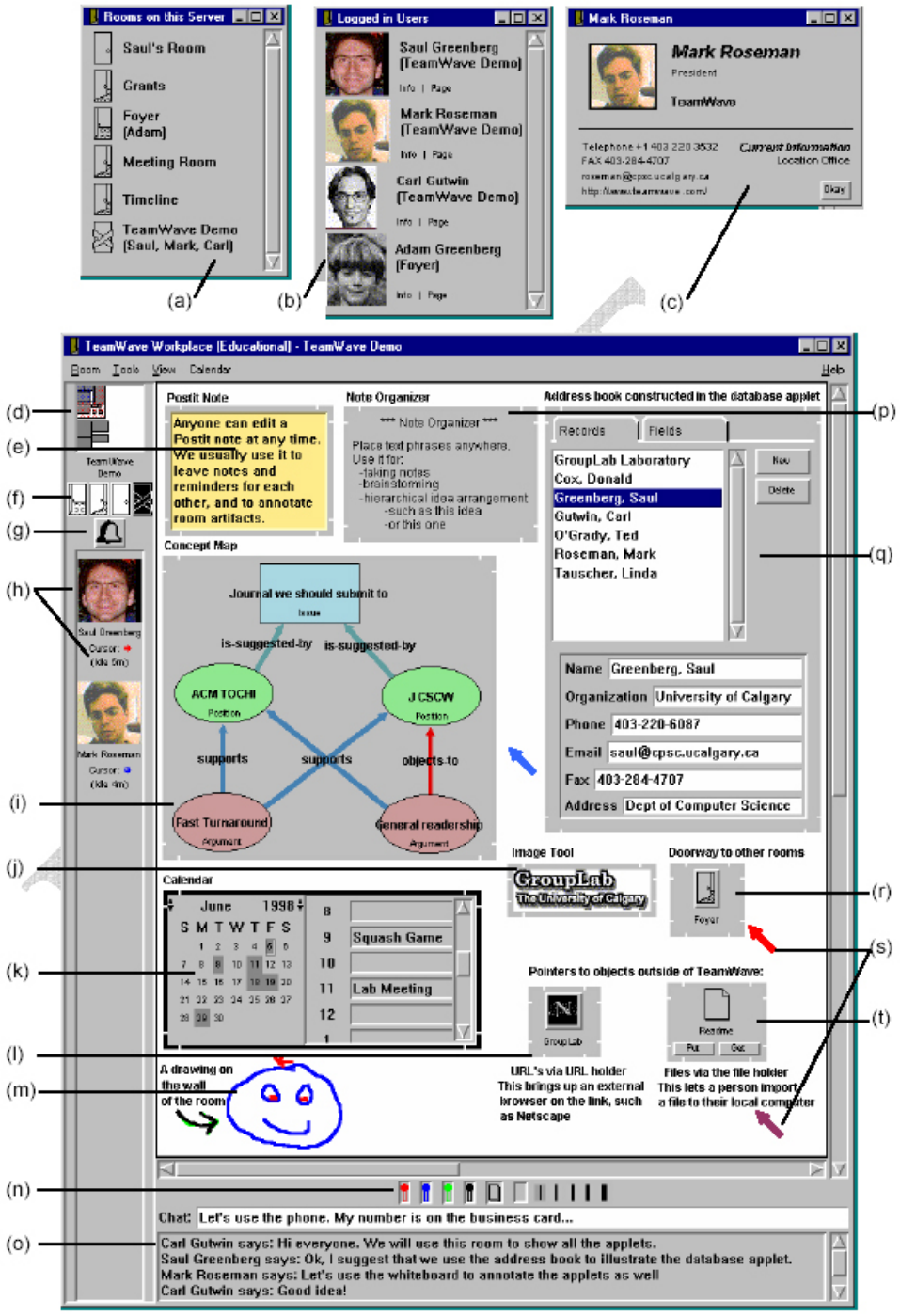


FIG. 1.25 – Une pièce partagée avec TeamWave – extrait de (Greenberg & Roseman, 2002)

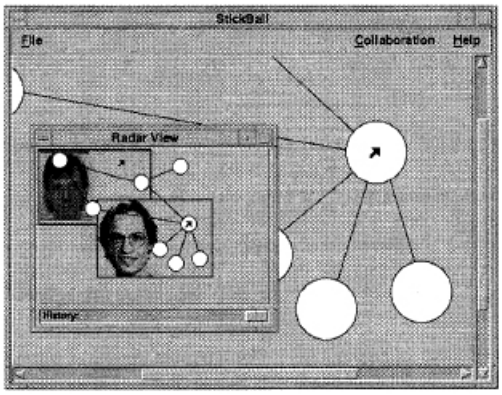


FIG. 1.26 – Vues “radar” – extrait de (Gutwin et al., 1996)

Un aspect essentiel dans la collaboration à distance est la conscience réciproque des actions des utilisateurs qui collaborent (ou “awareness”). Ceci reste vrai aussi bien pour les applications synchrones (qui fait quoi, et où) et les applications asynchrones (qui *a fait* quoi). Cet aspect devient encore plus important pour les espaces partagés car le nombre d’artefacts partageables simultanément peut devenir important, et l’espace a souvent des dimensions non négligeables. Dans la plupart des espaces de travail (collaboratifs), nous retrouvons ainsi l’idée de vue radar qui permet d’indiquer quelles zones de l’espace de travail sont visualisées par les différents utilisateurs. La figure 1.26 donne l’exemple de deux vues radar qui représentent en miniature les deux zones du graphique visualisées par deux utilisateurs. Nous reviendrons sur cet aspect de conscience mutuelle, essentiel pour les collecticiels, dans la section 1.5 de ce chapitre et dans la section 2 du chapitre suivant.

1.4.4.2 Espaces médiatisés

Nous avons présenté dans les sections précédentes différents systèmes basés sur une approche synchrone ou asynchrone de la collaboration. Moran & Anderson (1990) ont identifié trois aspects fondamentaux pour supporter le travail collaboratif à distance :

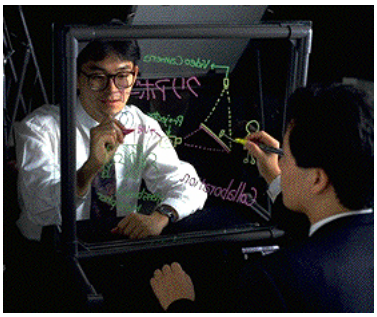
1. Les applications synchrones et les espaces de travail partagés permettent à des personnes de coopérer pour la réalisation de tâches communes en partageant différents artefacts (documents, vues, fenêtres, bureau, etc.).
2. Les applications asynchrones et les workflow permettent à des personnes de coopérer d’une manière structurée pour des processus largement pré-définis.
3. L’interaction informelle permet quant à elle à des personnes de coopérer d’une manière non planifiée et non structurée.

Le dernier point, que nous n’avons jusqu’ici pas abordé, met en jeu les interactions informelles que les personnes mettent en jeu lorsqu’elles sont proches les unes des autres. Ceci inclut l’expression des personnes et donc la gestuelle, le regard, l’attitude corporelle, l’expression du visage ainsi que l’intonation de la voix. Ces éléments d’interaction tout à fait essentiels dans le cadre des collecticiels relèvent *tous* d’actes de communication. L’idée des espaces médiatisés, ou “mediaspace”, est de reproduire le plus fidèlement possible ces éléments d’interaction informelle dans un espace géographiquement distribué. Ceci implique naturellement l’utilisation des canaux audio et vidéo au travers d’un réseau informatique.

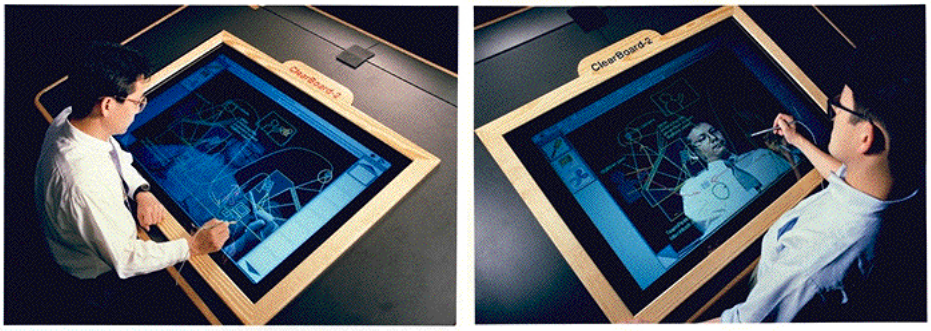
Le terme “Media Space” (espace médiatisé) a été inventé par Stults (1986) qui a mis en œuvre probablement le premier espace médiatisé. Le Media Space du XeroxPARC permettait de relier les centres de recherche de Palo Alto (Californie) et de Portland (Oregon).

Le système était à la fois utilisé comme outil pour les activités de recherche et comme sujet de recherche à proprement parler. Il a montré l'intérêt d'utiliser l'audio et la vidéo d'une manière intégrée au quotidien, de permettre ainsi de favoriser les échanges de nature informelle et d'améliorer la coordination entre individus.

Les années suivantes ont été très actives dans la recherche sur les mediaspaces. Nous ne citerons que quelques exemples extraits de (Mackay, 1999). Le système RAVE[15] a été installé dans le bâtiment du Xerox EuroPARC à Cambridge (Angleterre), bâtiment dont l'architecture ne favorisait pas la communication (Gaver et al., 1992). Ce système a été conçu de manière à favoriser "l'interaction sociale" et non à remplacer la communication en face-à-face. Le système Portholes a été développé afin d'aider les membres du laboratoire à rester en contact en capturant assez fréquemment un ensemble d'images fixes des différents bureaux "médiatisés" de PARC et de l'EuroPARC (Dourish & Bly, 1992). L'idée directrice est ici de proposer une approche asynchrone et passive qui permet de capturer une vue d'ensemble de plusieurs espaces médiatisés, conjointement à une approche synchrone et active par le biais d'un canal audiovisuel. Un autre exemple illustre une approche complémentaire des mediaspaces : le système Cruiser (Root, 1988). Le modèle de Cruiser utilise la métaphore d'une personne qui se déplace dans un couloir et jette un coup d'œil aux portes des bureaux afin d'engager éventuellement une "conversation"[16]. Le système est ainsi dédié au partage d'un espace médiatisé (un bureau) pour des périodes assez longues.



(a) ClearBoard-0



(b) ClearBoard-2

FIG. 1.27 – Le système ClearBoard – extrait de http ://web.media.mit.edu/~ishii/CB.html

Le système ClearBoard qui illustre très bien la transition des espaces partagés vers les espaces médiatisés puisqu'il fusionne ces deux aspects (Ishii et al., 1992). Ce système est basé sur une métaphore élégante : deux utilisateurs voient mutuellement au travers

[15]"Ravenscroft Audio Visual Environment"

[16]Notons que cette métaphore est liée à la culture américaine pour laquelle l'état d'une porte (fermée, ouverte ou entre-ouverte) a un sens très précis.

d'une vitre sur laquelle ils peuvent tracer des figures. Le prototype élémentaire, appelé ClearBoard-0, met en œuvre cette métaphore avec un matériel rudimentaire (figure 1.27-a). La vitre est ainsi la surface partagée, et la transparence de cette surface permet de maintenir une vision permanente de l'autre utilisateur qui se situe alors en léger arrière plan. Cette perception réciproque est un point essentiel puisqu'elle permet la conscience mutuelle des actions de chaque utilisateur, ce qui inclut aussi bien les gestes "formels" de tracé sur la vitre que de les actes informels exprimés par la gestuel ou le regard. A partir de ce prototype ont été réalisées les versions ClearBoard-1 puis ClearBoard-2, lesquels permettent aux deux participants d'être réellement distants. Le système ClearBoard-2 (figure 1.27-b) fait appel à des technologies assez sophistiquées et, par conséquent, assez onéreuses. Sa "vitre" est un écran à quatre couches : un miroir semi-réfléchissant, un film polarisant, l'écran de projection et une feuille transparente pour la mesure de la position du styler. Le système utilise une caméra pour la capture de l'image de l'utilisateur et un projecteur pour l'affichage sur l'écran de projection. Si dans le prototype ClearBoard-0, le dessin effectué par l'un des deux utilisateurs est perçu dans le sens inverse par l'autre utilisateur, le système ClearBoard-2 inverse par contre l'image de manière à pouvoir effectivement collaborer sur la surface "dans le bon sens"[17]. L'enseignement de ClearBoard est significatif : le système montre qu'il est possible de mettre en œuvre un système collaboratif dans un espace de travail partagé *et* médiatisé. Ainsi, ce système ne se contente pas d'adjoindre des canaux médiatisés à un espace de travail comme cela est souvent fait, mais réellement d'intégrer (par superposition) l'espace médiatisé *et* l'espace partagé. L'inconvénient de ClearBoard reste intrinsèque à la vitre et ses deux "seules" faces : la collaboration concerne deux utilisateurs seulement.

La démocratisation récente de l'Internet a permis d'étendre le champ des "mediaspaces" au domaine public. Ainsi, Windows XP contient par défaut l'application NetMeeting qui est un petit système de visioconférence à deux et auquel est adjoint d'autres outils de collaboration tels qu'une application de "chat" et un tableau blanc partagé. La boîte à outils Mediascape de Roussel (1999) illustre la possibilité d'étendre le champ des espaces médiatisés par le vecteur Internet. Elle permet d'intégrer la vidéo à des pages Web à partir d'une simple page écrite en HTML, et d'étendre les possibilités de communication vidéo à un réel mediaspace, en facilitant l'accès à plusieurs sites, la notification des utilisateurs et le contrôle d'accès.

Les différents exemples que nous avons donnés illustrent l'intérêt des espaces médiatisés quant à la facette informelle de la communication entre individus. Cet aspect est très important puisque nos actes de communication quotidiens sont liés aux gestes et aux expressions. Ainsi, il est essentiel de garder à l'esprit cette facette pour la conception des espaces de travail collaboratifs : ces derniers doivent permettre une communication aussi bien informelle par le biais des canaux audiovisuels, que formelle par le biais d'éléments partagés de l'espace de travail. Rappelons que cet aspect avait déjà été mis en avant il y plus de 25 ans par Engelbart & English (1968), et que le système ClearBoard est probablement la plus aboutie des approches intégrant les deux facettes formelle et informelle en un seul espace de travail interactif.

1.4.4.3 La co-présence

Lorsque le terme de collecticiel est mentionné, il évoque le plus souvent la collaboration distante. Cependant, comme le mentionnent Stewart et al. (1999), la collaboration basée sur la co-présence des intervenants offre un champ de possibilités tout à fait intéressant et qui ne saurait être atteint en passant par un réseau. Dans le cas de la co-présence, l'écran

[17] En conséquence, chaque utilisateur voit une image inversée de l'autre utilisateur, comme s'il le voyait dans un miroir.

est partagé entre les différents utilisateurs qui collaborent ; l'écran est donc la surface physique partagée. Ce mode de collaboration est appelé "single display groupware" (SDG). L'application KidPad est un bonne illustration de l'intérêt de collaborer autour d'un même écran (Druin et al., 1997). Il permet à plusieurs enfants d'utiliser des outils de dessins sur une même surface de travail et de manière simultanée. Les enfants manifestent notamment leur intérêt par le fait qu'ils n'ont pas besoin de s'échanger la souris pour dessiner sur la même "feuille", et qu'ils peuvent librement intervenir sur le dessin d'autrui. Ils peuvent ainsi dessiner simultanément, ce qui demeure difficile, voire impossible, avec des crayons et une feuille réelle.

Il peut paraître curieux que l'idée de SDG ne tende que récemment à devenir un nouvel aspect de la collaboration. Pourtant, la boîte à outils pour la construction d'interfaces homme-machine MMM proposait un ensemble de services permettant de réaliser simplement une application basée sur la co-présence (Bier & Freeman, 1991). Elle permettait entre autres de gérer plusieurs périphériques d'entrée pour le pointage (plusieurs souris par exemple) pour une même application.

Cette émergence tardive provient probablement du fait pré-cité, à savoir que le collecticiel évoque plus souvent l'idée de collaboration à distance que la co-présence. L'ordinateur, même s'il est de nos jours très souvent connecté à un réseau, est généralement considéré comme un objet *personnel* : l'idée de partager cet objet n'est ainsi pas entrée dans les mœurs des utilisateurs. Une autre raison complète probablement ce fait : la connexion de plusieurs périphériques d'entrée n'était pas nécessairement une chose aisée sur les machines et systèmes d'exploitation ne disposant pas d'un "bus" adapté[18], et les écrans restaient de petites tailles alors qu'aujourd'hui, les projecteurs vidéo permettent de projeter le contenu d'un écran sur un large tableau blanc.

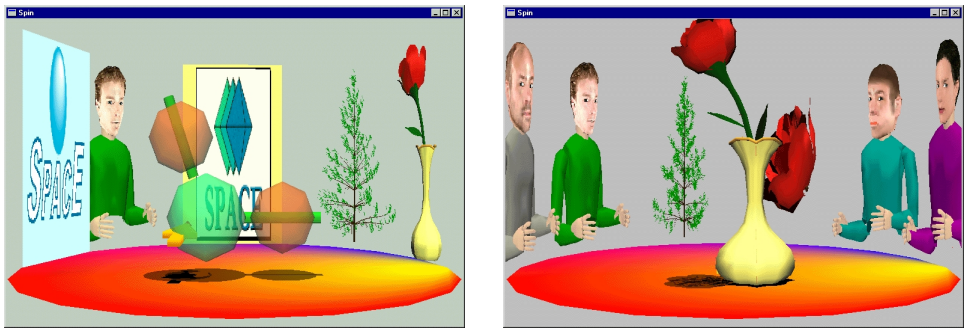
1.4.4.4 Autres approches

Réalité virtuelle Les environnements interactifs actuels sont basés sur un certain nombre de métaphores, tels que le bureau, les dossiers, les documents ou les boutons. Une métaphore permet de transposer un objet ou un concept présent dans le monde réel dans l'environnement informatisé augmentant ainsi la facilité d'utilisation de cet environnement : les connaissances que les utilisateurs ont de l'objet réel peuvent être transférées depuis le monde réel jusque dans l'environnement "virtuel", minimisant ainsi l'effort d'apprentissage. Le principe de la réalité virtuelle est de pousser cette approche au plus haut niveau en transposant le plus rigoureusement possible l'espace réel dans l'environnement informatique. L'espace réel est alors rendu par une représentation visuelle en trois dimensions appelée la scène et dans lequel l'utilisateur peut se déplacer et interagir. Faisant partie de cet espace réel, l'utilisateur *doit* être immergé dans l'espace "virtualisé" ; il perçoit alors sa propre image sous la forme d'un *avatar*. Cette sensation d'immersion est souvent renforcée *via* des lunettes reproduisant notre vision stéréoscopique (vision 3D).

La capacité de créer des scènes virtuelles dans lesquels les utilisateurs sont représentés par des avatars a été mise à profit pour créer des environnements collaboratifs virtuels. Dans de tels environnements, la scène virtuelle est l'élément partagé entre les utilisateurs.

L'application Spin (figure 1.28) propose une interface 3D basée sur une table métaphorique (Carlier et al., 1997; Dumas et al., 1998). Les participants, visibles sous la forme d'avatar, et les documents sont placés tout autour de cette table. Un mécanisme simple permet de regarder les objets de la scène et d'en s'en rapprocher tout en les maintenant visibles à tout moment. La représentation continue de la scène permet de rendre perceptibles les liens entre les participants et les objets. Cette approche est originale car elle repose sur un ensemble d'abstractions ne respectant pas la géométrie et les actions réelles. Ceci permet

[18] Le bus USB (Universal Serial Bus) ou le bus ADB (Apple Desktop Bus) par exemple.



(a) Manipulation d'objets (b) Collaboration

FIG. 1.28 – Spin – extrait de http :// www.lifl.fr/ GRAPHIX/ rechfonda/ space

d'offrir des possibilités nouvelles aux utilisateurs, lesquelles n'ont pas d'équivalent dans l'espace réel.

Si l'utilisation de la réalité virtuelle semble intéressante et attractive, elle induit des problèmes qui vont jusqu'à contredire les intérêts initiaux prônés par ses défenseurs. Pekkola (2002) met clairement en avant ce fait en montrant pourquoi les cinq avantages des environnements collaboratifs virtuels prônés par Singhall & Zyda (1999) ne sont finalement pas recevables :

Perception commune de l'espace – Comme l'indique (Harrison & Dourish, 1996), la notion de place est plus importante que la notion d'espace. Le seul cas où une visualisation 3D est intéressante concerne les situations et les tâches pour lesquels l'espace à un sens précis. Cependant, la réalité virtuelle n'a généralement pas d'avantages sur les autres média pour recréer l'illusion de place.

Perception commune de la présence d'autrui – Le plus souvent, une simple liste d'utilisateurs suffit à fournir cette perception.

Apport pour la communication – La réalité virtuelle n'apporte rien de significatif concernant l'aspect communication puisque cette communication est en fait réalisée *via* d'autres médias.

Partage d'objets – La manière de partager les objets dans l'espace n'a de sens que pour les objets réellement 3D. Dans les autres cas, le partage est réalisé de manière plus simple en utilisant d'autres médias.

Perception commune du temps – Ce point reste tout aussi vrai pour les collecticiels temps réel.

Nous pouvons de plus nous questionner sur la nécessité d'imiter le monde réel : n'est-il pas plus judicieux de créer un environnement réellement adapté aux besoins de la collaboration distante ? Ce point avait déjà été soulevé avant l'émergence de la réalité virtuelle par Hollan & Stornetta (1992) : le collecticiel doit apporter plus que simplement récréer – et donc simuler – la présence des utilisateurs distants.

Pour cet ensemble de raisons, nous ne prenons pas en compte la réalité virtuelle dans notre approche de l'espace de travail interactif et collaboratif. Notons cependant que le modèle que nous proposons reste compatible avec la notion d'espace virtuel puisqu'il est basé notamment sur l'idée de chaînage des actions depuis les périphériques d'entrées jusqu'aux objets ciblés, ces derniers pouvant parfaitement être des objets virtuels en trois dimensions. Nous ne proposerons cependant pas de modèle pour la modélisation d'objet 3D dans l'espace.

Réalité augmentée Une approche radicalement opposée à la réalité virtuelle et appelée réalité augmentée consiste à immerger l'ordinateur dans le monde réel (celui de l'utilisateur), et non plus l'utilisateur dans un monde virtuel (celui de l'ordinateur) :

> "La réalité virtuelle, bien plus que la station de travail, nous coupe et nous exclut du monde dans lequel nous vivons, travaillons et jouons. Une autre approche est en train d'émerger en prenant une approche radicalement opposée à la réalité virtuelle. Plutôt que d'utiliser l'ordinateur afin d'enfermer les utilisateurs dans un monde artificiel, nous pouvons utiliser l'ordinateur pour *augmenter* les objets dans le monde réel. (...) Les environnements augmentés tendent à amener des systèmes électroniques dans le monde physique plutôt que de les remplacer par des artefacts virtuels et purement informatiques." traduit de (Wellner et al., 1993)

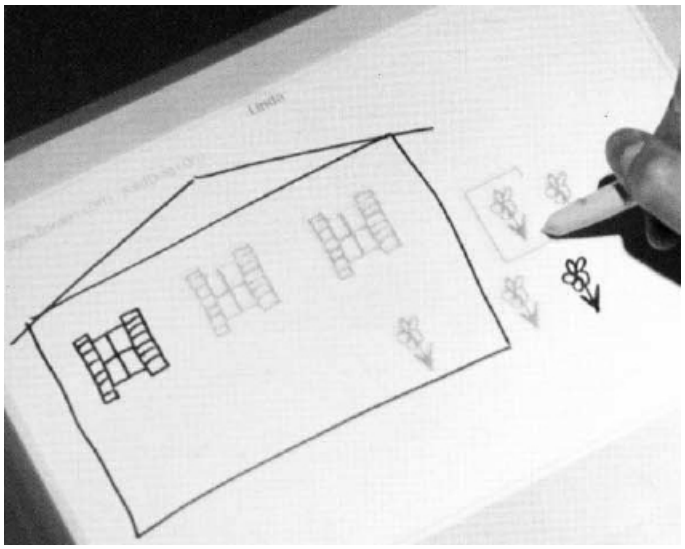
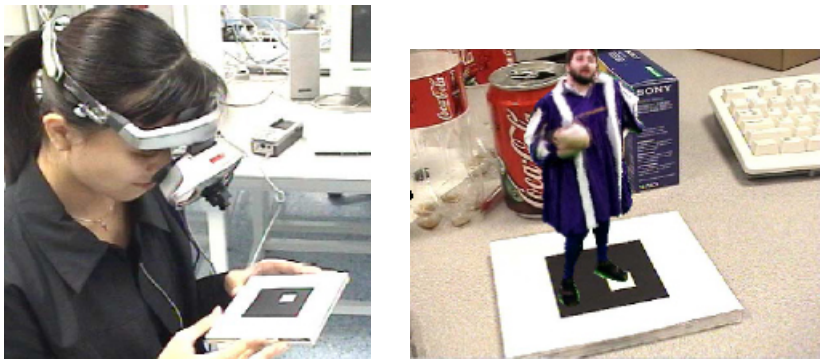


FIG. 1.29 – Copier/coller avec l'application PaperPaint du DigitalDesk – extrait de (Wellner, 1993)

L'exemple du système DigitalDesk illustre bien le concept de la réalité augmentée (Wellner, 1993). Ce système propose d'augmenter le document papier ordinaire en rendant possibles des actions normalement réservées aux seuls documents électroniques, comme par exemple le copier /coller. Le système consiste à filmer la feuille de façon à recueillir les informations physiques qui y figurent (elles sont tracées avec un feutre réel par exemple) et conjointement, à projeter sur cette même feuille des informations supplémentaires. Dans l'exemple de la figure 1.29, l'utilisateur a réalisé le dessin d'une maison et d'une fleur à l'aide d'un feutre réel. En utilisant l'application PaintPaper, il sélectionne une portion du dessin et peut alors la copier électroniquement à d'autres emplacements en utilisant la projection. Le contenu résultant du papier est ainsi mixte : il contient le dessin réel initial ainsi que le dessin augmenté des parties copiées et projetées sur le papier. Une extension intéressante de ce principe consiste à créer des documents papiers purement électroniques à partir d'un catalogue réel dans lequel l'utilisateur vient "piocher" des éléments pour les insérer sur le papier (action appelée "pick and drop" par Rekimoto (1997)).

L'approche des systèmes par la réalité augmentée nous semble pertinente dans le sens où elle tente de faire disparaître l'interface entre la personne et la machine en "immergeant" la machine dans l'espace physique. Le modèle proposé dans cette thèse étant basé sur la

notion de document “informatique”, c’est à dire un document dont le système connaît à l’avance la *structure* et le *type* de données qu’il comporte, il sera pertinent de voir comment la notion originale de document peut-elle être “augmentée” afin de pouvoir être un document papier physique, à l’instar du papier augmenté du DigitalDesk. Nous n’abordons cependant pas cet aspect dans cette thèse puisque nous jugeons que “l’augmentation” du document ne relève pas spécifiquement du modèle DPI : il peut être abordé dans un contexte plus générale, *i.e.* indépendamment de DPI.



(a) Casque de visualisation (b) Représentation d’un collaborateur

FIG. 1.30 – Télé-présence avec le système “3-D Live” – extrait de (Prince et al., 2002)

Réalité mixte Une dernière approche consiste à augmenter l’espace de l’utilisateur par des objets purement virtuels. En agissant de la sorte, l’utilisateur n’est pas immergé dans un monde virtuel, ce qui évite ainsi les problèmes inhérents à la réalité virtuelle susmentionnés. Cette approche est appelée “réalité mixte” car elle mixe la réalité et le virtuel[19]. Dans les systèmes à réalité mixte, l’utilisateur porte typiquement un casque de visualisation (“Head Mounted Display” ou HMD, figure 1.30-a) lequel affiche des informations qui se superposent à l’environnement physique. Par exemple, le système 3-D Live (Prince et al., 2002) permet d’afficher une représentation d’un collaborateur distant dans un endroit précis de l’espace de travail de l’utilisateur (figure 1.30-b). L’image du collaborateur est capturée par un ensemble de caméras qui permettent par un algorithme adapté de reconstituer sa silhouette depuis n’importe quel point de vue (la silhouette étant définie comme l’image du collaborateur sans l’arrière plan). La position du point de vue de l’utilisateur est déterminée en fonction des positions respectives du réceptacle carré de la silhouette (le carré blanc contenant un carré noir, figures 1.30-a et b) et du dispositif HMD qu’il porte. Le résultat est alors assez surprenant : l’utilisateur perçoit le collaborateur sous la forme d’un avatar qui semble réellement “présent” sur le réceptacle carré.

L’inconvénient principal de l’approche à réalité mixte tient dans la difficulté (voire l’impossibilité) de collaborer de manière bi-directionnelle. Dans l’exemple précédent, l’utilisateur perçoit le collaborateur, mais l’inverse n’est pas vrai. Cette limitation est inhérente au dispositif HMD invasif mais nécessaire pour construire la “mixité” espace réel plus objet virtuel (1.30-a) : ce dispositif ne permet pas d’établir un contact visuel avec la personne qui le porte.

[19] Notons que, très souvent, le terme “réalité augmentée” est utilisé par abus de langage pour désigner la réalité mixte.

1.5 Conclusion

L'expérience du Xerox Star nous a montré qu'il est possible de concevoir un système simple, intuitif *et* puissant ("simple yet powerful"). Les systèmes actuels ne suivent pas cette tendance et visent, probablement pour des raisons commerciales, à accroître toujours plus le nombre des fonctionnalités offertes au détriment de la simplicité d'utilisation. Nous pouvons de plus constater que les principes de la manipulation directe ne sont pas souvent mis en pratique.

Les techniques d'interaction telles que l'interaction à base de gestes et l'interaction bimanuelle élargissent le champs des possibilités offertes par les espaces de travail. Elles vont au delà de ce que le tandem clavier + souris offre. D'une manière tout à fait analogue, les techniques de visualisation telles que la distorsion de l'affichage, les interfaces zoomables et les approches diverses du fenêtrage sont rarement mises en œuvre dans les systèmes actuels. Ces différentes techniques sont pourtant réalisables aujourd'hui étant donnés les progrès réalisés tant sur les périphériques d'entrée que sur les cartes graphiques.

Bien que le document soit au centre de l'activité de l'entreprise, les systèmes actuels sont davantage centrés sur la notion d'application que sur la notion de document, et les applications interopèrent rarement et difficilement. L'approche d'OpenDoc semblait prometteuse mais n'a pas abouti. Elle a cependant planté les bases de l'approche centrée sur les documents. Notre modèle s'en inspire mais propose une granularité plus fine tant du point de vue de l'interaction que la représentation (graphique).

Enfin, malgré l'engouement actuel pour tout ce qui a attrait à la communication à distance, les systèmes d'exploitation n'offrent pas de réels services pour gérer la collaboration. Par conséquent, chaque application doit pendre en charge les aspects standards liés à la collaboration distante tel que la concurrence d'accès et la gestion de la cohérence. Les espaces de travail actuels devraient permettre de collaborer de manière synchrone ou asynchrone, en partageant un espace de travail ou non.

Nous proposons dans cette thèse un modèle qui permet d'élaborer des espaces de travail interactifs et collaboratifs en tentant de répondre aux exigences précédentes. Ce modèle est conçu selon une approche d'abord théorique puis pratique. La boîte à outil OpenDPI a été réalisée afin de tester la "faisabilité" du modèle théorique ainsi que sa pertinence. Son architecture reste très différente des boîtes à outils actuelles qui restent presque toujours centrées sur le concept de "widget".

Chapitre 2

Fondements pour un modèle d'interaction et de collaboration

Contents

L'état de l'art du chapitre 1 nous a permis de préciser un certain nombre de notions relatives à l'espace de travail interactif et collaboratif que notre modèle doit permettre de réaliser. Nous effectuons dans ce chapitre un état de l'art des principes jugés fondamentaux pour le modèle DPI. Ces principes définissent les fondements qui servent à la construction du modèle tant du point de vue théorique (chapitre 3) que du point de vue pratique (chapitre 4).

Le chapitre est divisé en trois sections. La section 2.1 présente les fondements de la facette interactive du modèle. Ils prennent en considération la manipulation directe, les interfaces utilisateur orientées objet, les interacteurs, l'interaction instrumentale, le document et le composant logiciel interactif. La section 2.2 présente les fondements de la facette collaborative de DPI. Elle propose la métaphore de la "place", décrit l'aspect temporel de la collaboration, précise l'idée d'action dans un contexte collaboratif et le concept de composant collaboratif. La section 2.3 résume l'ensemble des fondements retenus pour notre modèle.

2.1 Fondements pour la facette interactive

2.1.1 Manipulation directe

Les premiers principes fondamentaux des interfaces graphiques modernes ont été établis par Shneiderman (1983) au travers du paradigme de la manipulation directe. Ce principe est centré sur l'utilisateur et la satisfaction que ce dernier peut (et doit) éprouver lorsqu'il utilise un système interactif. Ce dernier point est effectif les aspects suivants sont respectés :

1. L'utilisateur *maîtrise* le système.
2. Il accomplit des tâches de manière *efficace*.
3. Il *apprend* aisément à utiliser l'environnement avec une assimilation progressive de ses fonctionnalités avancées.
4. Il reste *confiant* quant à sa capacité à rester maître du système au fil du temps.
5. Il utilise le système avec *plaisir*.
6. Il a envie de faire *partager* ce système avec les utilisateurs novices.
7. Il a envie d'*explorer* des aspects plus évolués du système.

Le paradigme de manipulation directe est proposé dans le but de rendre effectifs ces différents points. Ses idées centrales, qu'il est essentiel de conserver à l'esprit pour la conception de notre modèle, sont ainsi synthétisées :

1. Les objets d'intérêt sont représentés de *manière continue*.
2. Les actions demeurent physiques et ne relèvent *pas de commandes à syntaxe complexe*.
3. Les opérations sont *rapides*, *incrémentales* et *réversibles*. Elle produisent un impact immédiatement visible sur les objets d'intérêt.
4. L'approche de l'apprentissage se fait en couches et en *spirale* de manière à minimiser les connaissances préalablement requises.

Ces quatre principes permettent de concevoir des systèmes possédant les caractéristiques bénéfiques suivantes :

– Les débutants peuvent apprendre les fonctionnalités élémentaires rapidement, généralement *via* une démonstration effectuée par une personne plus expérimentée, ou en explorant le système.

– Les experts peuvent travailler très rapidement afin de mener à bien un grand nombre de tâches, même si cela implique de définir de nouvelles fonctionnalités.
– Les messages d'erreur sont rarement nécessaires.
– Les utilisateurs peuvent voir si leurs actions mènent au résultat escompté et, dans le cas contraire, ils peuvent modifier la stratégie de manière à parvenir à ce résultat.
– Les utilisateurs sont plus rassurés par un système qui reste compréhensible et qui permet la réversibilité des actions qu'ils effectuent.
– Les utilisateurs gagnent en confiance et en maîtrise parce qu'ils initient les actions, les contrôlent, et peuvent en prévoir le résultat.

Enfin, Shneiderman soulève également les problèmes intrinsèques à ce paradigme, notamment ce qui touche à la représentation graphique :

– Les représentations des objets de l'interface peuvent être tronquées ce qui engendre alors une perte du contexte.
– La signification d'une représentation d'un composant (icone par exemple) n'est pas toujours simple et peu dépendre d'aspects culturels variables d'un pays à l'autre.
– L'utilisateur peut faire certaines fois des conclusions erronées sur les opérations permises (notion de croyance).
– L'espace utilisé par les objets graphiques peut devenir rapidement excessif par rapport à la taille de l'écran.

L'approche de la manipulation directe met ainsi en avant la manipulation d'objets souvent empruntés au monde réel (approche métaphorique), cette manipulation se faisant "au plus près" de l'objet et en faisant appel à une gestuelle proche de celle que nous utilisons dans la vie courante. L'idée que l'objet est l'élément central d'un espace de travail interactif est ainsi omniprésente dans l'approche de la manipulation directe ; elle s'est développée dans les années qui ont suivi sous le terme des "interfaces utilisateur orientées objet"[1].

2.1.2 Interfaces utilisateur orientées objet

Les premiers systèmes interactifs ont mis en avant la manipulation d'*objets* au travers d'une interface principalement graphique. Ceci a eu pour conséquence l'apparition du terme objet tant du point de vue de l'interface homme-machine que du point de vue de la programmation. Le langage SmallTalk, né au début des années 1970[2], a été pensé aussi bien en terme d'environnement de développement qu'en terme de langage de programmation ("co-design"). La naissance des interfaces graphiques à manipulation directe et la naissance de la programmation orientée objet (POO[3]) ont des racines réellement communes. Ainsi les interfaces à manipulation directe et la programmation orientée objet ont des similitudes qu'il est important d'avoir à l'esprit.

2.1.2.1 Programmation orientée objet

Dans l'approche impérative des langages de programmation tels que le C ou le Pascal, le comportement d'un programme dépend à la fois des algorithmes et des structures de données qu'il met en œuvre. Les algorithmes sont typiquement codés dans des fonctions et les structures de données manipulées étant considérées indépendamment de ces fonctions. Du point de vue de la POO, le comportement d'un programme est régi par un ensemble d'objets qui représentent typiquement des objets du monde réel. Donnons brièvement les caractéristiques essentielles de la POO :

[1] "Object Oriented User Interfaces" ou OOUI

[2] Le langage SmallTalk a existé selon trois versions successives, SmallTalk-72, 76 puis 80, ces numéros de version correspondant aux années de commercialisation (1972, 1976 et 1980).

[3] "Object Oriented Programming" ou OOP

Encapsulation – Un objet *encapsule* à la fois les fonctions *et* les données qui le concernent. Il définit son propre comportement par le biais de "fonctions membres" et masque sa structure de données dans des "données membres".

Composition – Les objets peuvent être *composés* les uns avec les autres pour former des objets de plus forte granularité. Un exemple classique concerne l'objet "bicyclette" qui est composé d'objets de type "roue", "cadre", "guidon", etc.

Classes – La notion de *classe* permet de catégoriser les objets les uns par rapport aux autres. Une classe est un modèle d'objet à partir duquel un programme peut créer de nouveaux objets ; ces derniers sont appelés instances de la classe considérée. Le concept de classe différencie nettement la POO de la programmation impérative basée sur les fonctions : les fonctions deviennent des fonctions membres de la classe, ce principe s'appliquant aussi aux données.

Héritage – De nombreuses classes d'objets ont des fonctions et des données membres communes avec d'autres classes. Il est intéressant dans ce cas de pouvoir factoriser leurs comportements communs en une classe unique dite classe de base. Le mécanisme d'*héritage* autorise une telle factorisation : lorsqu'une classe hérite d'une classe de base, elle possède les fonctions et le données membres telles qu'elles sont définies dans la classe de base. Un exemple d'héritage de classes : la classe des "Formes géométriques" est une classe de base et les classes des "Rectangles" et des "Polygones" sont des classes héritant de la classe des "Formes géométriques".

Classes abstraites – Dans l'exemple précédent, la classe des "Formes géométriques" est *abstraite* car il ne peut exister un objet qui ne soit qu'une forme géométrique : il s'agit nécessairement d'une forme particulière tel qu'un rectangle ou un polygone. Les rectangles et les polygones sont concrets par nature et les formes géométriques représentent alors une classe qualifiée d'abstraite. L'existence de la classe des "Formes géométriques" est intéressante car il est possible d'y définir certaines fonctions de manipulation des formes au sens général, telle que leur translation, leur rotation ou le calcul de leur dimensions, sans nécessairement les implanter.

L'approche de la POO est devenue effectivement populaire dès le début des années 1990, les arguments commerciaux mis en avant pour la POO étant la modularité, l'extensibilité et la maintenabilité des applications. Curieusement, l'idée d'objet du point de vue de l'utilisateur n'a pas eu cette popularité bien que la POO soit née des systèmes interactifs précurseurs. Ceci est probablement dû au fait que l'industrie informatique s'intéressait davantage au côté application qu'au côté utilisation, c'est à dire aux langages de programmation et de modélisation[4].

2.1.2.2 Interfaces orientées objet

Le terme "d'interface utilisateur orientée objet" (IUOO) a été cependant largement défendu par IBM dans les années 1990. Collins (1995) souligne les similitudes entre les approches POO et IUOO et montre que le bénéfice de l'approche orientée objet peut concerner les interfaces utilisateurs tout autant que la programmation. Ces similitudes sont, d'une manière synthétique, les suivantes :

Objets – Un objet est une entité dont l'*utilisateur* a besoin pour réaliser une tâche particulière. C'est une entité qui peut être manipulée en tant qu'élément indépendant (un document sur le bureau par exemple) ou qui peut être perçue par l'utilisateur comme étant capable d'exister de manière indépendante (un mot dans un texte par exemple). Les objets sont organisés dans l'espace de travail de l'utilisateur, organisation qui rejoint celle des objets du monde réel.

[4] Ce fait est peut-être encore aujourd'hui d'actualité...

Ils sont typiquement représentés à l'écran sous la forme d'icones, un icone étant une petite image qui facilite l'identification d'un objet par l'utilisateur sans avoir à connaître toutes les informations relatives à l'objet. Il est typiquement constitué d'un titre qui permet d'identifier l'objet en particulier et d'une image qui permet d'identifier sa classe.

Encapsulation – L'utilisateur appréhende les différents objets sans pour autant savoir comment les comportements de ces objets sont effectivement codés au travers des fonctions et données membres. Ce constat rejoint la métaphore suivante : le conducteur d'une voiture n'a pas nécessairement besoin d'être un expert en mécanique automobile pour comprendre comment utiliser son véhicule. L'utilisateur manipule les objets de l'interface connaissant uniquement leurs fonctionnalités, c'est à dire les fonctions membres publiques.

Composition – Les objets des interfaces graphiques sont naturellement des objets composés. Un dossier contient par exemple des documents, chaque document contenant par exemple du texte, des graphiques ou des tableaux. La composition spatiale aide l'utilisateur à organiser et structurer son espace de travail ainsi que ses documents, et ainsi à pouvoir se concentrer sur les objets élémentaires lorsqu'il s'agit de les éditer (édition d'une partie d'un document par exemple).

Classes – L'utilisateur perçoit de manière plus ou moins consciente que différents objets de son espace de travail ont des comportements semblables et peuvent donc être catégorisés en classes : la classe des documents, des fenêtres, des boutons ou des applications par exemple.

Héritage – La classe des dossiers est un bon exemple de classe de base. Un dossier est un élément de l'espace de travail qui peut contenir des documents ou d'autres dossiers. La corbeille, la file d'attente d'impression ou le porte-documents sont des instances de classes héritant de la classe des dossiers. Pour l'utilisateur, les différentes actions qu'il peut réaliser avec les dossiers (comme y déplacer un document) sont également réalisables sur les objets dérivés corbeille, porte-document ou file d'impression.
L'avantage pour l'utilisateur est considérable : il peut *transférer* ses connaissances acquises pour une classe aux différentes classes dérivées. Par exemple, quel que soit le type de dossier concerné, il sait qu'il peut y déposer des documents ou en extraire.

Classes abstraites – La plupart des objets de l'espace de travail sont des objets pouvant être représentés sous la forme d'un icone. En manipulant les icones, l'utilisateur comprend qu'il existe des points communs entre tous ces icones bien qu'ils désignent des objets de nature très diverse. Par exemple, l'icone peut être ouvert pour afficher pleinement l'objet qu'il désigne, tout comme il peut subir l'action "glisser-déposer" d'un point à l'autre de l'espace de travail.
Il est ainsi possible de définir la classe abstraite des objets "iconifiables" puisqu'il existe des comportements semblables entre les différents types d'icone (icone de document, de dossier, ou d'application par exemple). Le fait que la classe soit abstraite ne fait pas perdre l'avantage de l'héritage mentionné au point précédent : il y a là encore transfert des connaissances acquises lors de la manipulation des icones de différentes natures.

Ces différents points montrent ainsi que la notion d'héritage est une relation typique de la POO qui joue également un rôle important dans l'utilisabilité des objets de l'espace de travail.

2.1.2.3 Trois principes de conception

D'une manière tout à fait complémentaire à l'approche OOUI, Beaudouin-Lafon & Mackay (2000) définissent trois principes essentiels pour la conception des interfaces graphiques. Il s'agit :

1. de la réification,
2. du polymorphisme,
3. et de la réutilisabilité.

La *réification* est le processus qui transforme les concepts en objets, lesquels deviennent alors manipulables par l'utilisateur. Citons quelques exemples extraits de (Beaudouin-Lafon & Mackay, 2000). Une commande correspond à une action que l'utilisateur peut appliquer sur un objet ciblé, tel que le déplacement d'un objet graphique. La notion de commande peut être réifiée en un outil (ou instrument) tel que l'outil "main" permettant de déplacer les objets mobiles. Un style de caractère est un ensemble d'attributs qui précise le rendu graphique des caractères, tels que le nom de la fonte, sa taille, sa mise en gras ou/et en italique. La notion de style peut être réifiée en un outil de type "brosse" permettant l'application du style associé aux caractères visés. Enfin, l'alignement de formes géométriques peut être réifié en un objet de type "guide" : lorsqu'un objet est positionné près du guide, il y est lié et devient par conséquent aligné avec les autres objets liés à ce guide. Dans ces différents cas, il s'agit du concept général de commande (déplacement, application d'un style, alignement) réifié en différents instruments selon la nature de la commande. La réification est ainsi un principe précis qui favorise la manipulation directe par le simple fait de substituer à la notion d'action utilisateur l'idée d'objet interactif prenant en charge le paramétrage de l'action considérée (la spécification d'un style par exemple) ainsi que l'application de l'action sur les objets ciblés.

Le *polymorphisme* est une caractéristique intrinsèque des objets manipulés qui permet à une même commande d'être appliquée à un ensemble d'objets de natures différentes. Par exemple, l'application d'une couleur à une forme géométrique peut consister à "peindre" le fond de la forme avec cette couleur, alors que son application à une image peut consister à "teinter" chaque pixel de manière à accentuer cette couleur. Le polymorphisme permet alors de maintenir un nombre réduit de commandes. Cette faculté reprend ainsi un principe clé du Xerox Star qui consiste à améliorer les applications par la simplification : cette dernière est induite par la factorisation des commandes et non par leur sur-multiplication. Par ailleurs, le concept de polymorphisme peut être étendu à la notion de groupe d'objets. Une commande applicable à un objet peut également s'appliquer à un groupe d'objets de même type, la commande étant alors exécutée sur chaque objet du groupe. Par extension, une commande peut être appliquée à un groupe d'objets de types différents pour autant que cette commande ait un sens pour chacun des objets concernés.

Le dernier principe, la *réutilisabilité,* concerne la capacité de ré-exécuter les commandes préalablement appliquées par l'utilisateur ("input reuse") et que les (objets) résultats de ces commandes peuvent être également réutilisés ("output reuse"). Le premier cas correspond typiquement à la commande "refaire[5]" et le second à la duplication et au copier-coller. Cette approche par la réutilisabilité permet un accroissement de l'efficacité de l'interface graphique en autorisant des "opérations rapides, incrémentales et réversibles" indispensables en manipulation directe.

Les trois principes de l'OOUI, l'encapsulation, la composition et l'héritage, conjointement aux trois principes de Beaudouin-Lafon & Mackay (2000), la réification, le polymorphisme et la réutilisabilité, définissent un ensemble de règles que notre modèle prendra pleinement en compte.

2.1.3 Interacteurs

La notion d'interacteur a été inventée lors du développement de la boîte à outils Garnet, laquelle est dédiée à la construction d'interfaces graphiques (Myers et al., 1990). Myers (1990) caractérise les interacteurs de la manière suivante :

[5]Cette commande est généralement conjointe à la commande "défaire" ("undo / redo" en anglais).

> "Les interacteurs sont des objets qui peuvent gérer l'ensemble des techniques d'interaction conventionnelles basées sur l'utilisation du clavier et de la souris. Il s'agit des interactions relatives aux menus, barres de défilement et boutons, ainsi que des interactions tels que la sélection et le déplacement de rectangles et de lignes fléchées dans un éditeur de graphes" (Myers, 1990, page 289)

La notion d'interacteur facilite grandement le développement des interfaces utilisateur. L'indépendance des interacteurs vis-à-vis des objets sur lesquels ils interagissent en est la raison principale. Les apports des interacteurs sont ainsi résumés :

- Les interacteurs restent indépendants des objets graphiques et des applications ; ils peuvent en conséquence être spécifiés séparément.
- Les détails de la gestion des entrées sont masqués pour le développeur, rendant ainsi la gestion de l'interaction indépendante du gestionnaire de fenêtre (c'est généralement ce dernier qui distribue les événements aux applications).
- Leur utilisation renforce la "réutilisabilité" du code.
- Les interacteurs permettent de définir des comportements différents attachés à un même objet, sans se soucier des détails de l'interaction liée à ces comportements.
- Ils permettent l'utilisation simultanée de plusieurs dispositifs d'entrée (typiquement plusieurs dispositifs de pointage).

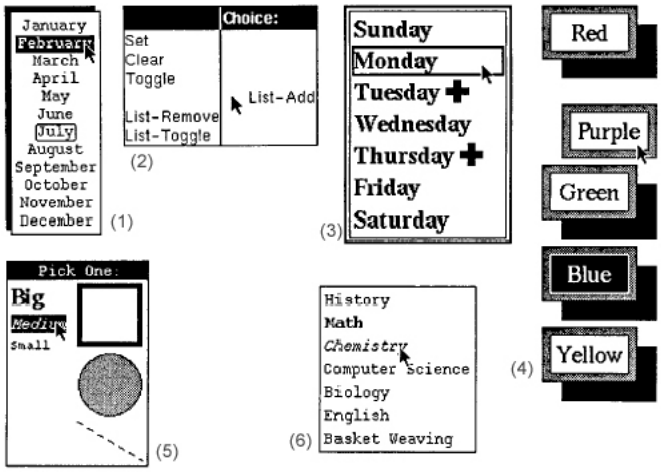
Le nombre d'interacteurs défini dans Garnet s'élève à seulement six. Ces six interacteurs permettent cependant de traiter l'ensemble des interactions des interfaces usuelles qui correspondent à cinq types d'action générique : la sélection, le positionnement, l'orientation, la saisie de texte et le tracé à main levée. Ces cinq actions sont gérées par les six interacteurs suivants :

- Sélection : interacteur "Menu"
- Position : interacteurs "Move-Grow" et "New-Point"
- Orientation : interacteur "Angle"
- Saisie de texte : interacteur "Text"
- Tracé à main levée : interacteur "Trace"

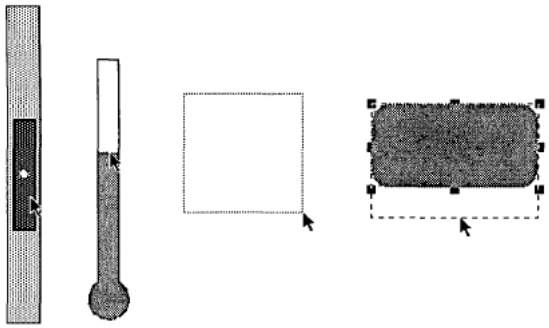
Chaque interacteur définit différents paramètres tels que les événements qui les font s'activer ou s'arrêter, et les éléments graphiques qu'ils peuvent manipuler. Cet aspect rend les interacteurs fortement paramétrables.

La figure 2.1 illustre comment les interacteurs "Menu" et "Move-Grow" peuvent être utilisés pour interagir avec des éléments de l'interface. La figure 2.1-a présente l'interacteur "Menu" qui permet la sélection d'un intitulé parmi une liste. Cette sélection fournit un retour visuel qui dépend de la nature de l'élément ciblé (et non de l'interacteur). Ce retour visuel peut être dans un premier temps un retour intermédiaire : rectangle en "vidéo-inverse" (1) et (5), contour rectangulaire (3), enfoncement d'un bouton (disparition de l'ombre portée) (4), mise en italique de l'intitulé lui-même (6). Le retour d'information final peut compléter la perception de la sélection : contour rectangulaire (1), déplacement de l'intitulé à droite ou à gauche (2), un signe plus (3), rectangle en "vidéo-inverse" (4), mise en gras de l'intitulé lui-même (6). La figure 2.1-b montre, quant à elle, l'interacteur "Move-Grow" qui peut agir sur la position (contrainte à une dimension) d'une barre de défilement ou d'une barre de thermomètre, ou bien sur la position ou la dimension d'un objet graphique.

L'interacteur est ainsi un élément interactif globalement *décorrélé des objets cibles*, ce qui est un point clé dans notre approche de l'espace de travail interactif. Dans un tel espace, nous souhaitons que l'utilisateur puisse disposer des outils d'interaction qu'il juge adaptés à son problème et à l'approche personnelle qu'il peut en faire. En conséquence, cet espace sera une zone dans laquelle les outils peuvent être installés et configurés au gré de l'utilisateur. Il est ainsi tout à fait nécessaire que les objets de cet espace soient décorrélés des outils qui permettent de les manipuler. Cet aspect de l'interaction est également présent dans la notion d'instrument d'interaction.



(a) Interacteur



(b) Interacteur

FIG. 2.1 – Interacteurs “Menu” et “Move-Grow” – extrait de (Myers, 1990)

2.1.4 L’interaction instrumentale

Les interacteurs permettent d’aborder l’interaction principalement d’un point de vue du *développeur*. Cette approche factorise élégamment les interactions basées sur le tandem clavier - souris en six interacteurs, lesquels ont tous un caractère *générique*. Cependant, ces six interactions sont déjà présentes dans les interfaces et la notion d’interacteur, et donc des bénéfices qu’elle apporte, est ainsi invisible pour l’utilisateur.

La motivation première de l’interaction instrumentale est de proposer un modèle d’interaction *centré sur l’utilisateur* (Beaudouin-Lafon, 1997; Beaudouin-Lafon, 2000), là où les boîtes à outils graphiques actuelles proposent des modèles exclusivement architecturaux. Ces modèles architecturaux sont principalement centrés sur le “widget”[6], élément qui se focalise sur son rendu visuel et sur le traitement des événements qu’il reçoit. Ces modèles facilitent la programmation des interfaces mais peu leur conception, ce qui reste vrai avec les interacteurs de Garnet. L’interaction instrumentale vise à fournir un référentiel pour les concepteurs d’interfaces en ce qui concerne l’interaction.

Un instrument d’interaction est défini comme étant le *médiateur* entre l’utilisateur et les objets qu’il souhaite manipuler. Le principe général de fonctionnement d’un instrument est le suivant :

[6]“Window Object” : un objet rectangulaire de type composant graphique interactif.

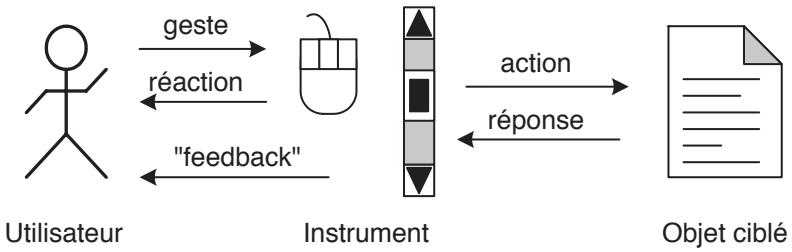


FIG. 2.2 – Principe général d'un instrument – inspiré de (Beaudouin-Lafon, 2000)

1. L'utilisateur agit sur l'instrument par le biais d'un ou plusieurs périphériques d'entrée utilisateur qui capturent les *gestes* de l'utilisateur.
2. Les gestes sont ensuite transformés en *actions*[7] en fonction de la nature de l'instrument. Ces actions sont appliquées à l'objet ciblé.
3. L'instrument réagit aux gestes de l'utilisateur et lui fournit donc une *réaction*.
4. Il fournit de plus un *retour d'information* (ou "feed-back") permettant à utilisateur de percevoir l'état de l'instrument et donc de l'action en cours.
5. Quand l'action est appliquée à l'objet ciblé, ce dernier renvoie une *réponse* quand l'action est traitée.

La figure 2.2 illustre ce principe pour l'instrument de type "barre de défilement" :

1. L'utilisateur effectue un "cliquer-déplacer" de l'ascenseur de la barre de défilement (rectangle noir). La souris capture alors les gestes de type "clic" et "déplacement".
2. Ces gestes sont transformés en une action de "défilement" du document dans sa fenêtre.
3. L'enfoncement du bouton de la souris, son relâchement et le déplacement de la souris fournissent un retour d'informations kinesthésiques à l'utilisateur.
4. L'utilisateur perçoit également la nouvelle position de l'ascenseur de la barre de défilement tout au long du geste.
5. Parallèlement, le document effectue le défilement escompté fournissant ainsi la réponse à l'instrument et, par voie de conséquence, à l'utilisateur.

L'interaction instrumentale fournit un modèle général s'appuyant sur la métaphore de l'outil. Cette approche va ainsi dans le sens de la manipulation directe puisque les outils du monde réel, tels que le crayon pour dessiner ou le stylo pour écrire, sont souvent directs.

Elle va également dans le sens des interfaces orientées objets, les instruments devenant les *objets* d'interaction : il est possible d'organiser les instruments en classes, d'établir une hiérarchie des instruments (héritage) et de les composer (composition des actions). De plus, les instruments encapsulent leurs états et leurs comportements, permettant ainsi de masquer leur complexité éventuelle.

Les instruments mettent en œuvre le principe de réification de par la transformation des gestes de l'utilisateur en actions qu'ils exécutent sur les objets ciblés. De plus, le principe de polymorphisme devient naturel avec les instruments puisqu'une même action peut être appliquée à plusieurs types d'objet. Un même instrument peut être utilisé dans des contextes

[7] Les termes de "geste" et d'"action" correspondent respectivement aux termes d'"action", et de "commande" de (Beaudouin-Lafon, 2000). Nous utiliserons les termes de "geste" et "action" dans notre modèle.

très variés. Certains instruments ont la caractéristique d'être *génériques* : ils peuvent s'utiliser dans de nombreux contextes. Par exemple, un instrument qui permet de déplacer des objets est un instrument générique puisque beaucoup d'objets de nature pourtant différente peuvent effectivement être déplacés.

L'interaction instrumentale est ainsi un modèle pertinent du point de vue de la manipulation directe, des interfaces orientées objets et des trois principes de Beaudouin-Lafon & Mackay (2000). Ce modèle reste cependant abstrait dans le sens où il n'a pas été testé à une échelle suffisante pour permettre son évaluation. L'application CPN2000, dédiée à l'édition de réseaux de Pétri colorés, intègre un ensemble de techniques d'interaction modernes (section 1.2) dans un environnement intégré. L'interaction est basée sur le modèle instrumental et en fournit une première implantation dans le contexte précis des documents de type "réseaux de Pétri". Le propos de cette thèse est, d'un certain point de vue, de proposer un modèle de l'interaction instrumentale plus général, adapté à tout type de document et permettant d'aborder la collaboration.

2.1.5 Le document en tant que support d'information

2.1.5.1 Document antique

(a) (b)

FIG. 2.3 – Supports antiques de l'écriture
(a) Papyrus mythologique 1185-950 av. J.-C. – extrait de http ://classes.bnf.fr/dossisup/supports/index12.htm
(b) Préparation médicale Argile, Ier mill. av. J.-C. – extrait de http ://classes.bnf.fr/dossisup/supports/index11.htm

La notion de document en tant que *support* pour l'écriture – et corollairement pour la communication – remonte à l'antiquité (BNF, 2003). Le support le plus représentatif des écritures antiques est sans aucun doute le papyrus (figure 2.3-a). Plante répandue essentiellement dans le delta du Nil, le papyrus a reçu toute sorte d'écritures tels que le grec, le démotique[8], le latin ou l'arabe. La structure de la feuille de papyrus imposait la forme du rouleau. Le papyrus est remarquable de par sa persistance dans l'histoire : les plus anciens

[8]Écriture égyptienne et forme simplifiée des hiéroglyphes utilisée pour les besoins courants.

manuscrits retrouvés datent du IIIe millénaire et les plus récents du XIe siècle après J.-C. . À partir du IIe siècle, le codex de parchemin remplacera peu à peu le rouleau de papyrus.

Les tablettes gravées, antérieures au papyrus, ont également constitué un support remarquable de l'écriture. S'il est difficile d'affirmer que l'écriture naquit de l'argile, les "écrits" les plus anciens à ce jour ont été retrouvés sur des tablettes d'argile de la fin du IVe millénaire (figure 2.3-b). En Basse Mésopotamie où l'argile abondait, les Sumériens utilisaient largement les tablettes d'argile gravées en écriture cunéiforme et relatant leur vie sociale, religieuse, culturelle et scientifique. Fait très marquant, ces tablettes servaient de cahiers d'exercices aux élèves : tant que l'argile était humide, les calculs pouvaient être effacés et refaits. Ceci préfigure la notion de document en tant qu'objet interactif.

De nos jours, la notion de document n'est pas nécessairement relative à la notion de papier : un document peut être numérique. Le document peut ainsi être de nature très variée car il peut s'appuyer sur différents médias. La métaphore du papier reste toujours présente et permet la composition de documents structurés, typiquement autour du texte et du graphique. Elle est de plus complétée par la nature dynamique de l'information : les documents peuvent être animés, sonores, audiovisuels, ou bien écrits en braille par exemple.

2.1.5.2 Document informatique composite

La grande majorité des tâches que réalisent les utilisateurs dans les environnements graphiques actuels est centrée sur le concept de document. Ces tâches consistent généralement à construire un document à l'aide d'éditeurs dédiés (applications qualifiées de "bureautiques" ou de suites logicielles telles que "Microsoft Office") ou à le visualiser et éventuellement l'annoter à l'aide d'une application de type "visionneuse" (tel qu'un navigateur Web pour visionner des documents au format HTML[9] ou Acrobat Reader pour visualiser des documents au format PDF[10]). L'utilisateur interagit également avec l'environnement graphique dans le but de (re)trouver des documents. Cette tâche de recherche et de navigation s'effectue dans un espace documentaire très vaste puisqu'il inclut les documents locaux à la machine de l'utilisateur, ainsi que les documents accessibles *via* des réseaux locaux ou l'Internet.

Le document est également très présent dans les formes diverses de la collaboration interpersonnelle. La collaboration dans la vie quotidienne est basée sur la co-présence : les personnes communiquent au même endroit *et* au même moment. Cette co-présence s'enrichit souvent d'artefacts proches de l'idée de document en tant que support pour la collaboration. Par exemple, un enseignant peut utiliser un polycopié en tant que support de cours, animer le cours en appuyant son discours à l'aide de transparents et d'inscriptions sur le tableau blanc, tout comme il peut utiliser la projection audiovisuelle pour alimenter le débat. Un autre exemple concerne la discussion d'un groupe d'architectes autour du plan d'un édifice : chaque architecte peut annoter le plan et éventuellement le modifier en concertation avec les autres membres du groupe. Dans ces deux exemples, les documents (pris au sens large) sont des objets pivot pour la collaboration : leur construction ne constitue pas nécessairement l'objectif de la collaboration mais ils représentent le vecteur autour duquel la communication s'anime.

Lorsque les participants sont géographiquement distants, l'utilisation d'outils de communication (informatiques ou non) reste toujours centrée sur la notion de document. Typiquement, la collaboration consiste en un échange ou en un partage de documents, l'échange de documents participant d'un mode de coopération asynchrone et le partage de documents d'un mode de coopération synchrone.

[9] HyperText Markup Language
[10] Portable Document Format

Nous avons déjà montré dans la section 1.1.1 que le concept de document était au centre de l'ancêtre de nos systèmes actuels, le Xerox Star. Si l'idée de document reste encore largement présente dans les interfaces graphiques, force est de constater que ces dernières n'ont guère évolué et confortent les applications dans leur position dominante. Du point de vue de l'utilisateur comme du point de vue du concepteur, il serait préférable que l'application devienne *transparente*. L'approche proposée par OpenDoc est probablement le système le plus abouti pour faire disparaître le concept d'application (section 1.3.2.4) : l'utilisateur manipule des (parties de) documents au lieu de manipuler des applications spécifiques. Les substituts des applications sont les éditeurs et les visionneuses de parties qui sont d'une granularité plus faible. Cette diminution de la granularité rend ces substituts moins visibles. Le fait que les éditeurs et les visionneuses soient "à l'intérieur" des documents (alors que dans les systèmes actuels, les documents sont "à l'intérieur" des applications) participe également à cette sensation d'invisibilité.

Cependant, si l'application est moins visible dans l'approche OpenDoc, il subsiste un inconvénient déjà présent dans l'approche centrée sur les applications : le choix de l'éditeur adapté à la tâche d'édition à réaliser. Ce choix est toujours délicat car il implique, une fois effectué, de rester contraint à l'intérieur de l'éditeur élu. Il peut en particulier arriver des situations où deux éditeurs proposent des capacités d'édition spécifiques et non communes mais qui sont jugées pertinentes par l'utilisateur quant à la tâche qui lui incombe. Nous tenterons dans notre modèle de réparer cette lacune en ne contraignant pas l'utilisateur à l'intérieur de parties et donc aux fonctionnalités proposées par l'éditeur. Ceci sera possible en considérant une granularité plus fine que celle de OpenDoc.

2.1.5.3 Format unifié pour les documents

Du point de vue de la représentation des données, le formalisme XML[11] est devenu un (le ?) standard reconnu pour la définition des formats de documents (W3C, 2000). Il formalise (voire normalise) à la fois la structuration des données d'un document (utilisation des balises à l'intérieur de données textuelles) ainsi que la grammaire des documents. Autant de grammaires peuvent être imaginées afin de définir des types variés de documents. Dans la pratique, certaines grammaires deviennent des standards de fait, comme SVG[12] pour la description de graphiques vectoriels 2D ou MathML pour la description d'équations mathématiques. Le formalisme XML est en particulier adapté à la représentation de documents composites puisqu'il définit la notion d'espace de nommage. Un espace de nommage définit une portion d'un document XML qui utilise une grammaire donnée, un document pouvant utiliser autant d'espaces de nommage (et donc de grammaires) qu'il est jugé nécessaire.

A notre connaissance, aucune approche ne se propose de définir un modèle d'interaction sur les documents XML. Le modèle que nous proposons dans cette thèse tente d'initier une telle approche. Il se veut à cet égard le plus générique possible, au même titre que le formalisme XML l'est pour la représentation des données des documents.

2.1.6 Composants interactifs

Lors de l'analyse de la tendance des systèmes actuels (section 1.3.1), nous avons illustré les différentes approches qui permettent d'augmenter la fluidité de l'interaction quand l'utilisateur bascule d'une application à l'autre. Nous avons en particulier mentionné l'approche par extensions (ou "plug-ins") qui consiste à agrémenter les applications de nouvelles extensions développées par des tierces parties. Dans l'application Adobe Photoshop, il devient

[11] eXtended Markup Language
[12] "Scallable Vector Graphics"

ainsi possible de créer des graphiques vectoriels bien que l'application soit pourtant dédiée à la retouche d'images photographiques. Du point de vue de l'utilisateur, cette approche est pertinente puisqu'elle permet d'*adapter* les applications disponibles à ses propres besoins. La popularité de plug-ins illustre bien l'intérêt grandissant de la modularité et de l'extensibilité tant du point de vue de l'utilisateur que du point de vue des éditeurs de logiciels, dans un contexte pourtant toujours basé sur les applications. Cette approche par extensions rejoint celle plus générale et ambitieuse de la construction de logiciels à partir de *composants* prêts à l'emploi.

2.1.6.1 Services

Les technologies liées au composant logiciel sont apparues il y un peu moins de dix ans. Nous pouvons citer par exemple l'architecture COM[13] de Microsoft (1995) et les JavaBeans de Sun (1997). Si l'approche de Microsoft est relativement technique et complexe, celle proposée par Sun reste simple. Ce dernier point est illustré par le fait que l'ensemble des "widgets" proposés par Java/Swing sont des composants JavaBeans à part entière (Eckstein et al., 1998) ; les classes Swing n'en sont pas pour autant plus complexes à utiliser.

La facette interactive (*i.e.* sans prendre en considération les aspects collaboratifs) est réalisable à condition de définir certains *services* spécifiques aux composants. Notre implantation est assez proche de celle des JavaBeans dont nous donnons ci-dessous les services caractéristiques (Szyperski, 2002) :

Propriétés Les propriétés d'un composant permettent de paramétrer l'apparence et le comportement du composant. Les propriétés sont des grandeurs accessibles par des "accesseurs", *i.e.* des méthodes préfixées de "get" pour l'accès en lecture, ou de "set" pour l'accès en écriture.

Méthodes Les méthodes publiques peuvent être invoquées par n'importe quel autre composant. Ceci permet la "scriptabilité" des composants, *i.e.* la possibilité de dialoguer avec les composants selon un langage de script adapté.

Événements Les JavaBeans sont notifiés par l'envoi d'événements. Un mécanisme proche du motif de conception observable / observateur et basé sur les écouteurs (ou "listeners") est mis en œuvre à cette fin.

Introspection Un composant fournit un mécanisme introspectif qui lui permet de rendre publiques ses différentes caractéristiques : nom de la classe et hiérarchie d'héritage, description des méthodes publiques, description des propriétés publiques, et description des événements qui peuvent être consommés ou produits.

Personnalisation Les composants doivent pouvoir être personnalisables en fonctions des besoins de l'utilisateur ou/et du concepteur de l'application. Ceci est possible en utilisant les propriétés et les méthodes précédemment définies.

Persistance Les composants doivent pouvoir être enregistrés et ultérieurement restaurés.

2.1.6.2 Espace de composants

Dans le cadre de cette thèse, nous visons à amener le concept de composant au niveau de l'espace de travail lui-même, les applications disparaissant au profit de ces composants. L'utilisateur n'aura ainsi pas à se soucier de savoir quelles applications il pourrait utiliser pour réaliser une tâche donnée : il choisira parmi un ensemble d'instruments d'interaction, lesquels sont des composants au même titre que les documents. Notons que, à notre connaissance, il n'existe pas de tels environnements où le concept même d'application

[13] "Component Object Model"

disparaît au profit des composants. Nous visons ainsi, au delà de cette thèse, à fournir une plate-forme expérimentale qui permettrait de tester et de valider (ou d'invalider !) la pertinence d'une approche exclusivement basée sur les composants au niveau de l'espace de travail informatique. Nous parlons de *composants interactifs* pour désigner les composants logiciels présents dans l'espace interactif, lesquels seront définis conformément à notre modèle centré sur des métaphores spécifiques (les documents et les instruments par exemple). Ces composants logiciels n'ont ainsi pas pour vocation de pouvoir coopérer avec d'autres types de composants (non interopérabilité). C'est pour cette raison que nous n'utiliserons pas des technologies existantes, telles que les JavaBeans, puisque ces dernières résolvent un problème très général. En faisant de la sorte, nous nous affranchissons de plus des contraintes imposées par l'utilisation d'une technologie particulière. Le coût à payer sera alors de fournir davantage d'effort pour la construction de notre boîte à outils.

Nous compléterons la notion de composant logiciel en abordant les fondements des composants distribués dans la section 2.2.4.

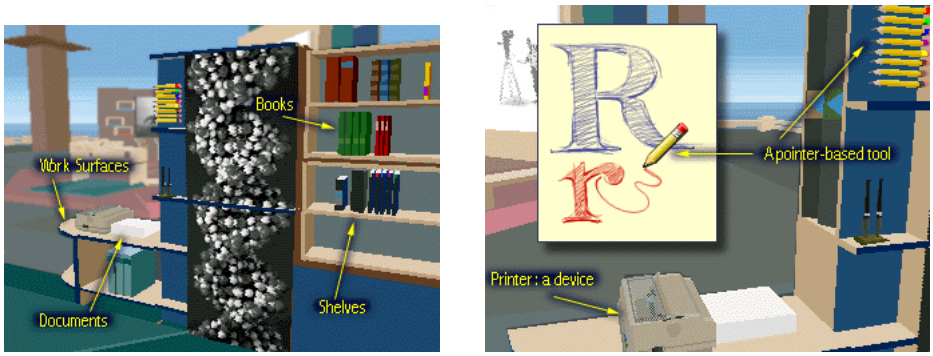
2.2 Fondements pour la facette collaborative

2.2.1 Métaphores

L'utilisation de métaphores dans un environnement interactif offre l'avantage de permettre aux utilisateurs de transférer leurs connaissances du monde réel vers l'environnement informatisé. Les métaphores les plus connues dans les systèmes informatiques actuels sont le bureau, les dossiers et les documents, les boutons de commande et les menus. En utilisant ces métaphores, l'utilisateur sait naturellement qu'il peut organiser son bureau comme il l'entend (positionnement des objets sur l'espace de travail), qu'il peut ranger ses documents dans différents dossiers qu'il nomme à sa guise, qu'il peut manipuler ses documents (lecture, annotation et édition) avec les applications adéquates, ou bien qu'il peut appuyer sur un bouton ou sélectionner un élément dans un menu pour activer une commande. Si l'apport des métaphores est indéniable, il ne faut cependant pas en abuser. Deux raisons à cela peuvent être avancées :

1. Les métaphores sont des emprunts d'entités du "monde réel", ces dernières étant bien adaptées à ce monde mais pas nécessairement à un "monde virtuel" (celui affiché par l'ordinateur).
2. Leur utilisation intensive peut limiter la découverte et la conception de nouveaux artefacts qui n'auraient pas d'équivalent dans le "monde réel".

L'exemple des "RealPlaces" de Roberts (2000) illustre le point 1. Les "RealPlaces" sont des espaces "physiques" représentés en trois dimensions et dans lequel l'utilisateur peut réaliser ses différentes tâches. Cette métaphore est ainsi celle du bureau mais au sens du lieu et non du meuble. Dans ces "places réelles", les objets sont des objets empruntés au monde réel, à savoir des bureaux, des étagères, des armoires, organisés en différentes pièces (figure 2.4-a). L'interface est en trois dimensions pour le contexte d'organisation général et les objets manipulés apparaissent dans une "projection" en deux dimensions dans cet espace 3D (figure 2.4-b). Si cet exemple de "RealPlaces" semble attractif du fait que nous y retrouvons des éléments très habituels de nos lieux de travail, nous pouvons douter de la pertinence de l'utilisation de "meubles" de rangement pour les différents éléments que sont les documents, les dossiers ou les outils. De tels meubles sont bien adaptés à l'organisation d'objets physiques parfois encombrants (un dossier volumineux ou une imprimante). Mais le restent-ils pour des objets informatiques qui eux ne sont pas "physiquement volumineux" ?



(a) (b)

FIG. 2.4 – Exemple de conception basées sur “RealPlaces” – extrait de http ://www-3.ibm.com/ibm/easy/eou_ext.nsf/Publish/580

Afin d’illustrer le point 2, nous pouvons prendre l’exemple de la navigation par “pan & zoom” des interfaces “zoomables” (section 1.2.2.1). Une telle façon de se déplacer dans un espace n’est pas directement empruntée à notre manière de nous déplacer physiquement dans nos bureaux puisqu’elle ne concerne que trois degrés de liberté (deux degrés pour le “pan” et un degré pour le “zoom”). Cependant, l’utilisation d’un tel mode de déplacement s’avère très efficace pour les interfaces graphiques (Guiard et al., 2001). Il permet de concevoir une nouvelle approche de la navigation dans les espaces de travail conjointement à de nouveaux artefacts tels que les “portails” de Pad (qui permettent aisément de passer d’un point à l’autre de l’espace).

Ces deux exemples illustrent l’utilisation délicate des métaphores. Dans l’approche basée sur la réalité virtuelle mentionnée dans la section 1.4.4.4, nous avons appuyé le fait que vouloir imiter le monde réel dans un système qui reste virtuel n’apporte pas les avantages escomptés. Ceci rejoint le constat que nous venons de faire sur l’utilisation (excessive) des métaphores.

Cet aspect général des métaphores (et de la réalité virtuelle) reste encore tout à fait pertinent dans le cadre de la collaboration. Hollan & Stornetta (1992) montrent en effet que les collecticiels ne doivent pas exclusivement viser à rendre possible la communication en face-à-face lorsque les personnes sont distantes. Cette possibilité est difficile (voire impossible) à atteindre puisque la communication en face-à-face possède des caractéristiques très informelles qu’il est difficile de rendre compte *via* des systèmes informatiques. Les auteurs argumentent alors qu’il est préférable de proposer des systèmes qui permettent “d’aller au-delà du fait d’être présent”[14], c’est à dire apporter des possibilités de communication qui n’existent pas dans la réalité. Leur proposition la plus significative est la suivante :

> “Si nous espérons résoudre le problème de la télécommunication, nous devons développer des outils que les personnes préféreront utiliser même lorsqu’ils ont par ailleurs la possibilité d’interagir à proximité réelle.” (Hollan & Stornetta, 1992, page 125)

Un exemple actuel va dans le sens de cette citation : le courrier électronique est largement utilisé de nos jours et est souvent préféré à la rencontre en face-à-face ou la rencontre téléphonique puisqu’il permet d’interagir avec une ou plusieurs personnes sans redouter de la déranger (communication asynchrone). Les systèmes de visioconférence peuvent proba-

[14] Traduction de “beyond being there”.

blement être considérés comme moins populaires de par le manque d'outils intégrés qui permettraient d'enrichir la collaboration.

Dans notre approche, le modèle que nous proposons sera basé sur quelques métaphores élémentaires tels que les documents et les instruments. Concernant la facette collaborative du modèle, nous portons notre réflexion sur la métaphore du bureau que nous étendrons aux notions d'espace de travail partagé et de place.

2.2.1.1 Espace partagé

La notion de collaboration basée sur la co-présence suggère naturellement la métaphore de la "pièce" initialement développée dans le cadre d'un espace mono-utilisateur (Henderson & Card, 1986), puis dans le cadre plus général des collecticiels (Greenberg & Roseman, 2002). Elle permet à la fois d'aborder les facettes synchrones et asynchrones de la collaboration en un unique objet. Il est ainsi naturel que cette métaphore soit présente dans notre modèle conceptuel : l'un des éléments de l'environnement interactif permettant le partage (ou l'échange) d'objets est l'espace de travail lui-même, considéré en tant que pièce accessible ou non par plusieurs utilisateurs (espace public ou espace privé).

La notion de partage des espaces de travail permettra d'aborder la façon dont les éléments tels que les documents, leurs présentations et les instruments pourront y être partagés et échangés.

2.2.1.2 Place

Si la capacité de partager des espaces de travail est pertinente et nécessaire pour la collaboration, elle peut s'avérer ne pas être suffisante. Il existe en effet des situations où la collaboration peut avoir lieu en dehors d'un espace physique partagé. Ceci est vrai pour la communication téléphonique ainsi que pour les listes de diffusion ou les forums de discussion : ces lieux de collaboration vivent sans le support d'un espace physique dédié à la collaboration, mais uniquement par le biais d'un canal de communication.

L'idée générale de "place " a été mise en avant par Harrison & Dourish (1996). Les auteurs soulignent la différence essentielle entre place et espace :

> "L'espace est l'opportunité ; la place est la réalité comprise"[15] (Harrison & Dourish, 1996, page 69)

Une place peut exister au travers d'un espace physique, ou bien ne dépendre d'aucun espace physique. La place joue ainsi un rôle *social* dans la collaboration : elle définit sa raison d'être ainsi que ses usages.

Nous soulignons ici cet aspect qui nous semble essentiel et pousse un peu plus loin la métaphore de la pièce en lui préférant celle de la place. Cette dernière est en effet plus générale car elle englobe la notion d'espace partagé, la notion de collaboration sans espace partagé, ainsi que l'aspect social. Au niveau de notre modèle conceptuel, cela signifie que nous devons prendre en considération les deux possibilités de collaboration basées sur le concept de place : la collaboration *via* un espace partagé (l'espace de travail) et la collaboration sans espace de partage.

[15] "Space is the opportunity ; place is the understood reality."

2.2.2 Aspect temporel dans la collaboration

La matrice des collecticiels (tableau 1.2 on page 30) nous montre que la collaboration synchrone et la collaboration asynchrone ont des domaines d'application assez distincts. Chacun de ces deux types d'application définit ses propres fondements qu'il est important d'avoir à l'esprit quant à l'élaboration de notre modèle. Nous tentons de synthétiser ici leurs différents aspects.

2.2.2.1 Collaboration synchrone

Nous avons vu que la collaboration synchrone concerne aussi bien la co-présence que la collaboration à distance. Dans le cas de personnes co-présentes, l'environnement interactif peut être un vecteur complémentaire à la collaboration dans un espace physique réellement commun (partagé). Cet environnement est alors typiquement constitué d'un écran commun aux différents utilisateurs[16] et nous parlons donc de "single display groupware" (SDG). Dans le cas de personnes distantes, l'espace physique de chacune d'elle n'est plus commun et le rôle premier des collecticiels est de réduire cette distance sans toutefois vouloir reproduire *stricto-sensu*. Nous parlons dans ce deuxième cas de collecticiel temps réel (ou "real-time groupware").

Les aspects pris en compte par les collecticiels synchrones sont :

1. la concurrence d'accès,
2. la cohérence des données,
3. le couplage des interfaces,
4. et la conscience mutuelle.

La *concurrence d'accès* précise comment détecter et gérer les conflits qui ont lieu lorsque plusieurs utilisateurs accèdent simultanément à un objet partagé en vue de le manipuler et donc de modifier son état interne. Ce point 1 concerne aussi bien le "single display groupware" que les collecticiels temps réel. Dans le premier cas, les objets de l'écran partagé sont *implicitement* les objets partagés entre les différents utilisateurs alors que, dans le deuxième cas, les objets partagés sont définis *explicitement* comme étant partagés (en utilisant la métaphore de la "pièce commune" par exemple). Dans les deux cas, les utilisateurs peuvent accéder simultanément aux mêmes objets partagés. Les trois autres points (2, 3 et 4) ne concernent par contre que des utilisateurs distants.

La *cohérence* des données partagées doit être assurée par le système collaboratif. Une incohérence peut survenir lorsque des utilisateurs *distants* manipulent un même objet à des instants relativement proches : leurs actions respectives peuvent ne pas être appliquées dans le même ordre sur les différents postes à cause des temps de latence non déterministes imposés par le transit des informations sur le réseau. Un exemple classique concerne l'édition partagée de texte. Soit un utilisateur A qui complète le mot partagé "hès" en insérant le caractère "e" en position 4 alors qu'un utilisateur B complète ce même mot en insérant le caractère "T" en position 1. Si ces deux actions ne sont pas appliquées dans le bonne ordre (insérer "e" en position 4 *puis* "T" en position 1), le résultat devient "Thèes" au lieu de "Thèse" et demeure alors inconsistant. Notons que la gestion de la cohérence est liée au choix du mécanisme de gestion de la concurrence d'accès, comme nous l'illustrerons dans le chapitre 6.

Le *couplage* des interfaces précise à quel niveau les différentes actions de l'utilisateur sur les objets de l'interface sont partagés. Par exemple, lors de l'édition synchrone d'un document, il peut être choisi de ne partager que le contenu du document sans partager la fenêtre

[16] Les périphériques d'entrée ne sont cependant pas communs.

qui le contient. Ainsi, les actions de défilement du document dans la fenêtre ne sont pas partagées puisque la fenêtre elle-même n'est pas partagée. Il peut être choisi de travailler avec un couplage plus fort en partageant cette fois-ci la fenêtre : les actions de défilement sont maintenant partagées. Ces deux exemples illustrent respectivement les deux termes de "WYSIWIS[17] strict" et de "WYSIWIS relâché" définis par Stefik et al. (1987). Le fait de pouvoir ajuster le degré de couplage entre des interfaces collaboratives est un point important et qui reste intrinsèquement impossible en SDG puisque l'écran est physiquement commun. Par conséquent, le réglage du degré de couplage est un plus en faveur de la collaboration distribuée par rapport à la co-présence, allant ainsi dans le sens de "beyond being there" de Hollan & Stornetta (1992).

Enfin, l'un des points les plus crucial et complexe à réaliser par les collecticiels concerne la *conscience mutuelle* des actions des utilisateurs distants (ou "awareness") . Ce point est important car cette conscience mutuelle disparaît naturellement dès que les utilisateurs sont distants alors qu'elle joue un rôle central dans la perception de "l'état" de la collaboration : si cette perception disparaît, la collaboration devient presque impossible. Nous aborderons dans la section 2.2.3 les différentes techniques proposées pour permettre de fournir un certain niveau de conscience mutuelle. Notons que ces techniques, bien qu'elles relèvent de la *conception* des espaces des travail, imposent des contraintes fondamentales qu'il est nécessaire d'identifier afin de les prendre en compte au niveau de notre modèle.

2.2.2.2 Collaboration asynchrone

Si l'idée de collaboration synchrone à distance est calquée sur la co-présence et le dialogue oral inhérent à la collaboration en face-à-face, la collaboration asynchrone s'inspire d'un dialogue basé sur le message posté et le "dialogue écrit". Nous avons illustré dans la section 1.4.3 les collecticiels asynchrones par quelques applications typiques appartenant à deux catégories, les éditeurs permettant d'annoter des documents d'une part et les systèmes de type "workflow" d'autre part. Ces différentes applications traitent toutes de points particuliers et d'une manière qui leur est propre. Ainsi et à notre connaissance, peu de travaux de recherche et de développement ont conduit à une réflexion d'ordre général sur les collecticiels asynchrones, ce qui se traduit par une absence de boîtes à outils dédiées à leur développement. Quand nous évoquons l'idée de boite à outils pour les collecticiels, nous pensons plus souvent à la facette temps réel qu'à la facette asynchrone. Ceci provient probablement du fait que les problèmes techniques liés au temps réel sont à juste titre considérés implicitement comme complexes, tandis que la collaboration asynchrone ne pose pas réellement de problèmes techniques. Cependant, force est de constater que la collaboration asynchrone pose des problèmes de *conception* non triviaux. Le problème le plus sensible concerne la notification des changements d'état des objets partagés : les utilisateurs doivent être avertis du fait que l'état de l'objet a changé sans pour autant être submergés de messages de notification. Il s'agit donc de trouver un compromis délicat entre une abondance des messages de notification et la possibilité pour les utilisateurs d'avoir une conscience précise de l'évolution des différents objets.

Preguiça et al. (2000) fournit une réflexion d'ordre assez général sur les aspects essentiels des collecticiels asynchrones en proposant une boîte à outils orientée objet. Les aspects fondamentaux des collecticiels asynchrones y sont identifiés :

1. Disponibilité élevée des données.
2. Concurrence d'accès aux données.
3. Intégration d'un support pour la conscience mutuelle.

[17] "What You See Is What I See"

Le point 1 traite de la possibilité pour l'utilisateur d'accéder aux données où qu'il soit. Ceci signifie qu'il doit être possible d'accéder à des documents partagés sans pour autant être physiquement connecté à un réseau. Ce cas est motivé par deux raisons distinctes : le réseau est inaccessible pour des raisons purement techniques, ou bien l'utilisateur ne souhaite pas être connecté à un réseau de manière à travailler en toute intimité. Cette possibilité est naturellement intéressante pour l'utilisateur et va dans le sens de la citation "beyond being there". Dans le système DOORS (Preguiça et al., 2000), les différents objets partagés sont répliqués sur l'ensemble des postes concernés. Les différentes répliques sont maintenues à jour de manière consistante par un algorithmique qualifié d'*épidémique* : les différentes modifications sont propagées sur les différents postes à la manière d'une épidémie, c'est à dire dès qu'un poste se connecte au réseau (il peut alors mettre à jour l'état des répliques modifiées sur les autres postes et propager ces propres modifications).

Le problème des accès concurrents n'est pas spécifique aux collecticiels temps réel mais se pose également pour les systèmes asynchrones. En effet, supposons que deux utilisateurs modifient chacun de leur côté la même portion de texte d'un document partagé. Le premier utilisateur valide les modifications qu'il a apportées au texte de manière à les propager sur le document d'origine. Lorsque le second utilisateur valide à son tour ses propres modifications, il y a conflit puisque les modifications concernent le document dans sa version d'origine et non pas sur la version mise à jour par le premier utilisateur. La résolution du conflit consiste à *choisir* parmi les différentes modifications apportées par les *deux* utilisateurs celles qui devront être effectivement appliquées sur le document d'origine. La gestion de la concurrence doit permettre de détecter le conflit *et* de le traiter.

Enfin, la prise de conscience mutuelle des actions des utilisateurs sur les objets partagés demeure un point tout aussi essentiel en asynchrone qu'en synchrone. Bien que la manière de restituer une telle prise de conscience diffère dans les deux cas, elle n'en demeure pas moins aussi complexe à concevoir. Comme le souligne Preguiça et al. (2000), les informations de restitution dépendent de la nature des applications (*i.e.* des tâches) à réaliser. Par exemple, une application d'aide à la planification va baser la conscience mutuelle sur la notification des intéressés à activer dès lors que le planning a effectivement changé, alors qu'une application d'édition de documents va plutôt utiliser un mécanisme de visualisation de l'historique des actions effectuées sur le document par des tierces personnes. Ces deux exemples sont significatifs car ils illustrent les deux techniques fondamentales de restitution de la conscience mutuelle en asynchrone, à savoir la *notification* et la *gestion de l'historique* (qui inclut notamment la perception des différences).

2.2.2.3 Synchrone *versus* asynchrone

L'usage veut souvent que l'on considère les aspects synchrones séparément des aspects asynchrones. Cependant, il n'y a pas nécessairement de frontière nette entre les deux colonnes "au même instant" et "en temps différé" de la matrice de collecticiels (tableau 1.2 on page 30). Pour s'en convaincre, donnons un petit exemple. Soit une application d'édition collaborative "temps réel" de documents textuels dans laquelle les différents utilisateurs perçoivent la saisie par chaque utilisateur du texte dans le document. Afin de ne pas trop perturber les utilisateurs par des modifications du document très fréquente, l'application prend en compte uniquement les ajouts de caractères qu'une fois chaque phrase terminée. L'application peut bien être qualifiée de synchrone car elle permet l'édition d'un même document au même instant et le filtrage des actions par phrase n'est finalement qu'une technique liée à la restitution de la conscience mutuelle. Cependant, il est clair que l'application fonctionne d'un point de vue asynchrone puisqu'elle attend chaque fin de nouvelle phrase pour mettre à jour les différentes interfaces des utilisateurs distants. Ainsi, nous devrions plutôt parler de granularité de synchronisme, une granularité élevée (fort synchronisme) signifiant que les objets partagés sont mis à jour "le plus souvent possible", et une

granularité basse (faible synchronisme) signifiant que les objets partagés sont mis à jour "de temps à autre". La différence entre synchrone et asynchrone tient principalement au fait que dans le premier cas, le système décide quand il faut mettre à jour l'objet partagé alors que, dans le second cas, l'utilisateur décide lui-même de cet instant. Cependant, cette considération concerne la facette *conception* de l'application et sort ainsi du cadre de notre modèle.

L'ajustement du "grain" de synchronisation est un point intéressant pour l'utilisateur. Par exemple, un utilisateur peut vouloir travailler de manière très couplée avec d'autres utilisateurs en partageant un espace de travail contenant une vue d'un document donné, ou bien en ne partageant que le contenu de ce document dans une fenêtre personnelle, ou bien encore en travaillant sur une version du document et en propageant ultérieurement les modifications apportées au document d'origine. Notons de plus que les techniques de restitution de la conscience mutuelle propres à la collaboration asynchrone peuvent être utilisées dans un contexte synchrone. Par exemple, un utilisateur qui édite un document partagé en temps réel n'a pas nécessairement intégré l'ensemble des modifications réalisées par les autres utilisateurs ; il peut donc souhaiter utiliser l'historique de manière à combler cette méconnaissance. En conséquence, les deux styles de collaborations (synchrone et asynchrone) ne doivent pas être considérés comme des modes, ce qui impliquerait d'un mode à l'autre dans les exemples pré-cités. Les aspects liés à la communication synchrone et à la communication asynchrone doivent être traités à un *même* niveau, c'est à dire dans un unique modèle fédérateur.

	Synchrone		Asynchrone	
	locale	distante	locale	distante
1. Disponibilité des données	−		−	×
2. Concurrence d'accès	×	×	−	×
3. Cohérence des données	−	×	−	×
4. Couplage des interfaces	−	×	−	
5. Conscience mutuelle	−	×	×	

TAB. 2.1 – Synchrone *versus* asynchrone

Dans le tableau 2.1, nous avons regroupé l'ensemble des fonctions essentielles des collecticiels et répertorié ces fonctions selon les quatre styles de collaboration : collaboration synchrone ou asynchrone, entre des utilisateurs co-présents ou distants. Nous observons que les fonctions sont soit à gérer par le système (symbole ×), soit implicitement présentes de par le style de collaboration (symbole −) :

1. Les données sont implicitement disponibles en collaboration synchrone et en collaboration asynchrone locale. Par contre, la disponibilité des données est un point important de la collaboration asynchrone et distante puisque les utilisateurs doivent pouvoir travailler de manière déconnectée du réseau.
2. La gestion de la concurrence d'accès est à prendre en compte dans tous les cas, sauf en collaboration asynchrone sur poste unique puisqu'il n'y a alors aucun cas d'accès concurrent possible.
3. La gestion de la cohérence des données doit intervenir dès que la collaboration devient distante. La cohérence est implicitement correcte lorsque la collaboration a lieu sur un unique système local.
4. Les interfaces peuvent avoir un degré de couplage variable dans le cas de la collaboration synchrone distribuée. Le couplage est par contre implicitement maximal en SDG puisque l'écran est partagé, et il est implicitement nul en asynchrone.
5. Enfin, la conscience mutuelle est implicite dans le cas de la co-présence et doit être restituée dans les autres cas.

Ce tableau de synthèse illustre le fait que les cinq points fondamentaux identifiés concernent aussi bien le côté asynchrone que le côté synchrone. Ceci devrait donc simplifier leur prise en compte respective dans un modèle unificateur et, par conséquent, dans une boîte à outils permettant de développer les deux facettes souvent considérées séparément. Notons que, dans le cadre de cette thèse, il ne sera pas envisagé d'implanter pleinement ces cinq aspects dans la boîte à outils, étant donné le travail que cela représente. Ils seront cependant pris en considération dans notre modèle conceptuel "théorique".

2.2.3 Action et collaboration

2.2.3.1 Interaction

Le terme d'interaction évoque naturellement, dans le contexte de l'Interaction Homme-Machine, les différentes actions de l'utilisateur vers les objets d'intérêt et les réponses fournies en retour par ces objets. L'homme dialogue ainsi avec les objets de l'interface. Dans le contexte de la collaboration utilisant les machines, l'interaction évoque de plus le dialogue entre les hommes et nous pouvons parler d'"Interaction Homme-Homme".

L'interaction entre les utilisateurs peut être de nature plus ou moins formelle. Par exemple, l'édition collaborative d'un document est une opération à caractère plutôt formel. Dans ce cas, un panel d'outils est fourni de manière à rendre possible la manipulation simultanée ou non d'un document partagé au travers d'un éditeur spécialisé. Un exemple simple de communication informelle est la rencontre et la discussion pendant la "pause café". L'informatique peut là encore tenter de fournir un support à ce type de collaboration. Les applications de type espaces médiatisés abordés à la section 1.4.4.2 on page 42 sont très significatives à cet égard.

Notre modèle proposera des principes de bases quant au côté interaction formelle par le biais des outils interactifs, les instruments d'interaction, ainsi que par le biais des cibles de l'interaction, les documents. Le côté informel de l'interaction concerne davantage la conception de l'espace de travail par la mise en œuvre de techniques liées aux "mediaspaces".

2.2.3.2 "Outeraction"

Nardi et al. (2000) soulignent le fait que le concept de "instant messaging" ne concerne pas à proprement parler l'interaction Homme-Homme. L'idée centrale de l'"instant messaging" part de l'observation que des utilisateurs peuvent potentiellement trouver un intérêt à interagir lorsqu'ils se rencontrent d'une manière inopinée. Quand plusieurs utilisateurs qui se connaissent préalablement sont en train d'écrire simultanément du courrier électronique au même moment, le système d'"instant messaging" les avertit de ce fait. Puisque ces utilisateurs sont effectivement en train d'utiliser leur machine à ce moment précis, il leur devient possible de se "rencontrer" tout comme s'ils se croisaient dans la rue. Ils peuvent alors entamer une discussion informelle par le biais d'un programme de "chat" par exemple. Nardi et al. (2000) définissent le terme de "outeraction" pour caractériser cette forme d'action non planifiée.

L'auteur définit l'interaction entre acteurs comme une forme d'échange d'information (formelle ou informelle), et l'outeraction comme une forme d'action qui ne procède *pas* d'un échange d'information. Nous avons ici mentionné ce point dans la section sur les fondements de la collaboration pour notre modèle car nous pensons que l'outeraction joue un rôle important (et attractif) dans la collaboration. Nous aborderons ce point dans le modèle par le biais du concept de place commune préalablement mentionnée dans la section 2.2.1.2.

2.2.3.3 Conscience mutuelle

Nous avons introduit l'idée de restitution de la conscience mutuelle des actions des différents collaborateurs distants dans la section 2.2.2. Nous avons insisté sur le rôle essentiel que joue cet aspect dans la collaboration. Nous donnons ici quelques exemples jugés pertinents de techniques de restitution de la conscience mutuelle qui peuvent être utilisées pour la *conception* d'un espace de travail collaboratif.

La conscience mutuelle fournit la réponse à deux questions générales et essentielles :

1. *"Qui" :* Qui accède à un objet partagé donné ?
2. *"Quoi" :* Quelle action cet utilisateur effectue-t-il sur cet objet ?

Ces deux questions, lorsqu'elles sont conjuguées au présent, relèvent de la collaboration synchrone ; lorsqu'elles sont conjuguées au passé, elles relèvent de la collaboration asynchrone.

Beaudouin-Lafon & Karsenty (1992) proposent la technique de *localisation* afin de fournir un élément de réponse à la première question. La localisation permet de percevoir quelle partie du document est visualisée par un tiers utilisateur, et ainsi d'en déduire qu'il peut potentiellement faire des modifications sur cette partie. Cette technique permet ainsi de déterminer qui accède à telle portion du diagramme partagé. La seconde question concerne la perception des actions des autres utilisateurs sur les objets partagés. Comme le soulignent Dewan & Choudhary (1995), afficher l'ensemble des détails des actions de tous les utilisateurs distants peut produire plus de désagrément que d'aide à la perception du contexte de collaboration. Plutôt que d'afficher l'intégralité des détails des actions distantes, il peut être préférable de ne montrer que l'action résultante. Par exemple, le déplacement d'une fenêtre peut être rendue en n'affichant que la transition *position de départ → position d'arrivée* et non l'ensemble des positions intermédiaires. Cette substitution d'une telle action longue par une action brève qui résume la première peut cependant s'avérer trop brutale et n'aide finalement pas à percevoir la nature de l'action effectuée. Ce mécanisme de substitution peut être affiné en utilisant la technique de l'*écho* : plutôt que de substituer à une action longue son action "résumée", elle est remplacée par une action d'écho qui fournit une animation dont le but est de percevoir la nature de l'action. Les techniques d'écho et de localisation relèvent de la collaboration synchrone et distante. En asynchrone, nous avons mentionné plus haut que les techniques principales de restitution de la conscience mutuelle s'appuient sur la notification et la visualisation de l'historique.

Soulignons enfin que l'outeraction présentée à la section précédente n'est possible que s'il y a conscience réciproque, et principalement lorsque cette conscience concerne la localisation des utilisateurs (question "qui ?"). Le concept de place est lui aussi lié au concept d'outeraction : dès qu'un utilisateur se joint à une place, il est susceptible de pouvoir rencontrer les autres personnes qui y sont également présentes.

Dans notre modèle conceptuel, il est donc tout à fait essentiel de fournir des services de base qui permettent la mise en œuvre de techniques de restitution de la conscience mutuelle. Par exemple, la notion d'écho sera intégrée dans le modèle des actions que nous proposerons. Le modèle de représentation des documents devra aussi prendre en compte la capacité de localisation des collaborateurs. Cette facette "awareness" de notre modèle sera détaillée dans le chapitre relatif au modèle collaboratif (chapitre 6).

2.2.4 Composants collaboratifs

Nous avons introduit la notion de composant interactif dans la section 2.1.6 et son rôle essentiel en tant que substitut des applications dans notre approche centrée sur les documents,

par opposition aux approches centrées sur les applications. Les différents objets de l'espace de travail interactif sont vus en tant que composants logiciels. Il en découle que les objets partagés doivent également être abordés sous l'angle du composant logiciel. Ces objets étant répliqués sur les différents postes distants, nous parlons de *composants répliqués*.

Les différentes technologies mentionnées dans la section 2.1.6, telles que les JavaBeans ou les composants COM, ont évolué afin de prendre en compte la répartition (ou distribution) des objets sur des réseaux informatiques. Les composants COM de Microsoft ont évolué en composant DCOM[18] puis .NET, et les JavaBeans ont évolué vers une architecture permettant la distribution d'objets, les EJB[19] (Szyperski, 2002). Parallèlement à ces évolutions, le standard CORBA[20] a été mis en œuvre afin de proposer des architectures pour composants distribués indépendamment des plateformes et des langages utilisés (Geib et al., 1999). Si cette dernière approche est encore largement enseignée dans les universités puisqu'elle définit les principes fondamentaux relatifs aux objets distribués, elle n'est qu'assez peu utilisée dans les entreprises à cause de son caractère complexe (lequel provient peut-être d'une trop forte généralité de CORBA, par rapport aux autres technologies qui sont davantage liées à un environnement et à un langage). Retenons de CORBA la caractérisation des services généraux que doivent gérer tout environnement de développement de systèmes à composants distribués (Geib et al., 1999) :

Nommage Les objets distribués doivent être identifiables simplement. Ils sont ainsi nommés d'une manière unique pour un référentiel donné, unicité qui doit être garantie par le système global.

Notification Les objets doivent pouvoir produire des événements destinés à des objets consommateurs afin de permettre la notification.

Transaction Les objets peuvent être manipulés de manière transactionnelle, c'est à dire en prenant en compte un lot d'opérations effectuées sur un ensemble d'objets.

Sécurité La sécurité a pour vocation d'identifier et d'authentifier les clients des objets, de chiffrer et certifier les messages transitant entre les objets, de surveiller la manipulation des objets et de veiller à respecter les autorisation d'accès.

Versions Il est important de pouvoir suivre l'évolution des versions des objets. Ces versions successives peuvent rester compatibles et dans ce cas la dernière version se substitue à la précédente. Si elles ne sont pas compatibles, les deux versions doivent pouvoir cohabiter.

Concurrence La concurrence d'accès fournit des mécanismes typiquement basés sur les verrous.

Licences Tout comme les applications commerciales, les composants logiciels sont soumis aux règles des licences.

Cycle de vie Les objets doivent pouvoir être créés, copiés, déplacés et détruits.

Relations Les objets sont très souvent en relation les uns avec les autres. Ces différents liens doivent pouvoir être activés dynamiquement entre des objets répartis.

Persistance Les objets doivent pouvoir être stockés de manière stable sur différents types de support, du simple fichier à la base de données objet.

Externalisation L'externalisation d'un objet consiste à pouvoir l'extraire de son environnement distribué (de type CORBA ou autre) pour être manipulé par ailleurs. La persistance est notamment basée sur ce service.

Propriétés Les utilisateurs peuvent ajouter à des objets des valeurs nommées, ou propriétés, de manière dynamique. Ces propriétés "dynamiques" se rajoutent aux propriétés "statiques" définies par la classe de l'objet.

[18] "Distributed Component Object Model"
[19] "Enterprise Java Beans". Attention : les EJB ne sont cependant pas du tout semblables aux JavaBeans !
[20] "Common Object Request Broker Architecture"

Interrogation Il est intéressant de pouvoir interroger le référentiel des objets à l'image des bases de données. Le langage OQL[21] est typiquement adapté à ce problème (Alashqur et al., 1989).

Collections Une collection est un objet contenant (ou plutôt référençant) d'autres objets. Il peut être par exemple intéressant d'effectuer des traitements itératifs sur des collections résultant de requêtes.

Temps Ce service permet d'obtenir une horloge commune pour l'ensemble des objets distribués, notamment afin de les synchroniser.

Vendeur Ce service ressemble aux pages jaunes de l'annuaire téléphonique. Les fournisseurs de composants enregistrent leurs objets auprès du vendeur. Les clients précisent les types de services voulus et le vendeur se charge de retrouver les composants proposant ces services dans sa base fournisseurs (*i.e.* les pages jaunes).

Pour rendre ces points effectifs, il faut s'inscrire dans une architecture relativement lourde tant pour développement d'une application que pour son déploiement et sa maintenance. Tout comme nous n'utilisons pas les composants JavaBeans ou COM pour notre modèle, nous n'utiliserons les composants CORBA ou EJB. La raison principale tient en ce que les architectures proposées permettent de résoudre des problèmes très généraux alors que notre problème a des caractéristiques bien particulières : il est centré sur les documents et les instruments d'interaction, et il se repose sur une architecture *répliquée* (voir le chapitre 6). Ainsi les architectures distribuées concernent davantage les architectures centralisées et dédiées au Web. Elles permettent un dialogue entre objets distribués sans la nécessité de les répliquer. Notre approche par la réplication est beaucoup plus spécifique et reste dédiée au problème spécifique de la conception d'espaces de travail interactifs et collaboratifs. Elle ne nécessite ainsi pas toutes les fonctionnalités pré-citées.

Nous préciserons, dans le chapitre suivant, notre propre modèle de composant et définirons dans le chapitre 6 son extension aux *composants répliqués*. Nous nous affranchirons globalement des technologies existantes et donc de leurs contraintes. Le prix à en payer sera un développement plus laborieux de la boîte à outil d'évaluation du modèle.

2.3 Conclusion

La facette interactive du modèle DPI s'appuie sur deux éléments fondamentaux : l'instrument et le document.

L'instrument joue le rôle de médiateur entre l'utilisateur et l'objet d'intérêt qui est, le plus souvent, un document. L'interaction instrumentale définit les principes généraux des instruments et affine le concept d'interacteurs. Elle permet de réaliser des interfaces selon le principe de manipulation directe dans un contexte orienté objet : les utilisateurs peuvent manipuler directement des objets cibles en utilisant des instruments.

Le document est l'objet d'intérêt principal pour les utilisateurs. Il contient les données que ces derniers souhaitent visualiser ou /et éditer. Il joue à la fois le rôle de support de l'information (persistance) et de sa présentation à l'utilisateur. Notre modèle définit comment les instruments peuvent interagir avec les documents (chapitres 3 et 4).

Les objets de l'espace de travail sont abordés sous l'angle des composants logiciels (interactifs) : l'instrument et le document sont deux exemples de composants. L'approche de la collaboration est envisagée au travers d'un composant interactif unique, la "place", qui jouera le rôle de pivot pour la collaboration (chapitre 6). Ce composant prend en considération les aspects temporels et spatiaux de la collaboration.

[21] "Object Query Language"

Chapitre 3

Principes théoriques du modèle DPI

Contents

Dans ce chapitre, nous abordons les principes théoriques du modèle DPI (Beaudoux & Beaudouin-Lafon, 2001). Le chapitre 4 présentera leur mise en œuvre dans la boîte à outils *OpenDPI* et approfondira certains aspects du modèle.

3.1 Introduction

Avant de présenter le modèle DPI, il est important de bien situer à quels niveaux il intervient étant donnés les différents acteurs directement ou indirectement concernés par le modèle. La figure 3.1 synthétise le rôle du modèle selon trois niveaux : utilisateur, concepteur et programmeur.

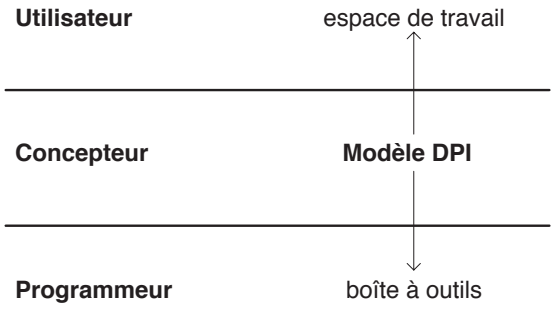


FIG. 3.1 – Les trois point de vue du modèle DPI

Le premier acteur concerné par le modèle est l'*utilisateur* lui-même. Si l'utilisateur n'a bien sûr pas à connaître les principes fondamentaux de DPI, il n'en demeure pas moins important de comprendre comment il est susceptible de le percevoir au fur et à mesure qu'il interagit avec son espace de travail. Les concepts de document et d'instrument doivent ainsi apparaître naturels pour les utilisateurs. Les *documents* définissent le support des données tandis que les *instruments* correspondent aux moyens de création et de modification des documents. Le modèle DPI est ainsi basé sur les *métaphores* du document et de l'instrument que l'utilisateur peut aisément appréhender sans avoir nécessairement conscience des détails constitutifs du modèle. Il sera important, dans la suite, de conserver à l'esprit cette appréhension du modèle par l'utilisateur. Par exemple, nous aborderons l'adaptation des instruments aux besoins des utilisateurs (section 3.2.5). La section 3.2, qui introduit les trois composants essentiels du modèle que sont le document, la présentation et l'instrument, concerne le niveau utilisateur.

Le second acteur impliqué dans le modèle DPI est le *concepteur*. Le concepteur est la personne qui définit tout ou partie de l'espace de travail en veillant à conserver "l'esprit" de DPI ainsi qu'à respecter les règles d'ergonomie des interfaces. Par exemple, il doit être capable de choisir et de définir des instruments ainsi que de veiller à leur cohérence. Le concepteur n'a pas à connaître nécessairement la manière dont seront programmées ses idées, mais il doit par contre veiller à ce que les utilisateurs aient une bonne perception du modèle DPI et, en particulier, de ses métaphores. Le concepteur est ainsi l'acteur qui doit connaître le plus précisément les fondements du modèle. Il est important de noter ici que cette thèse ne vise pas à définir les règles d'ergonomie quant à la conception d'un espace de travail basé sur DPI, pas plus qu'elle ne vise à fournir un espace de travail en particulier. L'objectif est de proposer les idées clés du modèle et les outils nécessaires pour *pouvoir* mettre en œuvre ces idées. La section 3.3, qui expose les principes de modèle DPI, concerne en priorité le niveau concepteur. Afin de préciser notre approche dans un contexte précis, nous décrivons une étude de cas dans la section 3.4.

Le troisième et dernier acteur concerné par le modèle est le programmeur. Le programmeur a en charge la réalisation logicielle des idées émises par le concepteur. Il utilise à cet effet la boîte à outils OpenDPI, laquelle permet de mettre en œuvre l'ensemble des principes de DPI. Le programmeur doit maîtriser l'utilisation de cette boîte à outils et connaître les principes du modèle de manière à appréhender correctement l'architecture de OpenDPI. Par contre, il a peu à se soucier de la manière dont l'utilisateur percevra les éléments qu'il construit. Les sections 4.1 et 4.2 du chapitre suivant fournissent les détails de la réalisation du modèle dans OpenDPI et sont donc essentielles pour le programmeur.

3.2 Composants essentiels du modèle

Nous définissons, dans cette section, les éléments de base du modèle DPI que sont le document, la présentation et l'instrument, puis nous analysons leur articulation en prenant en compte l'utilisateur. Ces définitions fournissent les bases pour l'élaboration des principes du modèle qui sont décrits dans la section 3.3.

3.2.1 Instrument

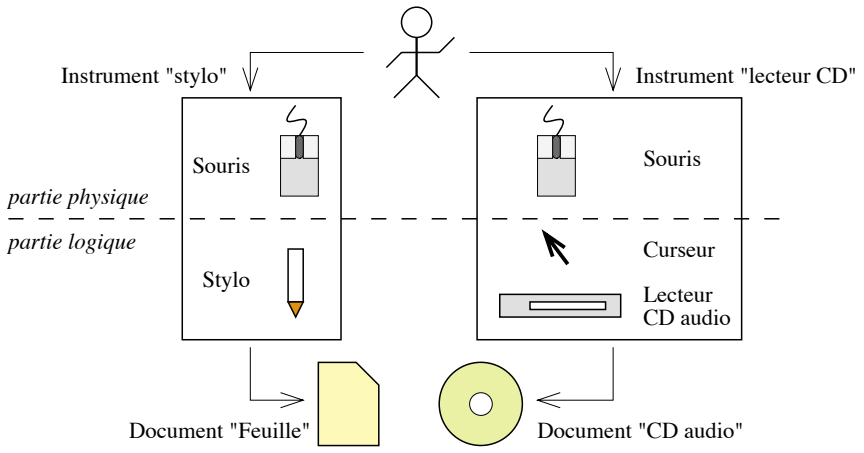


FIG. 3.2 – La métaphore de l'instrument

Quand nous agissons sur des objets réels, nous utilisons très souvent des outils ou des instruments : nous n'agissons que rarement directement sur ces objets. Par exemple, nous utilisons un stylo pour écrire sur du papier. Cette observation constitue la base de l'interaction instrumentale de Beaudouin-Lafon (2000). L'utilisation d'instruments conduit à des interactions plus ou moins directes. Par exemple, nous agissons directement sur la feuille de papier lorsque nous utilisons un stylo. Par contre, lorsque nous écoutons un CD audio, nous n'agissons pas directement sur le CD mais nous utilisons un lecteur approprié. Cet aspect de l'indirection des actions apparaît d'une manière semblable dans un système interactif qui *simule* les instruments de type stylo ou lecteur de CD audio et permet leur manipulation *via* une souris (figure 3.2). Dans la suite, nous appellerons *outil* tout instrument à caractère direct et *appareil* tout instrument à caractère indirect. Notons cependant que cette séparation n'est pas nécessairement nette et sera précisée dans la section 4.2.5.

Les deux exemples précédents illustrent le fait que les instruments ont deux facettes, la facette *physique* qui fait que l'instrument existe en dehors du système, et la facette *logique*

qui implique que l'instrument existe à l'intérieur du système tout en restant perceptible depuis l'extérieur :

> "Un instrument est composé d'une partie physique et d'une partie logique. La partie physique comprend les transducteurs d'entrée-sortie utilisés par l'instrument, en entrée pour capter l'action physique de l'utilisateur et en sortie pour lui présenter un retour d'information (...). La partie logique de l'instrument comprend en entrée la méthode de transformation des actions de l'utilisateur sur l'instrument logique et en sortie la représentation de l'instrument." Beaudouin-Lafon (1997)

Nous ne divisons cependant pas l'instrument en deux "sous-instruments", l'un physique et l'autre logique. Les utilisateurs doivent en effet ressentir qu'il y a continuité entre la partie logique et la partie physique. Par ailleurs, si la plupart des instruments ont pour rôle de modifier l'état des objets existants, d'autres permettent simplement de les observer, telles que les loupes et lentilles magiques de Perlin & Fox (1993). Ces derniers instruments fournissent des représentations complémentaires des documents (section 3.2.4). Nous appellerons dans la suite *lentille* tout instrument de perception[1]. Cette analyse nous conduit à classer les instruments d'interaction selon trois catégories :

1. Les *instruments directs*, métaphore de l'outil, sont manipulés en respectant une certaine forme de continuité entre la main et l'instrument : l'utilisateur agit directement sur l'instrument pour agir directement sur le document.
2. Les *instruments indirects*, métaphore de l'appareil, sont manipulés sans continuité directe avec l'objet d'intérêt : l'utilisateur (inter)agit directement sur l'instrument et indirectement sur le document.
3. Les *instruments de perception*, métaphore de la lentille optique, sont manipulés comme les instruments directs puisque l'utilisateur agit directement sur l'instrument (déplacement), mais agissent comme les instruments indirects puisqu'ils ne visent pas directement les document.

Notons que les instruments peuvent inter-opérer, c'est à dire que l'action d'un instrument peut viser un autre instrument et non un document. Par exemple, l'instrument "pinceau" peut permettre de peindre un objet graphique d'un document, le curseur d'un outil ou un bouton d'une boîte de dialogue. Le terme de méta-instrument, définit par Beaudouin-Lafon (1997), rejoint ce propos. Cependant, nous n'utiliserons pas un tel terme puisque un instrument (le pinceau par exemple) *n'est pas* un méta-instrument, mais *peut avoir* un comportement "méta" (*i.e.* agir sur un autre instrument) dans certaines circonstances. L'exemple du pinceau illustre, à notre sens, davantage la notion d'instrument *générique* (section 2.1.4) que de méta-instrument.

La définition que nous donnons de l'instrument est la suivante :

> *Un instrument est le médiateur entre l'*acteur*, défini comme étant l'entité qui agit en premier lieu, et l'*objet d'intérêt*, défini comme étant l'entité vers laquelle l'acteur souhaite agir.*

L'acteur est généralement l'utilisateur et l'objet d'intérêt un élément de document. Cependant, l'acteur est dans certain cas un objet du système. Par exemple, l'instrument "guide magnétique", qui attire tout objet magnétique situé en sa proximité, appartient au système. Nous donnerons plusieurs exemples d'instruments de natures différentes dans le chapitre A.

[1] Notons dès à présent qu'une présentation n'est pas un instrument de perception puisqu'elle ne dispose pas de partie logique (section 3.2.3).

3.2.2 Document

Un document physique, tel qu'une feuille de papier, un livre ou une partition musicale, est perçu par l'utilisateur selon deux facettes qui se confondent :

1. La facette *persistance* :
   Le document (*i.e.* le papier) est le support de la persistance puisqu'il absorbe l'encre du stylo. Quand il écrit sur un document, l'utilisateur perçoit le résultat persistant de son action (feed-back). En lisant le document, l'utilisateur visualise en premier lieu la facette persistance et interprète ensuite son contenu.
2. La facette *présentation* :
   La présentation est intrinsèque au document et lui confère une présence et une apparence concrètes. Un document possède une présentation qui est le résultat direct de la persistance des informations écrites.

Si le document physique couple naturellement la persistance et la présentation, le document informatique découple *par nécessité* la persistance de la présentation. La persistance du document est gérée par le système de fichier, tandis que la présentation se situe au niveau des périphériques de sortie tel que l'écran. Ce découplage apparaît dans les systèmes usuels de par le besoin permanent de sauvegarder les documents.

Plusieurs présentations d'un même document peuvent être construites avec un tel découplage, ce qui inclut des présentations de nature différente. Par exemple, la présentation d'un texte peut se faire en visualisant d'une part, le plan du texte et, d'autre part, le contenu du texte ; la présentation d'une partition musicale peut être visuelle ou bien sonore. Cette capacité est intéressante du point de vue de l'interaction mais altère la métaphore initiale du document. Du point de vue de l'utilisateur, il est important de réduire le plus possible le découplage entre documents et présentations.

Étant donnée la définition précédente de l'instrument, nous pouvons constater sans peine que l'objet d'intérêt est très souvent un élément d'un document. C'est pour cette principale raison que notre modèle est centré sur les documents. Par ailleurs, du point de vue du concepteur et du programmeur, tous les éléments de l'espace de travail, ainsi que l'espace de travail lui-même, seront considérés comme des documents. Le modèle sera en conséquence unifié autour de la notion de document.

Nous donnons la définition suivante pour le document :

> *Un document est un objet persistant et perceptible que l'utilisateur peut lire, écrire et annoter.*

Le fait que l'espace de travail soit lui-même un document (contenant d'autres documents) respecte bien cette définition. Il s'agit bien d'un objet persistant : un utilisateur qui entre dans son espace de travail doit le retrouver tel qu'il l'était la dernière fois qu'il l'a quitté. L'espace de travail est évidemment perceptible et l'utilisateur peut le lire (*i.e.* consulter son contenu), y écrire (par exemple l'ouverture d'une fenêtre modifie l'état de l'espace de travail), ou l'annoter (par exemple en y créant une note de type Post-It).

3.2.3 Présentations multiples

Si chaque document ne pouvait avoir qu'une unique présentation, la notion même de présentation deviendrait inutile du point de vue de l'utilisateur. Cependant, l'intérêt des présentations multiples est bien connu (le système Zelig de Celentano et al. (1992) illustre

ce propos). Notre objectif est de préserver la métaphore initiale du document tout en rendant possible l'utilisation de présentations multiples. Ainsi, du point de vue de l'utilisateur, chaque présentation du document *serait* le document lui-même.

Nous pourrions introduire la notion de présentations multiples par le biais des métaphores "caméras" et "moniteurs vidéo" du système X_{TV} (Beaudouin-Lafon et al., 1990) : un document est filmé par une ou plusieurs caméras et peut être visualisé *via* l'équivalent des moniteurs vidéo. Cette métaphore offre l'avantage d'être facile à comprendre. Cette approche préserve la métaphore initiale du document mais va cependant à l'encontre du principe de manipulation directe puisque les présentations alternatives demeurent passives.

Le modèle DPI étend cette métaphore en rendant actives les présentations alternatives, un document pouvant être édité au travers de *n'importe quelle* présentation. Ceci implique une nécessité fondamentale : les résultats d'édition doivent être *synchrones* entre les différentes présentations. Ceci permet d'assurer que la notion abstraite des présentations multiples devient concrète pour les utilisateurs. Ils observent en effet que les présentations multiples sont synchronisées et relèvent donc du *même* document.

L'utilisation de présentations multiples ouvre également la voie à l'édition partagée : il devient possible pour plusieurs utilisateurs (travaillant ou non sur la même machine) d'éditer simultanément un document au travers de présentations différentes mais synchronisées.

3.2.4 Boucle action - perception

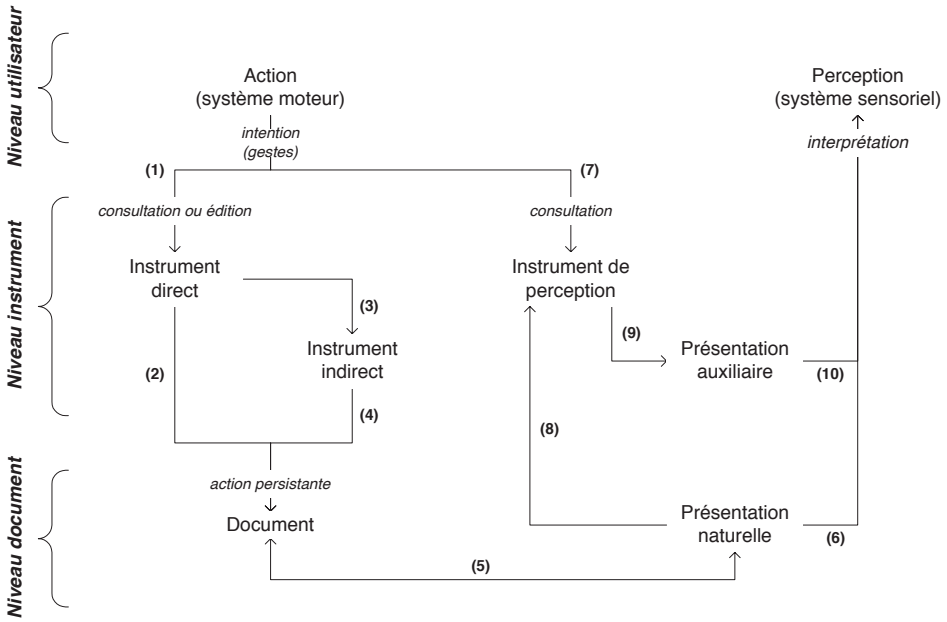


FIG. 3.3 – Action et perception dans DPI

Le modèle DPI met en jeu ses trois composants élémentaires, l'instrument, le document et ses présentations, selon un principe basé sur la théorie de l'action de Norman & Draper (1986). La figure 3.3 présente ce principe sous la forme d'un diagramme du flux d'action et de perception à trois niveaux : utilisateur, instrument et document.

1. Le niveau utilisateur précise que les utilisateurs spécifient leurs *intentions* par le biais d'*actions* (1, 7) et interprètent leurs résultats en utilisant leur sens (6, 10).

2. Le niveau instrument décrit comment les instruments transforment les actions des utilisateurs :

   Les instruments proposent deux types d'actions : l'édition (1) et la consultation (1, 7). La consultation peut être prise en charge par un instrument direct (1) tel qu'un instrument de défilement, par un instrument indirect (3) tel qu'un instrument de recherche, ou par un instrument de perception (7) tel qu'une la vue radar. L'édition peut être réalisée par un instrument direct (1) tel qu'un stylo ou par un instrument indirect (3) tel qu'un correcteur orthographique.

   La perception est induite par la présentation naturelle d'un document (6) ou indirectement par des instruments de perception (8), telle qu'une loupe, qui fournissent des présentations auxiliaires (9).

3. Le niveau document illustre le rôle double des documents : la persistance des actions de l'utilisateur (2, 4) et la présentation de son contenu (5).

Comme l'indique l'approche écologique de la perception de Gibson (1979), l'action et la perception sont fortement couplées : l'utilisateur doit percevoir avant d'agir (par exemple, en localisant un objet avant de le sélectionner) et doit agir afin de percevoir (par exemple, en navigant dans un document afin d'en retrouver une partie spécifique). Le couplage entre action et perception reste effectif pour un temps de cycle faible, typiquement inférieur à 100 ms. Le cas des interfaces conversationnelles n'entre donc pas dans ce schéma. Notre modèle prend en compte ce couplage en mettant en jeu les trois composants élémentaires dans la boucle action-perception. De plus, un couplage effectif entre action et perception nécessite de définir des règles de compatibilité entre les trois niveaux utilisation, instrument et document :

1. La compatibilité *utilisateur* ↔ *instrument* :
   L'interaction proposée par un instrument doit correspondre à sa fonction. Par exemple, un outil « pinceau » devrait être utilisé pour appliquer une couleur à un objet. Cet aspect correspond au concept d'*affordance* de Gibson (1977) : les instruments doivent exprimer leurs fonctions d'une manière directement perceptible.
2. La compatibilité *instrument* ↔ *document* :
   Un instrument travaille avec des types spécifiques de documents ou d'objets de document. Par exemple, un stylo est adapté à la feuille de papier puisque le papier peut recevoir l'encre du stylo. Cette compatibilité définit les interactions possibles entre les instruments et les documents.
3. La compatibilité *document* ↔ *utilisateur* :
   La présentation naturelle d'un document doit être adaptée à nos sens. Par exemple, une feuille de papier est compatible avec le perception visuelle alors qu'un écran Braille est compatible avec la perception tactile (et éventuellement visuelle). Cette compatibilité relève de l'ergonomie des interfaces homme-machine.

La notion de compatibilité est naturelle dans notre vie quotidienne. Nous n'avons généralement pas directement conscience d'une compatibilité mais plus souvent d'une *absence* de compatibilité. L'existence de la compatibilité rend possible un ensemble de combinaisons entre les documents électroniques, les instruments et les présentations. En séparant les instruments des documents, le modèle permet aux instruments d'agir sur des documents de natures variées, augmentant ainsi le nombre de combinaisons possibles. De telles combinaisons doivent simplifier l'interaction et réduire la charge cognitive des utilisateurs tout en offrant un ensemble de fonctionnalités riche. Par exemple, un même instrument peut être utilisé pour changer la couleur du titre dans un document ou la couleur d'un trait d'un graphique vectoriel. Les environnements actuels, basés sur la métaphore du bureau, n'offrent pas de telles capacités.

3.2.5 Principe d'indépendance et adaptation des instruments

La section précédente a mis en lumière l'indépendance des trois composants fondamentaux D, P et I. Cette indépendance s'inscrit dans une boucle action-perception qui établit les relations entre les composants. Cet aspect d'indépendance vise à fournir une combinaison riche quant aux capacités interactives d'un système basé sur DPI. Cependant, il est important de souligner que le concepteur ne saurait concevoir les composants indépendamment les uns des autres. Par exemple, si le stylo et la feuille de papier ne se "connaissent" pas, il n'en demeure pas moins qu'ils ont été conçus d'une manière non indépendante. Nous définirons formellement le *principe d'indépendance* des composants lorsque nous aborderons le fonctionnement du modèle qui, pour le moment, est décrit sous sa facette théorique. Il sera montré que les instruments seront les principaux vecteurs de cette indépendance puisqu'ils ne seront que faiblement couplés aux présentations.

Cependant, le fait de rendre possible l'interaction sur un document *via* un instrument qui n'a pas été *a priori* spécialement étudié pour interagir avec lui, peut poser le problème du manque d'adaptation entre l'instrument et le document. Par exemple, un bricoleur A souhaite dévisser une vis cruciforme mais il ne dispose que d'un ensemble de tournevis plats. Il imagine alors pouvoir utiliser un tournevis plat suffisamment petit pour pouvoir l'introduire dans la tête de la vis cruciforme et, au bout du compte, la dévisser. Si cette opération reste possible, elle peut conduire également à une détérioration de la vis qui la rendrait presque impossible à dévisser. Un bricoleur B, plus prévoyant que le bricoleur A, aura l'idée d'utiliser un embout adaptateur qui lui permet d'utiliser le tournevis plat en l'adaptant à la forme en croix. Dans cet exemple, nous voyons que l'utilisateur B *adapte* l'instrument initial à une fonction qui ne lui était pas allouée. Nous parlons dans ce cas d'*adaptation en sortie* de l'instrument, adaptation qui concerne la compatibilité instrument $\leftrightarrow$ document.

Le concept d'adaptation peut être complétée en poursuivant l'exemple précédent. Le bricoleur A dispose maintenant d'un tournevis cruciforme bien adapté à l'objet d'intérêt (la vis). Cependant, il ne dispose pas d'une poigne suffisante pour débloquer la vis. Jugeant que le manche du tournevis n'agrippe pas assez sa main, il entoure ce dernier d'un ruban adhésif en caoutchouc : ainsi, le bricoleur décuple sa force et parvient finalement à ses fins. Dans cet exemple, l'utilisateur a effectué une *adaptation en entrée* de l'instrument, adaptation qui concerne le compatibilité utilisateur $\leftrightarrow$ instrument.

Nous définissons l'adaptation des instruments comme suit :

> *L'*adaptation *(d'un instrument par l'utilisateur) concerne la capacité d'un instrument à pouvoir s'adapter à des interactions et à des fonctions pour lesquelles il n'a pas été initialement prévu.*
>
> *Une adaptation en* entrée *a lieu sur la partie physique de l'instrument et permet d'adapter l'interaction. Une adaptation en* sortie *se situe sur la partie logique de l'instrument et permet d'adapter sa fonction.*

Ces exemples illustrent qu'il est possible d'utiliser un instrument dans un contexte non prévu initialement par le concepteur, mais *étendu par l'utilisateur* lui-même.

3.3 Principes du modèle DPI

La figure 3.3 on page 83 illustre, du point de vue de l'utilisateur, le rôle des composants DPI dans la boucle action-perception. Nous précisons dans cette section *comment* de tels composants peuvent communiquer les uns avec les autres en définissant le *protocole de communication inter-composant* de DPI. Ce protocole est abordé ici d'une manière théorique et sera concrétisé dans les sections 4.1 et 4.2 par la description de son implantation

dans la boîte à outils OpenDPI. Cette section est importante pour le concepteur puisqu'elle évoque les principes du protocole sans présenter les détails de sa réalisation logicielle.

3.3.1 Interaction sur les documents

3.3.1.1 Principe

La communication entre un instrument et un document, qui concerne la compatibilité instrument ↔ document, s'effectue *via* le vecteur appelé *action*. Une action est une grandeur qui peut être produite par (la partie logique de) l'instrument et consommée par l'une des présentations du document, dans le respect de la boucle action - perception.

Un instrument transforme les *actions gestuelles* (ou gestes) de l'utilisateur, mesurées par les périphériques d'entrée utilisateur, en *actions intentionnelles* (ou actions) susceptibles d'être produites par l'instrument. L'instrument est ainsi qualifié de *producteur* (d'actions).

Une présentation est constituée d'un ensemble de *représentations* (typiquement graphiques) organisées hiérarchiquement. Les représentations sont ainsi des *nœuds* (typiquement graphiques) d'un arbre et la présentation peut-être considérée comme un nœud racine associé à un document particulier. Chaque représentation définit les actions qu'elle est capable de consommer. La présentation et ses représentations filles sont ainsi qualifiées de *consommatrices* (d'actions).

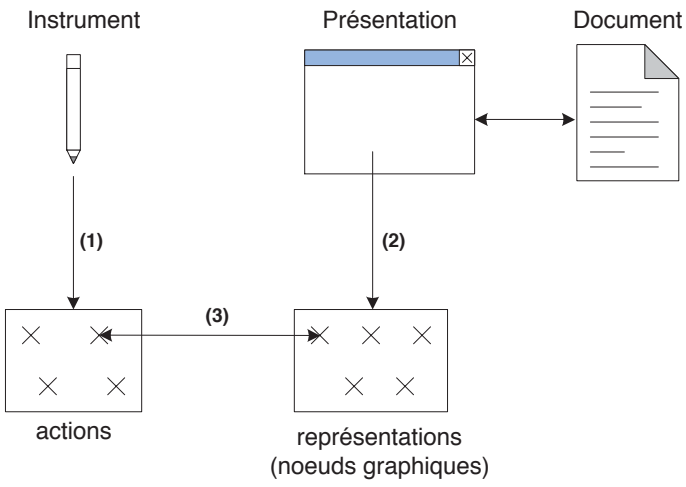


FIG. 3.4 – Interaction sur un document

La figure 3.4 illustre le chaînage qui s'opère entre la production d'une action par l'instrument producteur et sa consommation par la présentation consommatrice :

1. Lorsqu'une interaction est effectuée par l'utilisateur sur l'instrument, elle correspond à une intention particulière de l'utilisateur et donc à une action (intentionnelle) appartenant à l'ensemble des actions productibles de l'instrument.
2. Par ailleurs, l'utilisateur précise *via* l'instrument (directement ou indirectement) qu'elle représentation est la cible de son action. Cette représentation définit quant à elle les actions consommables qu'elle sait gérer.
3. Lorsque l'interaction a effectivement lieu, l'instrument effectue un *test de compatibilité* entre l'action qu'il doit produire et l'ensemble des actions que la représentation

ciblée est en mesure de consommer. Dès lors que le test retourne au moins une action consommable compatible, le chaînage de l'interaction peut avoir lieu : l'instrument transmet l'action productible (choisie par l'utilisateur si plusieurs actions sont compatibles) vers la représentation qui la consomme.

3.3.1.2 Compatibilité instrument ↔ document

Le test de compatibilité entre l'action productible par l'instrument et la représentation s'effectue ainsi :

- Soit a_p l'action productible et A_C l'ensemble des actions consommables par la représentation.
- Si $\exists\, a_c \in A_C$ tel que $a_p = a_c$ alors le test retourne le singleton $\{a_c\}$[2].
- Sinon, si $\exists\, a_c \in A_C$ tel que $a_p \rightsquigarrow a_c$, où le symbole $\rightsquigarrow$ dénote la compatibilité entre deux actions (voir ci-dessous), alors le test retourne l'ensemble $\{a_c \in A_C / a_p \rightsquigarrow a_c\}$.
- Sinon le test retourne l'ensemble vide $\{\}$.

La compatibilité entre une action a_p et une action a_c, symbolisée par la relation $a_p \rightsquigarrow a_c$, caractérise le fait que a_p peut se substituer à a_c bien que les deux actions ne soient *pas* identiques. Nous avons mentionné plus haut l'exemple du tournevis plat utilisé pour dévisser une vis cruciforme : le bricoleur A a constaté (à sa façon) que "dévisser une vis plate" $\rightsquigarrow$ "dévisser une vis cruciforme". Pour préciser la relation $\rightsquigarrow$ dans un contexte plus proche des documents électroniques, prenons l'exemple du pinceau :

- Un pinceau P_{rgb} peut produire l'action p_{rgb} qui consiste à peindre l'objet avec la couleur rgb définie par ses trois composantes rouge, vert et bleu.
- Un graphique vectoriel G_{rgba} peut consommer l'action p_{rgba}, consommation qui consiste à changer la couleur du graphique selon les quatre composantes rouge, vert, bleu et alpha[3].

Il semble assez naturel pour l'utilisateur disposant (uniquement) du pinceau P_{rgb} d'être tenté de l'utiliser sur le graphique G_{rgba} puisqu'il *sait* que ce graphique peut être peint (au sens large du terme). Ainsi, pour l'utilisateur, il serait intéressant d'avoir $p_{rgb} \rightsquigarrow p_{rgba}$. Cette compatibilité devient effective en considérant les deux options possibles :

1. Le coefficient a de p_{rgba} peut être forcé à 1 (pas de transparence) : le pinceau P_{rgb} applique ainsi une couleur opaque.
2. Le coefficient a de p_{rgba} peut prendre la valeur a du graphique G_{rgba} : le pinceau P_{rgb} peut ainsi appliquer sa couleur sans compromettre la transparence initiale du graphique.

Le choix de l'une des deux options peut être laissée au choix de l'utilisateur.

Il est évident que la compatibilité des actions est une relation non commutative : $a_1 \rightsquigarrow a_2$ n'implique pas $a_2 \rightsquigarrow a_1$. Dans l'exemple précédent, nous avons $p_{rgb} \rightsquigarrow p_{rgba}$. Il est intéressant de savoir s'il est possible de définir la compatibilité $p_{rgba} \rightsquigarrow p_{rgb}$ en considérant maintenant le matériel suivant :

- Un pinceau P_{rgba} peut produire l'action p_{rgba}.
- Un graphique vectoriel G_{rgb} peut consommer l'action p_{rgb} .

Cette deuxième compatibilité est en fait plus aisée à réaliser que la première : le coefficient a de p_{rgba} n'est simplement pas pris en compte par la consommation de p_{rgb} par G_{rgb}.

[2] A chaque action est associé un nom nécessairement *unique*. Deux actions sont égales (identiques) si elle ont le même nom.

[3] Alpha caractérise le degré de transparence.

Dans le premier cas, le pinceau P_{rgb} est *sur-utilisé* alors que, dans le second cas, le pinceau P_{rgba} est *sous-utilisé* (a non pris en compte).

En conclusion, il est possible d'établir la compatibilité entre des actions dès lors qu'il existe une *règle* (naturelle) de compatibilité entre les actions. En ce qui concerne les actions p_{rgb} et p_{rgba}, la compatibilité est réalisée sur leur *signature*.

Dans notre modèle DPI, les règles de compatibilité seront définies par des entités appelées *adaptateurs* d'action (ou simplement adaptateurs). Par exemple, l'adaptateur $p_{rgb} \rightsquigarrow p_{rgba}$ permettra à l'utilisateur d'utiliser le pinceau P_{rgb} pour peindre le graphique G_{rgba}. Notons cependant que l'utilisateur devra *explicitement* connecter l'adaptateur r $p_{rgb} \rightsquigarrow p_{rgba}$ sur l'instrument P_{rgb}, tout comme le bricoleur B utilise un embout adaptateur qui transforme son tournevis plat en tournevis cruciforme. Nous n'envisagerons pas de définir des règles d'adaptation implicites (*i.e.* ne nécessitant pas de branchement de la part de l'utilisateur), bien que ceci constitue un travail probablement intéressant qui compléterait notre modèle.

3.3.2 Couplage document / présentations

3.3.2.1 Principe

Bien que l'utilisateur interagisse sur l'une des présentations d'un document (figure 3.4), son intention initiale porte sur le document, la présentation fournissant le moyen d'accéder physiquement au contenu du document. Il est ainsi tout à fait naturel de parler de couplage document / présentation puisque, du point de vue de l'utilisateur, l'un ne saurait exister sans l'autre.

Un document, tout comme ses présentations, est constitué de nœuds hiérarchisés formant un arbre. Le document lui-même est le nœud racine (*i.e.* sans parent). La différence entre un document et une présentation tient dans la nature des données contenues : dans le cas d'une présentation, nous parlons de représentations (généralement graphiques) et, dans le cas d'un document, nous parlons de *données du domaine* (du document concerné). D'un certain point de vue, une présentation à caractère graphique est un document qui contient des données du domaine graphique.

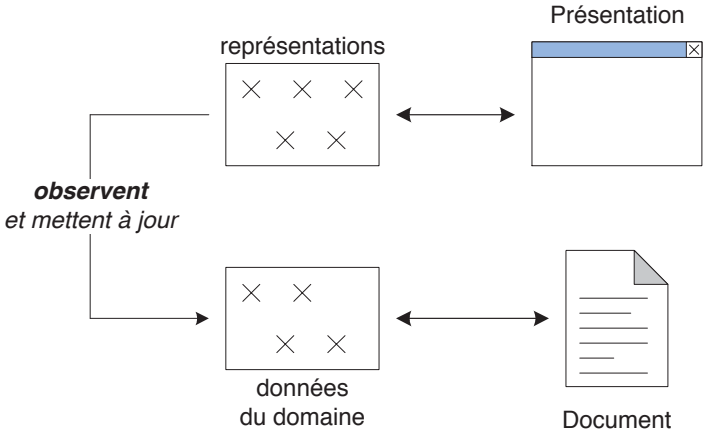


FIG. 3.5 – Couplage document / présentations

Le couplage document / présentation consiste à définir un mécanisme, qualifié de mécanisme d'observation, qui permet aux présentations d'observer tout changement de l'état du

document (figure 3.5). Cette observabilité permet aux présentations de rester synchronisées, sans pour autant préjuger de l'existence des autres présentations :

– Lorsque l'utilisateur effectue une action sur la présentation P_1 du document D, la présentation modifie son propre état *ainsi que* l'état de D.
– La présentation P_2 du même document D, observant l'état de D, peut alors mettre à jour son propre état s'il y a lieu.

Notons que le document n'observe pas l'état de ses présentations puisque la mise à jour du document se fait par les présentations elles-même.

L'*état* d'un nœud (représentant une données du domaine ou une représentation) est défini par les nœuds fils qu'il est susceptible de contenir ainsi que par un ensemble éventuellement vide de couples (nom, valeur) qui en définissent les *propriétés*. L'observation de l'état d'un nœud consiste alors en une *observation de structure* (ajout ou retrait d'un fils) ou/et en une *observation de propriétés* (changement de valeur).

Le mécanisme d'observation sera typiquement utilisé pour réaliser le couplage document / présentations mais peut être utilisé dans d'autres contextes. Par exemple, une représentation graphique contenant des nœuds fils peut être intéressée par l'observation de certaines propriétés de ses fils (voir l'exemple donné dans la section A.3.4).

3.3.2.2 Compatibilité document ↔ utilisateur

La construction des présentations est basée sur un mécanisme d'élection. A un nœud décrivant une donnée du domaine particulière est associé un ensemble *non vide* de représentations possibles : lorsqu'un utilisateur construit un document, il choisit les représentations qu'il souhaite utiliser pour présenter les données du domaine. Par exemple, lorsqu'un utilisateur souhaite définir un formulaire qui inclut la saisie d'une date, il peut utiliser une représentation de type champ d'entrée textuel ou bien une représentation plus évoluée permettant la navigation dans un calendrier. De même, lorsqu'un utilisateur fait une analyse statistique, il peut utiliser une représentation tabulaire pour afficher ses résultats, ou bien une représentation graphique. Le mécanisme d'élection autorise ainsi la compatibilité document ↔ utilisateur.

3.3.2.3 Compatibilité document ↔ présentation

L'association entre un type de donnée du domaine et un ensemble de représentations possibles de la donnée définit la base de la compatibilité document ↔ présentation. Comme nous l'avons fait pour les actions, nous étendons cette notion d'association avec la notion de compatibilité. Le test de compatibilité entre un *nœud* de donnée du domaine et une *représentation* est défini comme suit :

– Soit n le nœud du domaine, R_n l'ensemble des représentations possibles de n, et r_c la représentation dont on souhaite tester la compatibilité avec n.
– Si $r_c \in R_n$ alors le test retourne vrai signifiant que r_c est compatible avec n (cas trivial)
– Sinon, si $\exists\, r \in R_n$ tel que $r_c \rightsquigarrow r$, où le symbole $\rightsquigarrow$ dénote la compatibilité entre *deux représentations* (voir ci-dessous), alors le test retourne vrai.
– Sinon le test retourne faux.

La compatibilité $r_c \rightsquigarrow r$ entre la représentation r_c et la représentation r caractérise le fait que la représentation r_c peut se substituer à la représentation r bien que r_c ne fasse pas partie des représentations prévues pour représenter le nœud n. Par exemple, n peut être un nœud contentant du texte structuré en sections et sous-sections (comme cette thèse) et r une représentation graphique possible de ce texte qui utilise des attributs de style (nom et

taille de la police de caractère, numérotation des sections, etc.) afin de rendre correctement le contenu du nœud n. Une représentation r_c capable d'effectuer le rendu d'un texte simple (sans attributs de style, sans numérotation automatique) peut se substituer à r dans le cas où l'utilisateur ne dispose pas de r. Dans ce cas, il y a sur-utilisation de r_c. Le cas inverse correspond à une sous-utilisation : un texte simple doit pouvoir être rendu correctement par r. Nous pourrions compléter la liste des exemples en mentionnant une fois de plus l'exemple de la couleur rgb et $rgba$: la représentation d'une couleur $rgba$ peut être effectuée par une représentation qui n'affiche que les trois composantes rgb (sur-utilisation), de même qu'une couleur rgb peut être rendue correctement par une représentation qui affiche $rgba$ (sous-utilisation).

Cependant, nous n'intégrerons pas dans le modèle DPI la notion de représentations compatibles selon la relation $\rightsquigarrow$. En effet, lorsque l'utilisateur construit le document, il choisit d'abord des représentations et, par conséquent, les données du domaine qu'elles représentent. Ainsi, la compatibilité $r_c \rightsquigarrow r$ n'est pas utile lors de la construction. Par ailleurs, nous jugeons qu'il est préférable que le document contienne lui-même les différentes présentations qui lui sont associées. En procédant de la sorte, l'utilisateur n'a jamais à se préoccuper quant au choix d'une éventuelle représentation compatible parmi celles dont il dispose. Par ailleurs, ce choix "ergonomique" facilite la séparation présentations / données. Par exemple, la position de l'icone d'un fichier dans la représentation d'un dossier est du ressort de la présentation. Cependant, si les données du domaine sont les seules à être enregistrées, il n'est pas possible de restaurer convenablement la position de l'icone lors de l'ouverture ultérieure du dossier. Ce point caractérise une différence importante avec les documents "classiques" qui ne contiennent généralement que les données du domaine et non leurs représentations.

3.3.3 Principe d'indépendance des composants

Nous avons mentionné l'intérêt de définir des composants indépendants. Nous avons de plus illustré que ce principe permettait l'adaptation (en entrée et en sortie) des instruments aux besoins des utilisateurs. Le principe d'indépendance des composants DPI s'énonce ainsi :

> *Les composants D, P et I doivent être conçus de la manière la plus indépendante possible en utilisant le mécanisme de production et de consommation des actions, et le mécanisme d'observation de la structure et des propriétés des composants.*
>
> Les actions réalisent l'indépendance $I \leftrightarrow P$ *: Les producteurs d'actions que sont les instruments peuvent être conçus indépendamment des consommateurs d'action que sont les (re)présentations. Le couplage D / P induit alors l'indépendance* $I \leftrightarrow D$.
>
> L'observabilité réalise l'indépendance $D \rightarrow P$ *: Les présentations observent l'état du document associé si bien que le document n'a pas à connaître la nature de ses présentations.*
>
> *La relation de dépendance* $P \rightarrow D$ *est immuable et exprime le fait qu'une présentation connaît par nécessité la nature du document qu'elle représente.*

Ce principe est capital pour notre modèle. Il illustre l'intérêt qu'il peut y avoir, pour l'utilisateur, à mettre les composants de type instrument et de type document au centre du système interactif.

3.4 Étude de cas

3.4.1 Système étudié

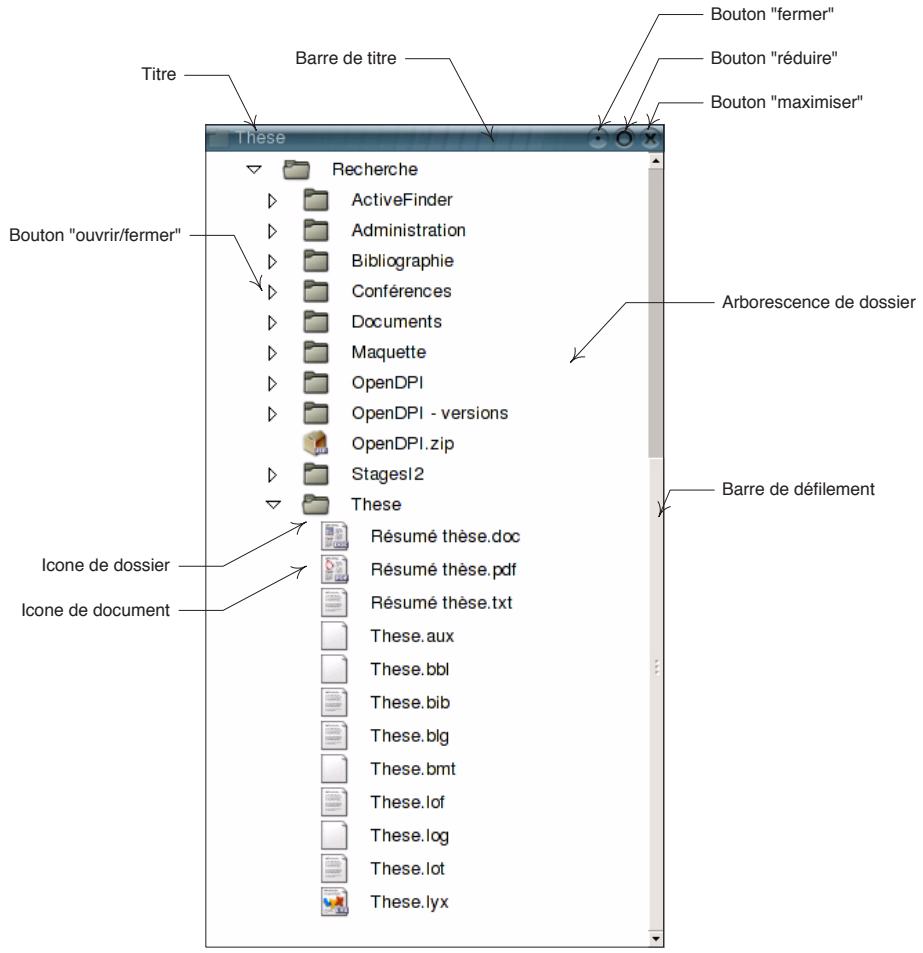


FIG. 3.6 – Étude de cas : le “finder”

Nous étudions dans cette section comment une application classique, de type gestionnaire de fichiers (“finder”), peut être abordée du point de vue du modèle DPI théorique[4].

La figure 3.6 décrit l’élément principal intervenant dans la tâche consistant à organiser ses fichiers personnels en dossiers : il s’agit d’une fenêtre affichant de manière hiérarchique le contenu d’un dossier. La fenêtre est ainsi constituée :

Barre de titre Elle contient la barre à proprement parler, laquelle permet le déplacement de la fenêtre dans l’espace de travail, le titre qui affiche le nom du dossier représenté dans la fenêtre, et un ensemble de trois boutons permettant respectivement de fermer, réduire ou maximiser la fenêtre.

[4]Du point de vue de DPI, il ne s’agit pas de définir à proprement parler une application, mais un ensemble de documents, de présentations et d’instruments qui permettent de réaliser des tâches préalablement identifiées.

Arborescence du dossier Elle permet la visualisation hiérarchique du dossier, l'arbre visualisé étant alors constitué de noeuds représentant les sous-dossiers et de feuilles représentants les fichiers.

Dossier Chaque dossier est représenté par un icone identifiant son état "ouvert" et "fermé", suivi d'un label donnant le nom du dossier. Cette représentation est complétée par un bouton à bascule triangulaire permettant de visualiser le contenu du dossier dans la même fenêtre.

Fichier Chaque fichier est représenté par un icone qui identifie le type du fichier, suivi d'un label qui fournit le nom du fichier.

Barre de défilement Elle permet de faire défiler l'arbre à l'intérieur de la fenêtre quand il a des dimensions supérieures à celles de la fenêtre.

Nous détaillons dans la suite comment un tel exemple peut être abordé du point de vue du modèle DPI théorique. Nous précisons trois aspects essentiels à notre démarche de conception : la définition des actions, la définition des documents et des présentations, et enfin la définition des instruments.

3.4.2 Définition des actions

Le premier aspect à aborder concerne la définition de l'ensemble des actions que doivent pouvoir consommer les différents éléments de l'interface proposée par le "finder". Étant donné le caractère polymorphe des actions, il est nécessaire de préciser le nom de l'action et ce qu'elle réalise quand elle est consommée. Notons qu'il n'est pas nécessaire (et pas souhaitable) de préciser ici quelles interactions et quels instruments permettent d'engendrer ces actions.

Ces actions sont données ci-dessous élément par élément :

Barre de titre

- *Activer* : met au premier plan de la fenêtre
- *Déplacer* : déplace la fenêtre dans l'espace de travail
- *Peindre* : change la couleur de la barre de titre, de ses boutons, et du fond de la fenêtre
- Bouton "fermer" :
  - *Activer* : ferme la fenêtre
- Bouton "maximiser" :
  - *Activer* : maximise la taille de la fenêtre (et transforme le bouton en un bouton "taille initiale" permettant de retourner à la taille initiale)
- Bouton "réduire" :
  - *Activer* : réduit la fenêtre à un icone affiché sur l'espace de travail.
- Titre :
  - *Ajouter / supprimer caractère* : édite le titre
  - *Peindre* : change la couleur du titre et du texte dans la fenêtre

Dossier

- Icone
  - *Ouvrir* : ouvre le dossier en affichant son contenu dans une nouvelle fenêtre hiérarchique
  - *Peindre* : applique une couleur de base à l'icone
  - *Supprimer* : supprime le dossier et son contenu
  - *Copier* : vide le presse-papier et y copie le dossier et (récursivement) son contenu
  - *Coller* : ajoute au dossier les éléments présents dans le presse-papier
- Titre

    - *Ajouter / supprimer caractère*s : édite le nom du dossier
    - *Peindre* : change la couleur du texte
  - Bouton triangulaire
    - *Activer* : étend l'affichage du dossier en "ouvrant" le noeud s'il était préalablement "fermé", ou compacte l'affichage en "fermant" le noeud s'il était préalablement "ouvert".

Fichier

  - Icone
    - *Ouvrir* : ouvre le fichier en affichant son contenu dans une fenêtre spécifique
    - *Peindre* : applique une couleur de base à l'icone
    - *Supprimer* : supprime le fichier
    - *Copier* : vide le presse-papier et y copie le fichier
  - Titre
    - *Ajouter / supprimer caractère*s : édite le nom du document
    - *Peindre* : change la couleur du texte

Barre de défilement

  - *Déplacer* : fait défiler le contenu de la fenêtre (ne prend en compte que le déplacement vertical)
  - Boutons haut et bas :
    - *Activer à répétition* : défilement d'une unité vers le bas ou vers le haut

La plupart des actions précédentes sont génériques puisqu'elles peuvent être appliquées à des éléments de nature différente. Elles sont de plus polymorphes car elles sont traitées de manière différente d'un élément à l'autre. Ces actions génériques sont : déplacer, activer, ouvrir, supprimer, et peindre. Nous étudierons comment ces actions pourront être mises en œuvre dans la section 3.4.4.

3.4.3 Définition des présentations et des documents

3.4.3.1 Présentations

Les présentations sont composées à l'intérieur du graphe de scène, ce dernier représentant l'espace de travail dans son intégralité. La figure 3.7 illustre la composition des présentations pour le "finder" :

- L'espace de travail est visualisé au travers de sa présentation, laquelle contient différents éléments graphiques. Dans notre exemple, cette présentation contient une fenêtre "finder" de visualisation d'un dossier.
- La fenêtre n'est pas une présentation. Elle contient par contre la présentation du contenu du dossier qu'elle affiche.
- La présentation du contenu du dossier est à son tour composée des présentations iconiques de chaque élément que le dossier contient (sous-dossiers ou / et fichiers) ainsi que des présentations du contenu des sous-dossiers s'ils ont été "ouverts" par le bouton triangulaire.
- La composition des présentations dossiers / fichiers est ainsi récursive et suit la hiérarchie des dossiers.

Chaque élément graphique, qu'il s'agisse d'une présentation ou d'un graphique simple, sait consommer les actions définies dans la section précédente.

3.4.3.2 Documents

Les présentations précédentes sont associées à des documents. Cependant, cette association n'est pas nécessairement perçue par l'utilisateur pour toutes les présentations. Par exemple,

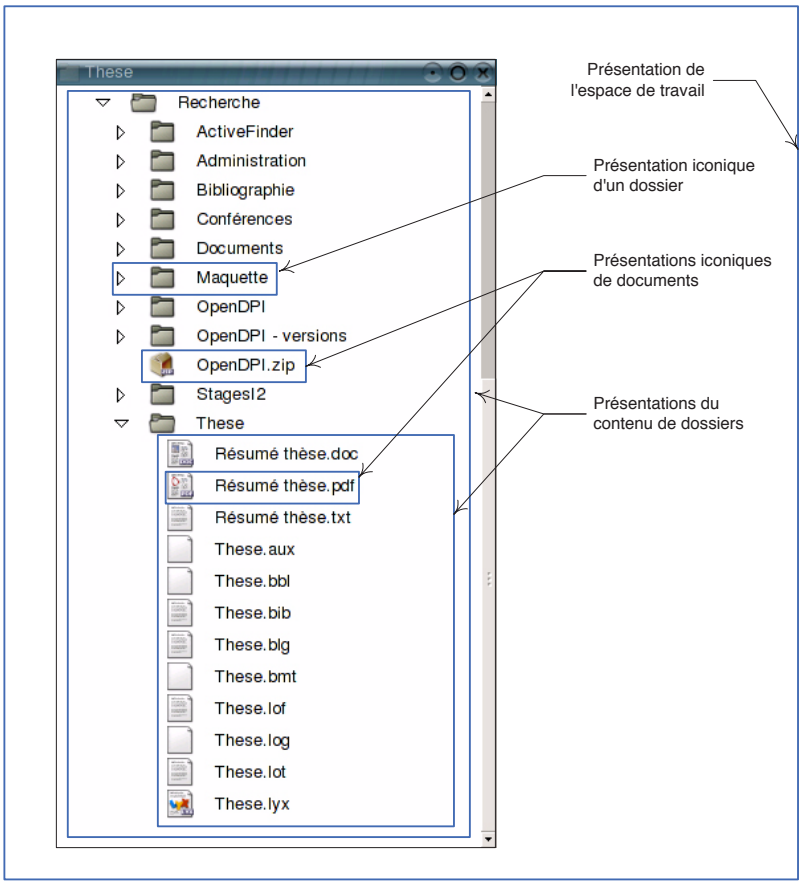


FIG. 3.7 – Composition de présentation du “finder”

l’utilisateur n’a pas nécessairement conscience qu’il existe un document associé à l’espace de travail.

La figure 3.8 illustre la manière dont sont composés les différents documents intervenant dans notre étude de cas.

L’espace de travail étant visualisé au travers d’une présentation, il est nécessairement associé à un document. Ce document définit l’ensemble des éléments de l’interface ainsi que les “préférences” de l’utilisateur relativement à cet espace. Il contient une représentation de la fenêtre “finder” sous deux formes : la fenêtre ouverte (représentée sur la figure 3.7) et la fenêtre “iconifiée” (non représentée sur la figure 3.7). Le document positionne la fenêtre ouverte dans le graphe de scène lorsque l’utilisateur demande l’ouverture de la fenêtre, ou bien positionne la fenêtre iconifiée si l’utilisateur demande de réduire cette fenêtre.

Un dossier est vu comme un document définissant le nom du dossier ainsi que son contenu. Le contenu du dossier est un ensemble de documents qui peuvent représenter des dossiers ou des fichiers. Ce contenu est précisé sous la forme de liens[5] vers les documents qui représentent les éléments que le dossier contient. Le document d’un dossier est associé à une présentation de dossier représentée dans la figure 3.7.

Un fichier est un document qui définit le nom du fichier ainsi que son contenu (non représenté dans la figure 3.8). Ce document possède une présentation iconique qui est celle

[5]Nous ne précisons pas la sémantique précise des flèches du diagramme de la figure 3.1. Nous détaillons ce point dans le chapitre 4.

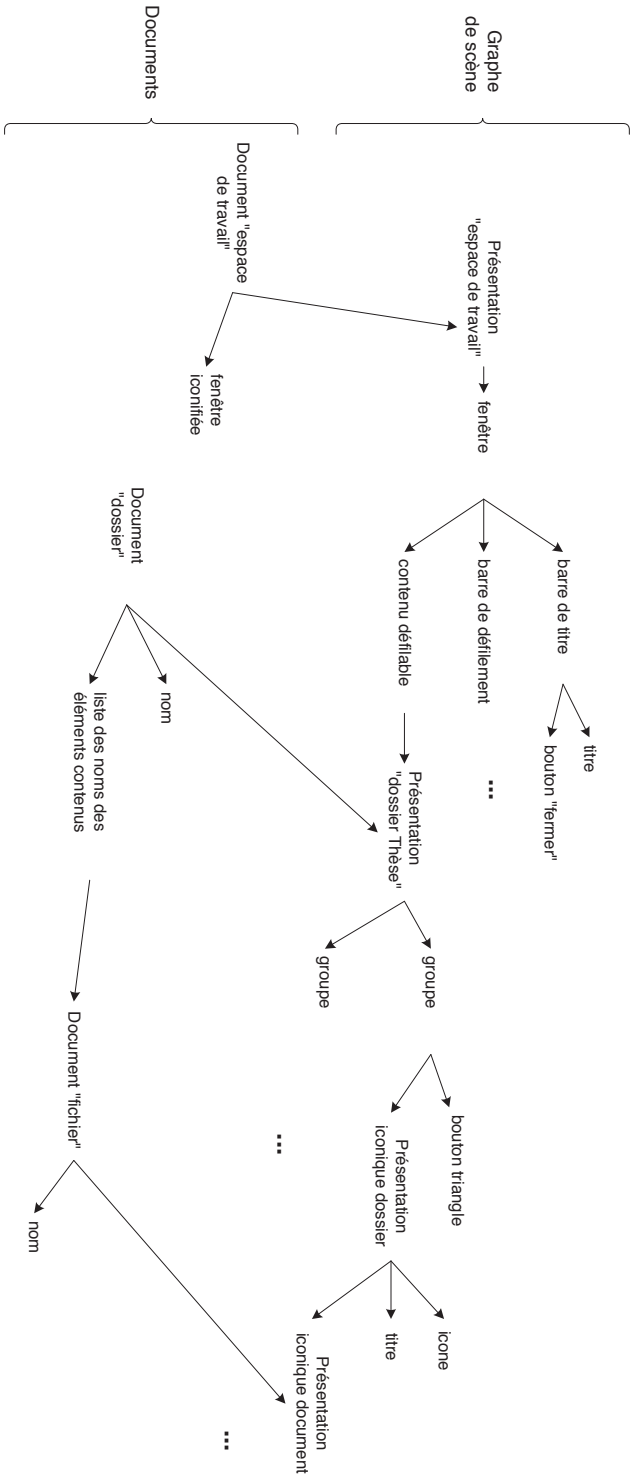


FIG. 3.8 – Arbres imbriqués des documents et présentations

affichée dans la présentation du dossier qui le contient. Notons que, tout comme pour la fenêtre de l'espace de travail, un fichier peut avoir plusieurs présentations : des présentations iconiques comme celles présentes dans le "finder" ainsi que des présentations du contenu des différents fichiers (non représentées).

Les présentations écoutent tout changement d'état du document de manière à pouvoir se mettre à jour. De plus, lorsque l'utilisateur interagit sur la présentation, le document associé est mis à jour. Par conséquent, si un même fichier est visualisé sous deux fenêtres différentes du "finder", le changement de son titre *via* l'édition du titre d'une de ses présentations iconiques engendre la modification de la propriété nom du document décrivant ce fichier. Toutes les présentations associées à ce document se mettront alors à jour de manière à refléter le nouveau nom. Selon un principe équivalent, la présentation du contenu du dossier associée à une fenêtre du "finder" peut avoir un titre synchronisé avec la propriété nom du document associé à ce dossier : il devient alors possible de changer le nom du dossier en éditant le titre de la fenêtre. La couleur que l'action "peindre" permet d'appliquer aux fichiers et aux dossiers est une information à caractère sémantique : nous reprenons le principe du "finder" du Macintosh pour lequel la "famille" d'un fichier est associée à une couleur que l'utilisateur peut paramétrer. Le document définit dans ces conditions la propriété "famille" tandis que les présentations définissent la propriété "couleur" associée. Cette synchronisation entre le document et ses présentations suit un principe assez analogue à celui mis en œuvre dans MVC (Krasner & Pope, 1988).

3.4.4 Définition des instruments

Dans les sections précédentes, nous avons précisé les actions que le "finder" permet de réaliser ainsi que la structure composite des documents et de leurs présentations. La dernière étape de conception consiste à définir les instruments qui permettront de produire les actions sur les présentations des documents.

3.4.4.1 Instruments indirects

Les instruments indirects ont déjà été définis puisqu'ils sont présents dans les représentations graphiques du "finder". Il s'agit des différents boutons ainsi que de la barre de défilement.

Les boutons "réduire", "maximiser" et "fermer" de la fenêtre du "finder" consomment l'action "activer". Ils produisent en retour les actions respectives "réduire", "maximiser" et "fermer" vers la fenêtre.

Les boutons triangulaires associés aux présentations iconiques des dossiers consomment l'action "activer" et produisent en retour l'action "ouvrir nœud" ou "fermer nœud" vers la présentation racine. L'action "activer" sera quant à elle produite par un instrument direct (section suivante).

L'utilisation de la barre de défilement suit sensiblement le même principe : elle produit l'action "défiler" vers la présentation racine. Nous ne présentons cependant pas cet instrument en détail, cela nécessitant une description principalement technique.

Les actions produites par les instruments indirects peuvent être ajoutées à l'ensemble des actions proposées par le "finder" décrit plus haut. Ces actions peuvent en effet être produites par d'autres instruments, en particulier des instruments directs (outils).

3.4.4.2 Instruments directs

Avant de décrire les instruments utilisables avec le “finder”, il est nécessaire de définir les périphériques d’entrée qu’ils utilisent. Nous choisissons pour cette étude de cas un profil classique qui comprend un clavier standard et une souris à un bouton.

Nous définissons deux outils : la “main” et le “pinceau”. La main est un instrument qui permet de réaliser les actions les plus usuelles et correspond au “pointeur” des environnements interactifs classiques. Le pinceau est un instrument qui permet de peindre différents éléments de l’interface, chaque élément réalisant cette action à sa façon (action polymorphe).

Action	**Périphérique**	**Interaction**
Activer	bouton souris	clic simple
Activer à répétition	bouton souris	clic simple long
Ouvrir	bouton souris	clic double
Réduire	bouton souris	clic double
Déplacer	bouton souris + (dx,dy) souris	appui bouton → translation souris → relâchement bouton
Fermer	(dx,dy) souris + ctrl-W clavier	pointage cible → appui touches ctrl → appui touches W
Copier	bouton souris + ctrl-C clavier	pointage cible → appui touches ctrl → appui touches C
Coller	bouton souris + ctrl-V clavier	pointage cible → appui touches ctrl → appui touches V
Supprimer	bouton souris + ctrl-D clavier	pointage cible → appui touches ctrl → appui touches D

TAB. 3.1 – Interactions de l’instrument direct “main”

Le tableau 3.1 synthétise la manière dont l’instrument “main” produit ses différentes actions. Trois types d’interaction entrent en jeu :

1. Les interactions clic simple, clic simple long et clic double utilisent uniquement le bouton de la souris et permettent de produire les actions “activer”, “activer à répétition”, “ouvrir” et “réduire”.
2. L’interaction “déplacer-lâcher” utilise le déplacement de la souris conjointement à son bouton et permet de produire l’action “déplacer” . Le consommateur est libre de ne tenir éventuellement compte que du déplacement horizontal ou vertical (polymorphisme de l’action déplacer).
3. L’interaction consistant à pointer la cible à l’aide de la souris et à effectuer un “raccourci clavier” permet d’appliquer diverses actions sur la cible.

Notons que, dans le point 3, nous n’utilisons pas de mécanisme de sélection mais un pointage direct. Une sélection peut s’avérer utile dans le cas où l’action concerne plusieurs éléments (section A.6.3).

La construction proposée de l’instrument “main” est consistante du point de vue des tâches liées à l’utilisation du “finder”. Cependant, elle peut s’avérer mal adaptée à d’autres tâches.

En vertu du principe d'indépendance des composants DPI, il n'est pas possible de prévoir dans quel contexte un instrument sera utilisé. Par exemple, il peut exister un contexte dans lequel une même interaction peut engendrer des actions différentes sur la cible ; dans pareil cas, l'instrument doit demander à l'utilisateur de choisir l'action à produire. Il est par conséquent nécessaire que l'utilisateur puisse personnaliser le profil d'interaction des instruments, tel que le profil proposé dans le tableau 3.1, de manière à l'adapter à ses propres besoins.

Action	**Périphérique**	**Interaction**
Peindre	(dx,dy) souris + bouton souris	pointage cible → clic simple
Récupérer la couleur	(dx,dy) souris + shift clavier + bouton souris	pointage cible → appui touches shift → clic simple

TAB. 3.2 – Interactions de l'instrument direct "main"

Le second et dernier outil[6] que nous envisageons est le "pinceau". Le profil d'interaction proposé pour le pinceau est simple puisque cet instrument ne produit que les deux actions "peindre" et "récupérer la couleur". L'action "peindre" utilise le pointage souris suivi d'un clic simple. Sa consommation par une cible a pour effet de modifier l'état de la cible. L'action "récupérer la couleur" est réalisée par une interaction semblable mais utilisant le "modificateur" shift du clavier. La consommation de cette action n'induit pas de modification de l'état de la cible : elle interroge la cible et attend en retour la "couleur" de l'objet ciblé (ce point est abordé dans la section A.6).

Le principe du "dialogue" instrument / présentation du modèle DPI respecte ce qui a été énoncé dans la section 3.3.1 :

- L'instrument détecte les gestes de l'utilisateur en observant l'état des périphériques associés.
- Ces gestes déterminent l'interaction faite par l'utilisateur. L'instrument peut alors décider, à l'aide du tableau précédent, quelle action il peut produire.
- L'instrument produit l'action si la cible a la possibilité de consommer une telle action (compatibilité des actions).
- La cible consomme l'action en effectuant le traitement approprié, lequel dépend de la nature de la cible (polymorphisme des actions).

La faisabilité de la consommation d'une action par un objet dépend du contexte. Par exemple, l'action "coller" est possible uniquement si la cible est capable de recevoir le type d'objet présent à ce moment dans le presse-papier : le fait de "coller" un fichier ou un dossier dans un dossier est possible, mais devient impossible pour d'autres types d'objet.

3.5 Conclusion

Nous avons proposé dans ce chapitre les principes fondamentaux du modèle DPI. Son élaboration a pris en compte les trois points de vue des différents acteurs que sont l'utilisateur, le concepteur et le programmeur. Nous avons décrit les deux métaphores du document et de l'instrument qui sont au cœur du modèle et nous les avons analysé du point de vue de l'utilisateur et du point de vue du concepteur. Nous avons proposé la notion de présentation de manière à autoriser la perception d'un document sous plusieurs facettes, altérant alors la métaphore initiale du document. Nous avons proposé les principes régissant l'échange

[6]Nous omettons volontairement l'édition de texte ; elle est abordée en détail dans la section A.6.4.

d'information entre les trois composants D, P et I, lesquels ont été illustrés dans une étude de cas.

La chapitre suivant développe la concrétisation du modèle DPI abordé ici de manière "théorique" en proposant une boîte à outil implantant la plupart des principes de DPI. Cette concrétisation concerne en priorité le concepteur et le programmeur.

Chapitre 4

De la théorie à la pratique

Contents

Dans ce chapitre, nous abordons la concrétisation du modèle théorique présenté dans le chapitre précédent. Cette concrétisation a été réalisée en construisant la boîte à outils OpenDPI.

La section 4.1 présente la facette document du modèle DPI. Le modèle des documents basé sur XML y est proposé, ainsi que le modèle des présentations basé sur le concept de graphe de scène. La notion d'observabilité entre les documents et les présentations complète et lie ces deux modèles.

La section 4.2 aborde la facette interactive de DPI, laquelle est fondée sur le concept d'instrument. La différence entre les actions gestuelles et les actions intentionnelles est explicitée. Le principe du chaînage des actions, depuis la source jusqu'à la cible, est précisée et réalisée selon un schéma de production et de consommation. Ce principe définit le cycle de production d'une action du producteur vers le consommateur. La réalisation des instruments est ensuite abordée et précise comment l'instrument peut effectivement jouer son rôle de médiateur entre l'utilisateur et les objets d'intérêt.

Une synthèse de la concrétisation du modèle DPI est proposée dans la section 4.3.

4.1 Les documents et leurs présentations

Dans cette section, nous abordons de proche en proche le modèle des documents mis en œuvre dans OpenDPI. En particulier, nous précisons comment s'organisent les parties données du domaine et présentations d'un document. Le modèle fourni prend en compte les différents aspects des documents introduits dans les sections 3.2 et 3.3.

4.1.1 Documents XML

Comme nous l'avons mentionné dans la section 3.3.2, un document est organisé hiérarchiquement en nœuds, chaque nœud précisant une relation de composition avec ses nœuds fils. De plus, un nœud définit des couples (nom, valeur) appelés propriétés. Nous utilisons le formalisme XML afin de décrire le contenu des documents (W3C, 2000), puisqu'il est en accord avec notre modèle hiérarchique basé sur les nœuds et les propriétés. Son intérêt majeur est de pouvoir attacher à un élément un sens précis : la *sémantique* d'un document XML peut donc être claire. Il est ainsi suffisamment général pour permettre la définition de tout type de document et offre l'avantage d'être un standard de fait. Il rendra notamment possible l'intégration de la notion de graphe de scène dans le document lui-même.

4.1.2 Graphe de scène

4.1.2.1 Définition

Un *graphe de scène* est un graphe qui fournit la représentation spatiale d'une scène contenant des objets géométriques variés. Il définit les objets qui y sont présents, leurs matériaux, leurs positions relatives ainsi que leur composition. Les graphes de scène sont adaptés à la description d'objets et de scènes en trois dimensions (3D).

L'exemple le plus significatif de boîte à outils centrée sur la notion de graphe de scène est probablement Open Inventor (Wernecke, 1994). Les graphes de scène y sont construits en composant divers types de nœuds dont nous citons les plus usuels :

Forme – Une forme définit les caractéristiques géométriques d'une forme 3D primitive tels qu'une sphère, un cube, des surfaces courbes, ou du texte.

Groupe – Les objets complexes sont construits à partir des primitives précédentes qui sont placées dans des groupes. Les groupes permettent ainsi de définir la composition des objets 3D et de la scène elle-même.

Matériau – L'objet peut "exister" (*i.e.* être perçu) dès lors qu'un matériau lui est associé. Un matériau peut-être une couleur émise ou une texture appliquée sur la surface de l'objet.

Transformation – Les positions respectives des différents objets dans la scène sont caractérisées par des transformations géométriques élémentaires. Il s'agit typiquement de translations et de rotations.

4.1.2.2 Intérêt

Si la notion de graphe de scène provient naturellement du registre de la 3D, il n'en demeure pas moins qu'elle s'avère tout à fait pertinente dans d'autres contextes. Par exemple, la boîte à outil Jazz, dédiée à la conception d'interface "zoomables" (section 1.2.2.1), met en œuvre la représentation de la surface "zoomable" et des objets qu'elle contient par le biais d'un graphe de scène (Bederson et al., 2000). Le standard SVG définit une instance d'XML décrivant des graphiques vectoriels 2D en mettant également en œuvre la notion de graphe de scène (W3C, 2003).

L'approche usuelle des boîtes à outils de conception d'interfaces graphiques consiste non pas à fournir des nœuds élémentaires qui, une fois assemblés, définissent la scène, mais à fournir des *fonctions* permettant de dessiner directement sur une surface, telles que les fonctions `drawLine` au `drawRectangle` (Knudsen, 1999). Les avantages de l'utilisation des graphes de scènes par rapport à cette approche usuelle sont les suivants :

Facilité de conception – De nouveaux objets sont conçus par assemblage d'objets primitifs. Leur *rendu* est alors tout à fait transparent pour l'utilisateur. Dans l'approche usuelle, le rendu est à la charge du programmeur.

Modèle unifié – L'ensemble des éléments graphiques, depuis les "widgets" jusqu'à la représentation des documents, est réalisé sous un *même* formalisme.

Transformations réifiées – Le graphe de scène gère de manière naturelle la notion de transformation, une transformation étant réifiée en un nœud du graphe. Ceci *facilite* la conception d'interfaces mettant en jeu des transformations, tels que le zoom des interfaces "zoomables" (Bederson et al., 2000) ou la rotation des "rotated windows" (Beaudouin-Lafon, 2001).

D'une manière générale, les graphes de scène sont adaptés à la représentation de graphiques vectoriels et permettent au concepteur de se concentrer sur la définition de la géométrie des objets et non sur leur rendu. De plus, le langage XML convient à la description d'un graphe de scène si bien qu'il peut être utilisé pour décrire le contenu sémantique des documents *et* la manière dont ils seront présentés à l'utilisateur. C'est un point important qui nous permettra d'aborder le document et les présentations sous la forme d'une *unique* classe de base, le *composant DPI*.

Les inconvénients de l'utilisation des graphes de scène découlent de la prise en charge systématique du rendu par la boîte à outils : cela nécessite davantage de ressources et s'effectue moins rapidement.

Dans notre modèle, nous utiliserons un graphe de scène basé sur une version simplifiée de Jazz appelée Piccolo[1].

[1] http ://www.cs.umd.edu/hcil/jazz/

4.1.3 Modèle des documents

4.1.3.1 Composants et propriétés

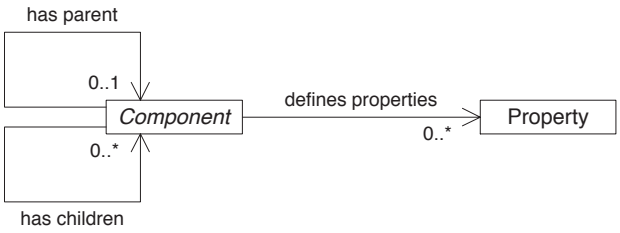


FIG. 4.1 – Atomes du documents : nœuds et propriétés

Un document, qu'il s'agisse de ses données du domaine ou de ses présentations, est défini hiérarchiquement sous la forme d'un arbre de composants. Chaque composant possède la capacité de produire ou/et de consommer différentes actions (section 4.2), et est ainsi un composant interactif.

Le modèle caractérisant l'état d'un composant est donné figure 4.1[2]. Chaque composant peut contenir des composants fils et être contenu dans un autre composant père. Un document est un cas particulier de composant racine : il n'est inclus dans aucun composant parent.

Chaque composant définit un ensemble de propriétés qui forment les feuilles de l'arbre du document. Ces propriétés caractérisent l'*état propre* du composant, indépendamment des composants fils qui caractérisent la *composition* des composants. Les propriétés sont définies *implicitement* par le composant d'une manière relativement analogue aux propriétés JavaBeans (Sun, 1997). Une propriété `x` de type `type` est définie par deux méthodes nommées `type getX()` et type `setX(type)`, respectivement qualifiées d'accesseur en lecture et d'accesseur en écriture. Une propriété est en *lecture seule* lorsque son accesseur en écriture est protégé (l'accesseur en lecture est toujours public) : cette protection autorise uniquement l'objet définissant la propriété à pouvoir la modifier[3].

Nous donnerons dans la suite quelques exemples de composition de composants et de propriétés de composants.

4.1.3.2 Données du domaine

Le document contient en premier lieu des données qualifiées de données du domaine. Ces données sont relatives au domaine abordé par le document et *ne préjugent pas* de la façon dont elles seront présentées à l'utilisateur.

La figure 4.2 donne la décomposition d'un document, nommé "Joli rectangle", qui contient la description d'une figure géométrique de type rectangle (deux représentations possibles sont données figure 4.3). Ce document est un nœud racine qui contient un unique nœud fils décrivant les données du rectangle. Le composant "Données rectangle" définit trois propriétés : les coordonnées du centre, les dimensions et la couleur. Le document décrivant un graphique, ces données sont bien des données du domaine (du graphisme). Les propriétés de ce composant sont définies ainsi :

[2]Nous utiliserons dans la suite, tant que faire se peut, le langage UML pour caractériser le modèle DPI (OMG, 2000).

[3]Cette légère différence avec les propriétés au sens des JavaBeans sera expliquée à la section 4.1.5.

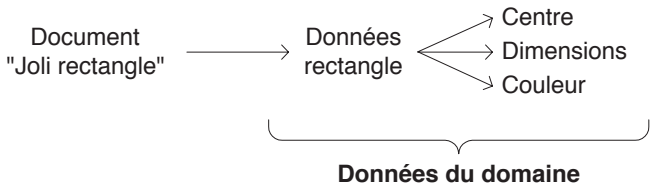


FIG. 4.2 – Exemple de données du domaine

```
public class RectangleData extends Component {
   //...
   public Location getCenter() {...}
   public void setCenter(Location c) {...}

   public Size getSize() {...}
   public void setSize(Size s) {...}

   public Color getColor() {...}
   public void setColor(Color c) {...}
}
```

Remarquons que les valeurs de ces propriétés sont définies en tant qu'instances des classes Location, Size et Color. Par conséquent, ces classes doivent nécessairement représenter des valeurs immuables de telle sorte qu'une référence retournée par un accesseur en lecture ne puisse servir à modifier ultérieurement la valeur de la propriété.

4.1.3.3 Présentations

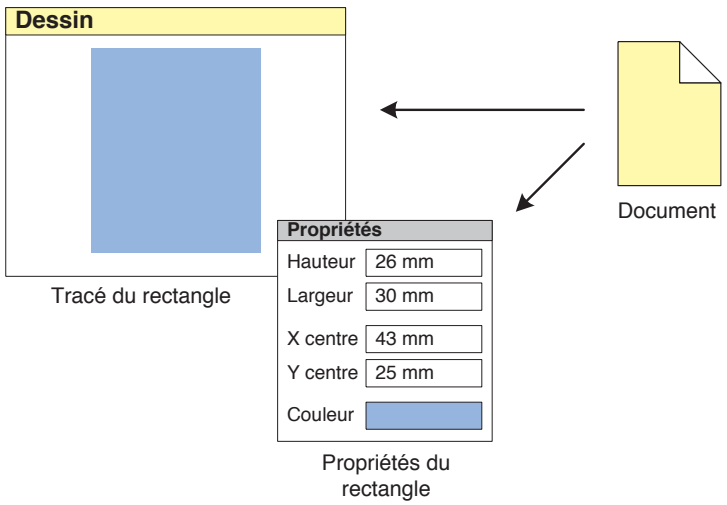


FIG. 4.3 – Deux présentations d'un même rectangle

Les présentations d'un document permettent la perception de l'état du document par l'utilisateur[4]. Le lien entre les présentations d'un document et le document sera précisé dans la

[4]Nous ne considérons dans cette thèse que le cas des présentations graphiques.

sous-section suivante (il ne s'agit pas d'un lien de composition). Le document de la figure 4.2 peut avoir les deux présentations graphiques de la figure 4.3 : la première présentation affiche le rectangle en tant qu'objet graphique, la seconde présente les propriétés du rectangle sous la forme de champs de saisie.

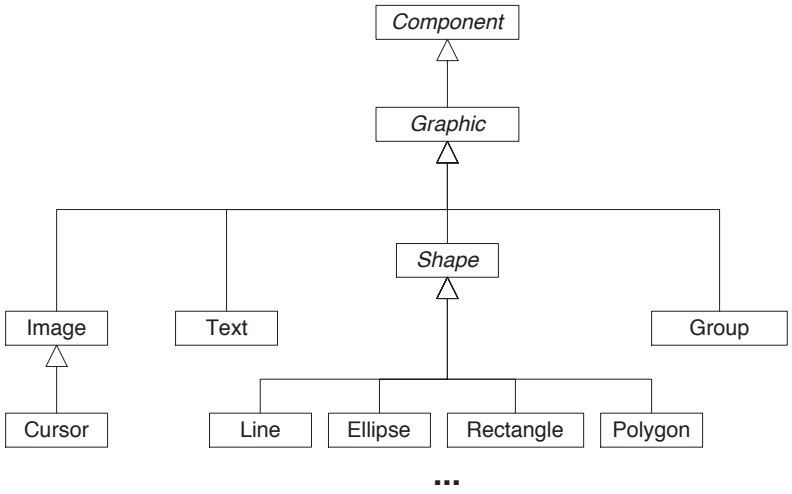
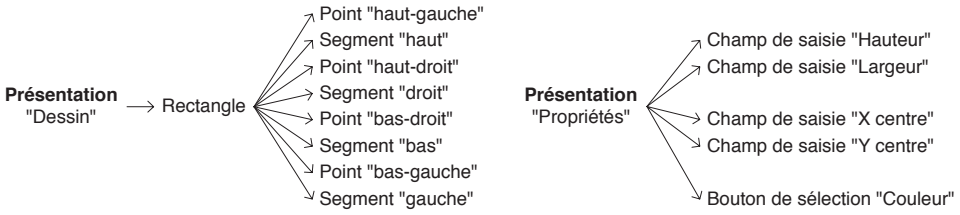


FIG. 4.4 – Classes des représentations graphiques

Les présentations sont insérées dans le graphe de scène au fur et à mesure des actions de l'utilisateur. Le composant de base contenu dans des présentations (graphiques) est défini par la classe Graphic (figure 4.4) : un *graphique* est un composant visuel élémentaire pouvant contenir, par composition, d'autres graphiques. Le graphe de scène est ainsi l'assemblage de briques graphiques de base, lesquelles peuvent être des images "bitmap" (classe Image et dérivées), du texte (classe Text), des formes géométriques vectorielles (classes Shape et dérivées), ou un groupe d'autres graphiques (classe Group). Une présentation (classe Presentation) est un cas particulier d'un groupe référencé par le document qu'elle présente (figure 4.7 on page 109). Notons que les graphes de scènes de OpenDPI et de Piccolo cohabitent par un mécanisme de délégation : chaque nœud du graphe Piccolo définit un lien vers un nœud DPI, et vice-versa.



(a) Présentation (b) Présentation

FIG. 4.5 – Graphes de scène des deux présentations du rectangle

La figure 4.5 donne les graphes de la présentation "Dessin" (a) et de la présentation "Propriétés" (b). La présentation "Dessin" contient le graphique "Rectangle" instance de CompositeRectangle : ce dernier est composé de quatre segments et de quatre points, et maintient entre ces huit éléments les contraintes de positionnement (de telle sorte que le rectangle reste toujours un rectangle). La présentation "Propriétés" contient quatre champ de

saisie pour définir la taille et la position du rectangle, ainsi qu'un bouton permettant la sélection d'une couleur.

Ces deux présentations font partie du graphe de scène global du système et sont liées au document "Joli rectangle" d'une manière précisée ci-après.

4.1.4 Documents et espace de travail

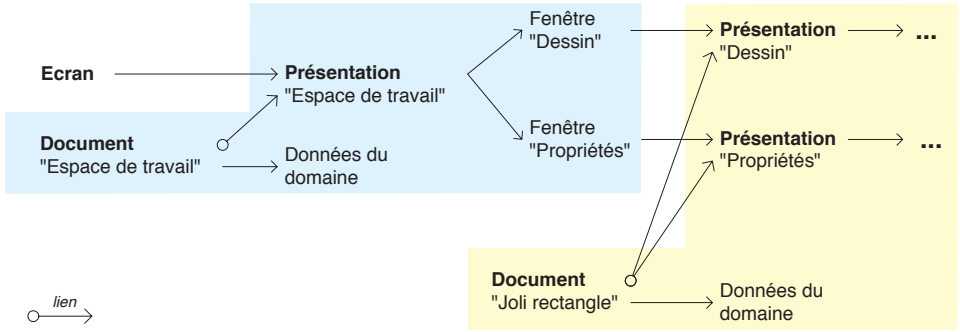


FIG. 4.6 – Imbrication des documents et du graphe de scène

La figure 4.6 illustre comment OpenDPI réalise l'imbrication de plusieurs arbres de documents dans le graphe de scène global du système. Les présentations du document "Joli rectangle" sont représentées à l'écran en étant insérées dans le graphe de scène comme suit :

- L'écran est un composant graphique particulier qui représente l'écran du système. C'est la racine du graphe de scène.
- L'écran affiche un espace de travail au travers de la présentation "espace de travail"[5].
- Cet espace de travail (figure 4.3) comporte deux fenêtres nommées "Dessin" et "Propriétés".
- Ces fenêtres contiennent les présentations "Dessin" et "Propriétés" du document "Joli rectangle".

Notre approche étant centrée sur la notion de document, l'espace de travail lui-même est considéré comme un document dont les données du domaine peuvent être, par exemple, les préférences de l'utilisateur sur cet espace. Nous constatons donc qu'il est nécessaire d'imbriquer différents arbres : dans notre exemple, il s'agit d'imbriquer les trois arbres dont les racines respectives sont l'écran, le document "espace de travail" et le document "Joli document". Cette imbrication n'est pas possible en utilisant la seule relation de composition : nous définissons en conséquence la relation document → présentation sous la forme d'un *lien*. Nous avons ainsi trois arbres indépendants, le graphe de scène, le document "espace de travail", et le document "Joli rectangle" et ses données du domaine, reliés entre eux par des liens de type document → présentation.

Ceci nous amène à définir les classes Document et Presentation de manière à faire apparaître ce lien de manière explicite (figure 4.7).

4.1.5 Observabilité

Dans l'exemple du "Joli rectangle", les deux présentations affichent le même document et doivent donc est synchronisées : dés que l'utilisateur modifie le rectangle *via* l'une de

[5]L'utilisateur peut définir plusieurs espaces de travail.

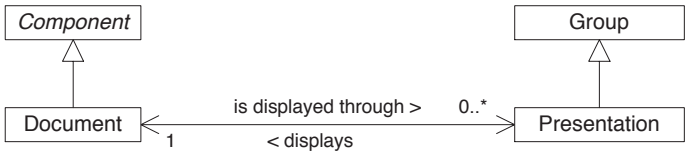


FIG. 4.7 – Modèle du document

ses présentations, les données du domaine doivent être mises à jour et leurs nouvelles valeurs doivent être envoyées à l'autre présentation. En conséquence, toute présentation doit observer les changements d'état du document auquel elle est associée : nous avons une observation "présentations → document".

Un autre contexte illustre que l'observation ne concerne pas uniquement le couplage document / présentation. Le "rectangle" de la présentation "Dessin" étant un rectangle composite, il est nécessaire que ce rectangle observe l'état des quatre segments et des quatre points qui le constituent de manière à appliquer les contraintes liées à la forme rectangulaire lorsque les coordonnées des éléments contenus changent. Nous avons donc une observation "présentations → représentations internes".

Nous allons en conséquence définir le mécanisme d'observation au niveau le plus général, *i.e* au niveau du composant DPI. Notons que ce mécanisme, en vertu des deux exemples précités, restera interne au document[6].

4.1.5.1 Observable et observateur

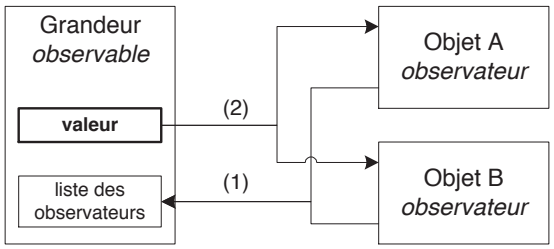


FIG. 4.8 – Grandeur observable et objets observateurs

Le mécanisme d'observation est basé sur le motif de conception observable / observateur (Gamma et al., 1994) : une grandeur qui peut être observée par un objet est dite *observable*, l'objet étant alors qualifié d'*observateur*. Le principe de fonctionnement se déroule en deux étapes (figure 4.8) :

1. Les observateurs s'inscrivent auprès des grandeurs observables qu'ils souhaitent observer.
2. Tout changement d'état des grandeurs observables est par la suite notifié aux observateurs inscrits.

Un composant DPI peut jouer à la fois le rôle d'observable et d'observateur. L'observation d'un composant observable par un composant observateur se fait *via* des *interfaces d'observation* : ces interfaces définissent les méthodes d'observation que tout observateur doit

[6]Cet aspect serait à revoir dès lors que la notion de lien entre deux documents est abordée.

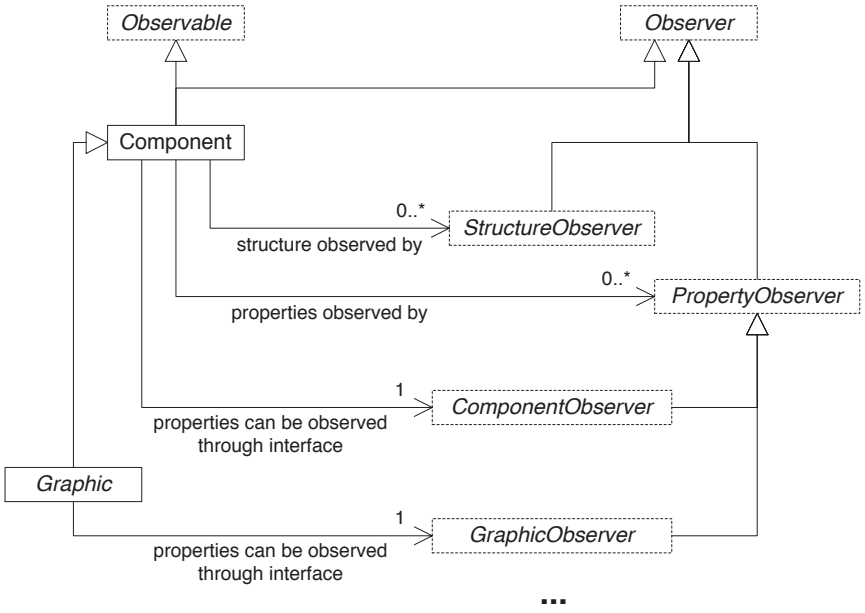


FIG. 4.9 – Modèle d'observation

implanter selon le mécanisme associé aux interfaces du langage Java (Eckel, 2002). Quand l'état du composant observable change, l'une des méthodes définies dans ces interfaces est invoquée sur chaque composant observateur.

L'état observable d'un composant étant constitué d'une part des valeurs de ses propriétés et d'autre part de sa structure arborescente, deux types d'interface d'observation sont définies (figure 4.9) : l'interface d'observation de la structure d'un composant (interface StructureObserver) et les interfaces d'observations des propriétés (interfaces PropertyObserver et dérivées).

4.1.5.2 L'interface d'observation de structure

La structure d'un composant évolue lorsque des composants fils y sont ajoutés ou retirés. L'interface StructureObserver définit ainsi les deux méthodes suivantes :

```
public void childInserted(Component child, int index) ;
public void childRemoved(Component child, int index) ;
```

Ces méthodes indiquent le fils concernés par l'ajout ou le retrait ainsi que sa position par rapport au composant parent. Un composant souhaitant observer la structure d'un composant c doivent s'inscrire en tant qu'observateur comme suit :

```
c.addStructureObserver(this) ;
```

L'objet this doit alors être instance d'une classe implantant l'interface StructureObserver de manière à ce que les méthodes childInserted et childRemoved puissent être invoquées sur this quand la structure de c évolue.

4.1.5.3 Les interfaces d'observation de propriétés

Les interfaces d'observation des propriétés s'utilisent d'une manière sensiblement identique à l'interface d'observation de la structure d'un composant : inscription de l'observateur, puis appel d'une méthode définie dans l'interface quand une des propriétés change d'état. Le mécanisme est seulement particulier étant donné le caractère implicite des propriétés.

Une classe décrivant un composant *Cccc* , si elle définit de nouvelles propriétés par rapport à sa classe de base, *doit* fournir une interface d'écoute de ses propriétés nommées *Cccc*Observer et héritant de PropertyObserver (figure 4.9). Cette fourniture est impérative car, de manière à respecter le principe d'indépendance, un composant ne doit pas préjuger de la possibilité d'écoute de son état par d'autres composants : cette écoute doit toujours être possible.

Tout comme les propriétés suivent une syntaxe particulière quant à leur définition (méthodes get et set), les interfaces d'observation des propriétés respectent une syntaxe précise : pour une nouvelle propriété *pppp* de type *type* définie dans la classe *Ccccc*, l'interface *Cccc*Observer doit définir la méthode new*Pppp*(*type*) dite méthode de *notification de valeur*. A chaque changement de la valeur de *pppp* d'une instance de *Ccccc*, tous les observateurs inscrits voient leur méthode new*Pppp* invoquée avec la nouvelle valeur de la propriété passée en argument. Cette invocation a lieu à la fin de la méthode set*Pppp* dans le cas où la valeur de la propriété est effectivement nouvelle.

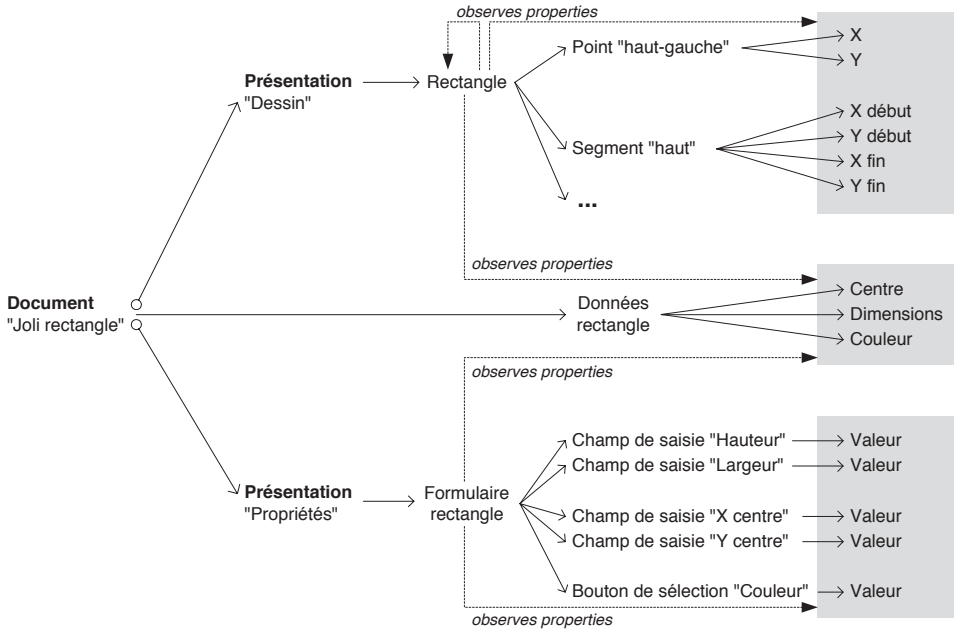


FIG. 4.10 – Exemple d'observations dans un document

Afin d'illustrer ce principe, reprenons l'exemple du "joli rectangle" et de ses deux présentations. Les ensembles des propriétés du document sont les feuilles de l'arbre, représentées à droite de la figure 4.10 ; les deux ensembles grisés sont ceux observés :

– Le rectangle composite observe :

1. Les propriétés x et y de ses 4 points, ainsi que les propriétés x et y des 2 extrémités début et fin des 4 segments, *via* les classes internes respectives PointObserver et

LineObserver. Cette observation permet de maintenir les contraintes de placement des éléments du rectangle pour que ce dernier demeure rectangulaire :

```
public class CompositeRectangle {
  //...
  protected class PointObserver
          implements GraphicObserver {
    public void newX(double x) { ... }
    public void newY(double y) { ... }
  //...
  protected class LineObserver
          implements GraphicObserver,
                  LineObserver {
    // interface GraphicObserver :
    public void newX(double x) { ... }
    public void newY(double y) { ... }
    //...
    // interface LineObserver :
    public void newXBegin(double x) { ... }
    public void newYBegin(double y) { ... }
    public void newXEnd(double x) { ... }
    public void newYEnd(double y) { ... }
  }
  //...
}
```

2. Il observe ses propres propriétés x, y, width, height et color afin de mettre à jour l'état du document lorsque la valeur de l'une d'entre-elle change.
3. Enfin, il observe les propriétés du document afin de mettre à jour son état lorsqu'elle la valeur de l'une d'entre-elle change.

– Ce mécanisme d'observation reste tout à fait similaire pour la présentation "Propriétés" qui affiche les propriétés du rectangle dans un formulaire.

Notons que les interfaces d'observation de propriétés sont le plus souvent implantées par des *classes internes* aux classes des composants observateurs. Ceci permet à un composant d'observer plusieurs composants en même temps (le rectangle précédents observe simultanément ses 8 composants internes).

Enfin, notons que, pour une classe de composants *C2* héritant d'une classe de composants *C1*, l'interface d'observation *C2*Observer *n'a pas* à hériter de la classe d'observation *C1*Observer (voir la figure 4.9, avec *C2* = Graphic et *C1* = Component). Cette non bijection entre les relations d'héritage des classes de composants et les interfaces d'observation associées est volontaire : elle permet à un composant de n'observer que la "facette" désirée du composant observable. Par exemple, une classe de composants ne souhaitant observer que les coordonnées de début et de fin d'un segment n'a pas à implanter l'interface GraphicObserver, mais uniquement l'interface LineObserver.

4.1.5.4 Notation

Afin de simplifier la représentation UML des composants DPI, nous adoptons une syntaxe caractérisant les relations entres les composants (figure 4.11) :

– Les "points de sortie" et les "points d'entrée" des composants sont représentés par des petits symboles, analogues aux "broches" des composants électroniques. Chaque point d'entrée représente une interface.
– Les points de sortie sont représentés en blanc et ceux d'entrée en noir.

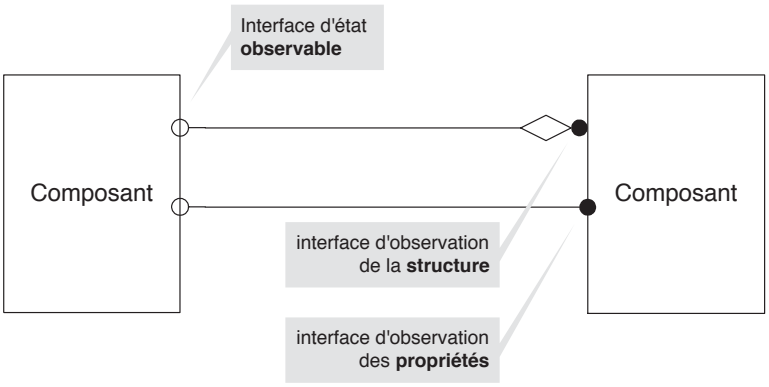


FIG. 4.11 – Notation du composant observateur et observable

– Le mécanisme d'observation est symbolisé par des broches en forme de disques : l'interface Observable est un disque blanc, et l'interface Observer un disque noir.
– L'observation d'un composant observable par un composant observateur est symbolisée par un trait reliant les deux broches.
– La broche d'observation de la structure laisse apparaître le symbole de composition utilisé dans les diagrammes de classes UML.

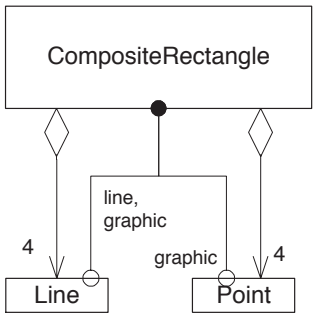


FIG. 4.12 – Notation composant pour le rectangle composite

La figure 4.12 donne l'exemple du composant rectangle composite : les 4 segments et les 4 points dont il est composé ont des propriétés qui sont observées par le rectangle conteneur.

4.2 L'interaction instrumentale

Nous abordons dans cette section la modélisation des instruments qui sont au centre de l'interaction dans le modèle DPI. Les principes essentiels sont ici abordés et seront affinés dans le chapitre suivant en considérant des instruments particuliers.

Le principe de l'instrument dans DPI est résumé dans la figure 4.13 et précise le schéma action / perception (section 3.2.4) :

1. L'utilisateur réalise son action intentionnelle en effectuant une action physique sur un ou plusieurs périphériques d'entrée.

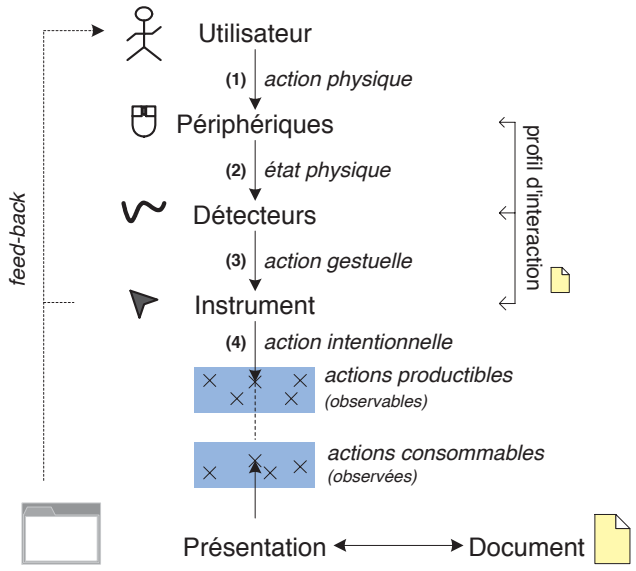


FIG. 4.13 – Principe de l'interaction instrumentale

2. Les périphériques mesurent l'action physique de l'utilisateur en fournissant l'état physique de leurs capteurs, tels que les déplacements X et Y et l'état des boutons d'une souris.
3. Les détecteurs de geste, définis dans la partie physique de l'instrument, analysent les états physiques des périphériques afin de détecter l'action gestuelle effectuée par l'utilisateur.
4. Cette action gestuelle est ensuite interprétée par l'instrument qui peut alors produire l'action intentionnelle sur le composant consommateur.

La possibilité pour l'instrument de produire une action sur le composant consommateur est basée sur le concept de compatibilité entre actions produites et actions consommées (section 3.3.1) : l'action peut effectivement être produite sur le composant consommateur quand le test de compatibilité entre l'action et l'ensemble des actions consommables retourne une action compatible avec l'action à produire. La relation de compatibilité entre deux actions est implantée *explicitement* au travers d'instruments appelés *adaptateurs*, lesquels seront abordés au chapitre suivant. En conséquence, le test de compatibilité sera réduit à un test d'égalité entre l'action à produire et les actions consommables.

L'adaptation en entrée des instruments est définie par le biais du "profil d'interaction". Il s'agit d'un document modifiable par l'utilisateur et dans lequel sont définies les règles de construction des instruments en fonction des périphériques d'entrée. Ces règles précisent comment sont construites les parties physiques des instruments de telle sorte que l'utilisateur puisse définir un style global d'interaction qui lui convienne.

4.2.1 Nature des actions

Le modèle DPI différencie deux types d'action : les actions gestuelles (ou gestes) et les actions intentionnelles (ou actions). Ces deux types ont des similitudes et des différences qui sont abordées ci-dessous.

4.2.1.1 Actions gestuelles

Les *gestes* définissent comment les utilisateurs peuvent provoquer des actions (intentionnelles) sur les objets d'intérêt. L'ensemble des gestes définit le *vocabulaire de l'interaction* du point de vue de l'utilisateur. Il faut entendre par geste toute action physique qu'un utilisateur peut effectuer pour interagir avec le système : il peut s'agir d'un geste au sens premier du terme, tel qu'un déplacement d'une souris ou l'appui sur un bouton, ou d'un geste au sens étendu, tel qu'une dictée vocale, un déplacement de la rétine, ou la production d'un son par exemple.

L'ensemble des gestes disponibles caractérise la richesse potentielle de l'interaction offerte par le système : une interaction potentiellement riche est synonyme d'un ensemble suffisamment étendu de gestes mis à disposition de l'utilisateur. Cet ensemble est dépendant des périphériques d'entrée utilisateur. Cependant, le principe d'adaptation en entrée des instruments doit permettre de ne pas compromettre l'étendue du vocabulaire quand ces périphériques d'entrée ne sont pas adaptés.

Cadoz (1994) a étudié la fonction du geste en tant que canal de communication entre l'homme et la machine. Il introduit la notion des gestes instrumentaux qu'il classe en trois catégories :

1. Geste d'excitation : Il détermine l'énergie fournie par l'utilisateur et qui sera transformée en information.
2. Geste de modulation : Il modifie les propriétés de l'instrument.
3. Geste de sélection : Il permet de sélectionner les objets d'intérêt.

Cette approche du geste a des similitudes avec l'interaction instrumentale (Beaudouin-Lafon, 1997). Nous retrouvons, dans notre modèle DPI, les trois types de gestes au sens de Cadoz (figure 4.13) :

1. L'action physique faite par l'utilisateur correspond au geste d'excitation. Cette entité ne sera pas explicitement définie dans notre modèle : nous formaliserons plutôt le résultat physique de l'action physique, résultat mesuré par les capteurs des périphériques d'entrée.
2. Le geste au sens DPI est détecté par la partie physique de l'instrument qui fournit une information en retour (feed-back). Dans l'exemple d'un outil, le geste de translation modifie la position du curseur représentant l'outil à l'écran et engendre, par exemple, une action de translation de la cible. Le geste DPI correspond à la notion du geste de modulation de Cadoz.
3. La sélection dans DPI ne concerne que le cas de la sélection d'un groupe d'objets sur lesquels on souhaite effectuer la même action. Dans le cas d'un objet unique, le pointage *direct* se substitue à la sélection. Le geste de sélection de Cadoz correspond à la désignation de la cible de l'action intentionnelle et, d'une certaine façon, peut-être considéré comme faisant partie de cette action.

Le tableau 4.1 précise les gestes définis dans la boîte à outils OpenDPI, lesquels sont classés en trois catégories : les gestes d'appui, les gestes de déplacement et les gestes de connexion. Chaque geste est caractérisé par une classe (deuxième colonne) et définit des périphériques typiques qui permettent une réalisation directe du geste (troisième colonne). Les gestes d'appui et de déplacement peuvent être capturés par les instruments. L'arrivée d'un périphérique est considéré comme un geste puisqu'il peut être associé à une intention de l'utilisateur. Un geste de connexion est capturé par le gestionnaire de périphériques, lequel peut décider de construire un nouvel instrument associé au nouveau périphérique en fonction de ce qui est précisé dans le profil de l'interaction.

Gestes d'appui		
Appui / relâchement	PushGesture	bouton d'une souris, touche d'un clavier
Clic simple, double, long	ClickGesture	bouton d'une souris
Pression	PressGesture	stylet d'une tablette graphique
Gestes de déplacement		
Translation	TranslateGesture	souris
Glissement	SlideGesture	molette d'une souris
Rotation	RotationGesture	souris d'une tablette graphique
Gestes de connexion		
Branchement / débranchement	PlugGesture	tout périphérique
Approche / éloignement	ProximityGesture	périphériques tablette graphique

TAB. 4.1 – Exemples de gestes

Dans DPI, les gestes sont des actions (re)produites par les *détecteurs* de geste : un détecteur mesure les états des périphériques associés et, quand le geste a effectivement lieu, produit ce geste vers son consommateur (l'instrument). Le geste suit ainsi le même schéma de production et de consommation introduit pour les actions. Nous soulignerons cependant quelques différences au fil des sections qui suivent.

Soulignons que l'adaptation en entrée des instruments consiste alors à définir les associations gestes / périphériques en fonction des périphériques disponibles. Par exemple, le geste de glissement réalisable simplement *via* la molette d'une souris peut être produit par le déplacement d'une souris selon l'axe (Oy) conjointement à l'appui sur un bouton.

4.2.1.2 Actions intentionnelles

Une action (intentionnelle) est l'action que l'utilisateur souhaite finalement appliquer à l'objet. Le principe du chaînage des actions est ainsi de reconstruire l'intention initiale de l'utilisateur, ce dernier l'ayant exprimée à partir de son vocabulaire gestuel et de sa connaissances des instruments qu'il manipule.

Si les gestes disponibles sont en nombre limité (lié à la variété des types de périphériques disponibles sur le marché), le nombre des actions définies par un système interactif peut devenir très important. Chaque domaine définit son propre jeu d'actions qui le caractérise, au même titre qu'il définira ces propres données (du domaine). Cependant, si certaines actions, dites actions spécialisées, appartiennent à un domaine particulier, un nombre non négligeable d'actions, participent à plusieurs domaine à la fois. Ces dernières actions sont qualifiées de *génériques* puisqu'elles peuvent s'appliquer à des objets de natures diverses.

La sémantique d'une action est classiquement définie par l'action elle-même. Cependant, nous considérons que la sémantique de l'action est *partiellement* définie par l'action elle-même, puis *complètement* définie par son consommateur. Par exemple, l'application d'une couleur (action "peindre") possède un sens précis seulement si l'on considère ses consommateurs potentiels : peindre une forme graphique vectorielle avec une couleur C peut consister à changer la couleur du fond du graphique, alors que peindre une image peut consister à transformer l'image de telle sorte que l'accent soit mis sur la couleur C. Les actions ont ainsi un caractère *polymorphe* au sens défini par Beaudouin-Lafon & Mackay (2000). Nous définissons le polymorphisme des actions comme suit :

> *Une action est polymorphe quand le résultat de son application prend une forme qui dépend de la nature de l'objet cible. Ainsi, l'action polymorphe a*

une forme imprécise définie par l'action elle-même, et des formes précises définies par les objets susceptibles de la consommer.

Le polymorphisme de l'action favorise la généricité des actions et donc la généricité des instruments.Par ailleurs, l'utilisation des actions induit une granularité fine de l'interaction (du niveau de l'instrument), là où cette granularité est forte pour le système OpenDoc (du niveau de l'éditeur).

Notons enfin qu'un geste peut tout à fait coïncider avec l'action intentionnelle. Par exemple, le geste de translation peut être utilisé directement afin de préciser une action de translation d'un objet à l'écran. Cette coïncidence peut être d'autant plus fine que le périphérique fournit la position absolue de la translation directement à l'écran : c'est le cas des écrans tactiles ou intégrant une tablette graphique transparente. Ainsi, la plupart des gestes définissent une action équivalente qu'ils sont susceptibles de générer directement.

4.2.2 Production et consommation

La figure 4.13 illustre le chaînage des actions depuis la source, l'utilisateur, jusqu'à l'objet ciblé, la présentation d'un document. Ce chaînage suit le modèle producteur / consommateur : les actions sont produites par les producteurs et consommées par les consommateurs.

4.2.2.1 Producteur et consommateur

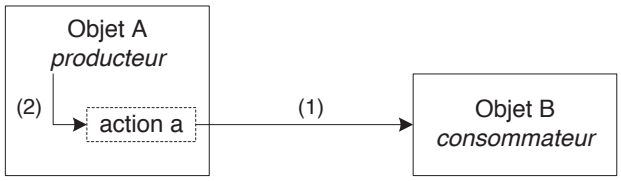


FIG. 4.14 – Principe du producteur et du consommateur d'une action

Le principe du producteur et du consommateur suit deux étapes (figure 4.14) :

1. L'objet A, producteur de l'action, interroge l'objet B pour savoir s'il sait consommer l'action. Dans le cas d'une réponse positive, il inscrit B comme étant le consommateur de l'action[7].
2. L'objet A envoie l'action à B en précisant les paramètres de l'action.

La réalisation du modèle producteur / consommateur suit le modèle donné dans la figure 4.15. A chaque action sont associées une interface de production et une interface de consommation qui permettent respectivement de caractériser son producteur et son consommateur. Ces interfaces suivent une convention de nommage qu'il est impératif de respecter. Par exemple, une action de translation, définie par la classe Translate, définira nécessairement les interfaces TranslateProducer et TranslateConsumer.

L'interrogation de l'objet B par l'objet A (point 1) est réalisée en inspectant les interfaces de consommation implantées par l'objet cible : si cet objet implante l'interface de consommation associée à l'action, l'interrogation retourne une réponse positive. L'inspection des interfaces de production d'un composant est, quant à elle, purement informative et permet de préciser quelles actions peuvent être produites par un composant.

[7]La procédure d'inscription est justifiée par la durée éventuelle d'une action, comme l'illustre le cycle de production des actions (section 4.2.3).

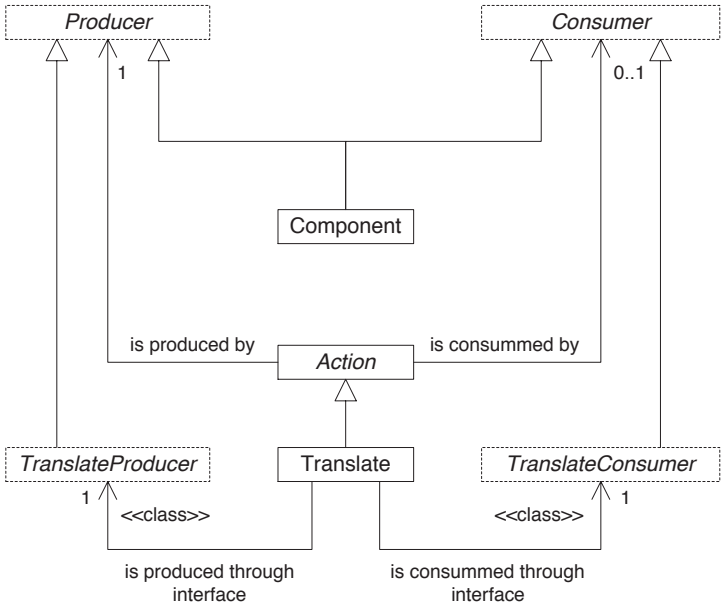


FIG. 4.15 – Modèle producteur / consommateur

4.2.2.2 Interfaces de consommation et de production

L'*interface de consommation* définit les méthodes que doivent implanter tout consommateur potentiel, ces méthodes étant invoquées par le producteur dès lors que l'action est produite. Cet ensemble de méthodes est appelé *commande*, par analogie avec la notion de commande que peuvent invoquer les utilisateurs *via* leurs interfaces. En particulier, l'un des éléments d'une commande est la méthode dite d'*exécution* dont le nom est constitué du nom de l'action préfixé par le verbe "*do*", et dont les arguments correspondent aux *propriétés de l'action*.

Considérons le cas de l'action Translate définie comme suit :

```
public class Translate extends Action {
   public double getDX() { return dx ; }
   public void setDX(double dx) { this.dx = dx ;}

   public double getDY(){ return dy ;}
   public void setDY(double dy) { this.dy = dy ;}

   protected double dx ;
   protected double dy ;
}
```

Cette action possède les deux propriétés dX et dY définies au travers de leurs accesseurs. L'interface de consommation doit alors définir la méthode d'exécution comme suit :

```
public interface TranslateConsumer extends Consumer {
   public void doTranslate(double dx, double dy) ;
}
```

Nous avons utilisé cette notation pour la méthode d'exécution dans un unique soucis d'homogénéité avec la notation propre aux méthodes. Ainsi, pour qu'une méthode puisse être

réifiée en action, il suffit de préfixer la méthode par le verbe “do” et de définir la classe d’action associée[8].

Les *interfaces de production* ont un but le plus souvent uniquement informatif : elles sont souvent vides. Pour certaines actions particulières, le producteur attend une réponse du consommateur une fois l’action produite : le rôle de l’interface de production est de préciser cette attente. Des exemples d’interface de production non vides sont donnés en annexe (section A.6).

4.2.2.3 Notation

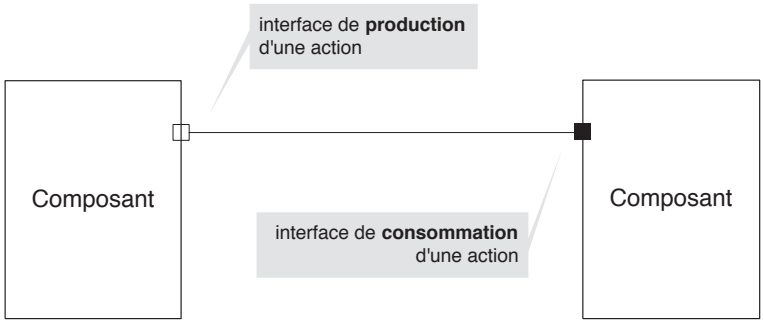


FIG. 4.16 – Notation des actions pour les composants

Le principe de production et de consommation fournissant une relation (en général momentanée) entre deux composants, nous complétons la représentation “composant” (figure 4.16) :

– Le symbole carré représente une broche du composant relative à une action.
– Le carré blanc représente l’interface de production (interface “de sortie”).
– Le carré noir représente l’interface de consommation (interface “d’entrée”).
– La connexion entre les deux broches indique la possibilité pour un composant producteur de produire son action sur le composant consommateur.

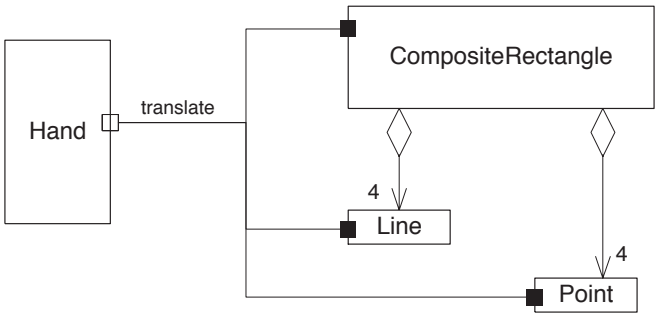


FIG. 4.17 – Exemple de la translation du rectangle composite

L’exemple de la figure 4.17 reprend le cas du rectangle composite et illustre la généricité de l’action de translation : elle peut s’appliquer au rectangle lui-même, à l’une de ses quatre

[8]Ceci peut être notamment réalisé d’une manière automatisée en utilisant un compilateur dédié ou, mieux encore, en étendant la grammaire du langage de telle sorte à rendre *explicite* la notion d’action (et de propriété).

lignes ou l'un de ses quatre points. Lorsqu'une telle action se produit sur une ligne ou un point, le rectangle est re-dimensionné et doit maintenir la cohérence de sa forme rectangulaire par le mécanisme d'observation déjà invoqué.

4.2.3 Cycle de production d'une action

4.2.3.1 Action intentionnelle

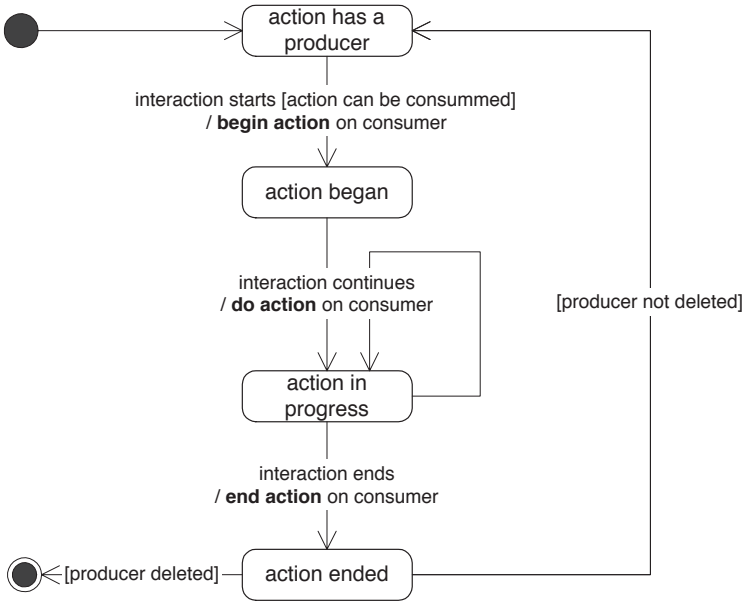


FIG. 4.18 – Cycle de production d'une action

La production d'une action suit un cycle immuable représenté figure 4.18 :

1. L'action est initialement construite en étant associée à un producteur donné.
2. Lorsque l'interaction commence sur un consommateur compatible avec l'action à produire, l'action passe dans l'état "début".
3. Tant que l'interaction (*i.e.* le geste) continue, l'action est dans l'état "en cours" (d'exécution).
4. Dès lors que l'interaction s'arrête, l'action passe dans l'état "fin".

Ce cycle fournit les différents états possibles pour une action. Lorsque l'action est "en cours", la méthode d'exécution ("doAction") est invoquée à chaque étape de l'interaction. Afin que le consommateur puisse prendre en compte les états "début" et "fin", l'interface de consommation définit deux méthodes préfixées respectivement par "begin" et "end". Par exemple, l'interface de consommation de l'action de translation s'écrit maintenant :

```
public interface TranslateConsumer extends Consumer {
   public void beginTranslate(TranslateProducer producer) ;
   public void doTranslate(double x, double y) ;
   public void endTranslate(TranslateProducer producer) ;
}
```

Ces deux nouvelles méthodes de notification de début et de fin seront nécessaires pour rendre possible l'annulation des actions (section A.2) ainsi que le verrouillage de propriétés (section 5.2).

Il est important de noter que la production d'une action ne préjuge pas du geste qui l'engendre. Par exemple, l'action "peindre" n'effectue pas nécessairement une seule invocation de la méthode d'exécution encadrée par les notifications de début et de fin : le fait de peindre peut passer par plusieurs étapes consistant à définir progressivement la couleur de la peinture.

Par ailleurs, il n'est pas toujours aisé de déterminer la fin d'une action. Par exemple, l'action "scale", qui consiste à modifier l'échelle d'un graphique, peut être produite en utilisant un geste de glissement associé à la molette d'une souris ; le glissement n'est alors pas encadré par une interaction spécifique délimitant son début ni surtout sa fin. Dans ce cas particulier, l'instrument producteur peut associer à l'action un délai de fin automatique : lorsque la méthode d'exécution de l'action n'est pas invoquée pendant ce délai, l'action est considérée comme terminée. Il peut également décider qu'une telle action est considérée comme terminée lorsqu'une autre action survient.

4.2.3.2 Geste

Les états de début et de fin étant explicitement définis dans le modèle afin de pouvoir gérer l'annulation ainsi que le verrouillage de propriétés, le cycle de production des gestes diffère de celui des actions pour les raisons suivantes :

1. Les gestes étant immuables, ils ne sauraient être annulés. Seule l'action engendrée peut l'être.
2. Les gestes ne sont pas concernés par le verrouillage de propriétés. Seule les actions modifient les propriétés des objets ciblés.
3. Les gestes de déplacement ne sauraient définir une fin explicite.

Le but des détecteurs de geste est de ramener la description du geste physique dans l'espace logique, et d'en fournir les mesures absolues dans cet espace. L'interface de consommation du geste de translation est ainsi codée :

```
public interface TranslateGestureConsumer extends Consumer {
  public void doTranslateGesture(
    double absoluteX, double absoluteY,
    double relativeX, double relativeY) ;
}
```

Les coordonnées relatives sont fournies dans un seul but de simplifier le codage de l'interaction au niveau des instruments, le geste de translation devant pouvoir être transformé en une action de translation (à coordonnées relatives).

4.2.4 Partie physique des instruments

La partie physique d'un instrument est composée d'un ou plusieurs périphériques et de détecteurs de geste associés. Nous précisons ici ces deux éléments.

4.2.4.1 Introduction sur les périphériques d'entrée

Dans (Beaudoux & Beaudouin-Lafon, 2001), nous avons distingué trois types de périphérique : le périphérique physique, le périphérique logique et le périphérique logique simulé.

Les périphériques physiques correspondent aux périphériques présents dans l'espace physique de l'utilisateur et déclarés dans la partie physique des instruments.

Les périphériques logiques sont construits par un "assemblage" de périphériques physiques dans l'espace logique. Par exemple, une souris logique à trois boutons peut être construite en couplant une souris physique à un bouton et deux touches d'une clavier. Dans le modèle DPI, les périphériques logiques ne sont pas explicitement définis mais sont en fait construits *implicitement* par l'instrument.

Les périphériques logiques simulés simulent les périphériques physiques dans l'espace logique. Un clavier représenté à l'écran est un périphérique logique simulé. Par exemple, le Xerox Star utilise un clavier simulé afin de gérer des saisies spécifiques, telle que la saisie de formules mathématiques (Johnson et al., 1989). La construction de périphériques simulés était possible avec la boîte à outil X_{TV} (Beaudouin-Lafon et al., 1990). De tels périphériques peuvent également être utilisés pour des besoins spécialisés, comme la souris simulée du système "metamouse" (Maulsby et al., 1989). Dans DPI, ces périphériques seront vus comme des instruments indirects (section 4.2.5.3).

4.2.4.2 Périphériques d'entrée utilisateur

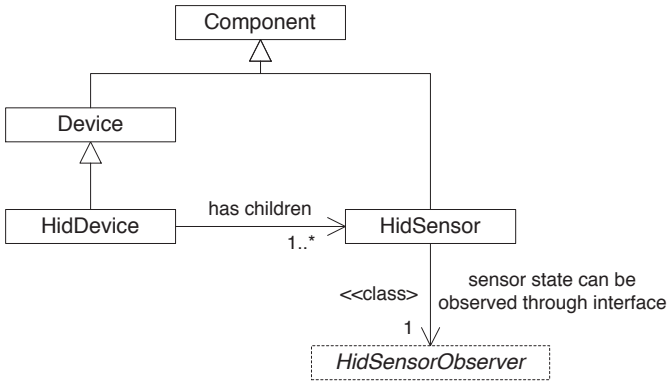


FIG. 4.19 – Décomposition des périphériques en capteurs

Les périphériques d'entrée utilisateur ("Human Input Device") sont composés de capteurs élémentaires (classes HidDevice et HidSensor). Un périphérique et ses capteurs sont considérés comme des composants DPI, les capteurs représentant les composants fils du périphérique associé (figure 4.19), et le périphérique un composant fils du gestionnaire de périphériques (non représenté). Cette décomposition en capteur élémentaires permet de satisfaire l'adaptabilité en entrée des instruments : un instrument construit sa partie physique en choisissant les capteurs adéquats, lesquels sont disponible sur un ou plusieurs périphériques (construction implicite d'un périphérique logique). Elle s'inspire de la représentation des périphériques d'entrée utilisateur USB[9] (Collectif, 2001).

Les capteurs permettent de caractériser tous les périphériques d'entrée possibles. Chaque capteur définit pour cela :

[9]Universal Serial Bus

– Son état propre sous la forme d'un entier signé. Il s'agit d'une propriété qui peut être écoutée au travers de l'interface d'observation HidSensorObserver. La résolution de l'état propre est toujours égale à l'unité.
– Un code d'identification du capteur. Par exemple, le code KEY_1 caractérise le capteur associé à la touche '1' d'un clavier, et le code BTN_LEFT le bouton gauche d'une souris (ou d'un autre dispositif de pointage).
– Un code d'identification de la nature de l'information délivrée par la propriété "état". Par exemple, le code EV_KEY caractérise un état booléen (d'une touche d'un clavier ou d'un bouton de souris par exemple), le code EV_REL indique un état représentant un déplacement relatif et EV_ABS une position absolue.
– Les valeurs minimale et maximale de l'état. Elles sont définies typiquement pour les capteurs fournissant une donnée absolue.

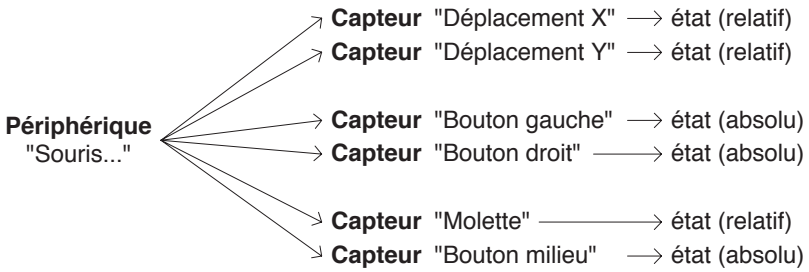


FIG. 4.20 – Exemple de la souris

La figure 4.20 précise les différents capteurs d'une souris : les capteurs de déplacement fournissent des positions relatives (capteur X, Y et molette) et les boutons fournissent les états booléens absolus.

4.2.4.3 Détecteurs de gestes

Comme nous l'avons déjà précisé, les gestes sont mesurés par les détecteurs, ces derniers produisant alors le geste vers l'instrument associé.

A chaque geste est associé un détecteur qui en est le producteur. Ainsi, le geste de translation (classe TranslateGesture) définit ses interfaces de production et de consommation (interfaces TranslateGestureProducer et TranslateGestureConsumer) ainsi que son détecteur (classe TranslateDetector qui implante l'interface TranslateGestureProducer).

La figure 4.21 donne l'exemple de la production de l'action de translation par l'instrument "main"[10]. La production de la translation s'effectue par la détection du geste de translation, lequel doit être encadré par un geste d'appui / relâchement : quand ces gestes sont exécutés (invocation des méthodes doPushGesture et doTranslateGesture de la classe Hand), le curseur est positionné à la position absolue fournie par le détecteur, et la main effectue l'action de translation si une cible compatible avec l'action est trouvée. Les deux détecteurs observent l'état des capteurs "position relative X", "position relative Y" et "bouton gauche", lesquels sont typiquement fournis par un unique périphérique de pointage (non représenté).

La boîte à outils ICON de Dragicevic & Fekete (2001) permet de configurer finement la façon dont les dispositifs d'entrée peuvent être utilisés dans diverses interactions. Sa richesse

[10] L'instrument "main" représente l'outil générique déjà présent sur les systèmes actuels et qui réalise les actions classiques de sélection, de déplacement et de glisser-déposer.

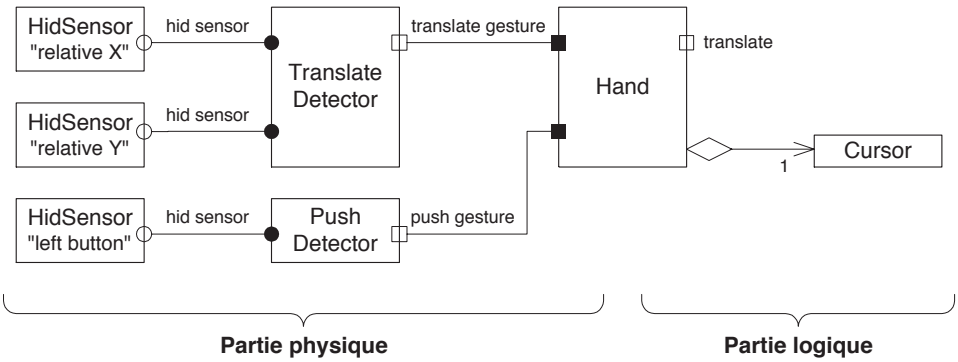
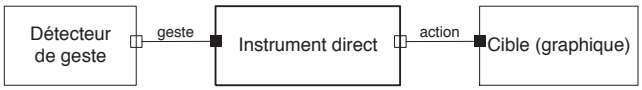


FIG. 4.21 – Détection et production de la translation

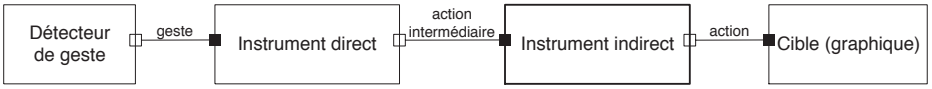
permet notamment la construction d'applications multi-modales. Du point de vue des instruments, elle permet de spécifier (graphiquement) la constitution de leur partie physique. Le pouvoir d'expression offert par ICON est important mais, en contre partie, nécessite une bonne connaissance de l'approche (niveau expert). La différence majeure entre ICON et notre approche concerne les gestes et les actions : ces deux notions sont absentes dans ICON et centrales dans DPI. Ces deux approches ne sont cependant pas incompatibles : le modèle DPI est centré le principe de production et de consommation des actions, lequel permet de traiter l'interaction indépendamment des objets cibles, et ICON se concentre davantage sur la spécification de la partie physique des instruments.

4.2.5 Partie logique des instruments

4.2.5.1 Directivité des instruments



(a) Action directe



(b) Action indirecte

FIG. 4.22 – Directivité des actions

Nous avons, dans la description de DPI, différencié les instruments directs des d'instruments indirects. La directivité d'un instrument précise la notion de *distance* entre l'intention initiale et sa transformation en action finale (Norman & Draper, 1986, chapitre 5). La figure 4.22 illustre que la longueur de la chaîne "geste → action → propriétés" est représentative de cette distance. Cette distance est faible lorsque l'action intentionnelle est directement appliquée par l'utilisateur *via* l'instrument (a). Par exemple, l'utilisation de l'instrument

“pinceau” induit une distance faible puisque l’action de “peindre” est directement appliquée sur la cible. Cette distance devient plus grande dès lors qu’un instrument indirect s’intercale dans la chaîne (b). Par exemple, l’utilisation d’un widget de type bouton induit une distance élevée puisque la manipulation du bouton nécessite un second instrument.

Ainsi, si des actions intermédiaires s’intercalent dans la chaîne de production de l’action intentionnelle, la distance, dite distance logique, s’en trouve alors accrue. Nous définissons la *distance logique* ainsi :

> *La* distance logique *entre un instrument et l’objet ciblé est le nombre d’actions entrant en jeu dans la chaîne de production de l’action intentionnelle considérée. Ce nombre exclut le geste, première action de la chaîne, ainsi que l’action intentionnelle elle-même, dernière action de la chaîne.*
>
> *Une distance logique $d_L = 0$ signifie que l’action est produite directement. Une distance logique $d_L > 0$ signifie que l’action intentionnelle est produite indirectement, via une ou plusieurs actions intermédiaires.*

Cette notion de distance logique complète la notion de distance géométrique qui caractérise la distance réelle entre l’instrument et la cible. Par exemple, l’outil “pinceau” applique l’action de “peindre” sur la cible qui se situe précisément dessous, la distance géométrique étant ainsi nulle. Par contre, l’instrument “bouton” applique une action sur une cible préalablement sélectionnée et éloignée du bouton, la distance géométrique étant alors non nulle. Nous définissons la *distance géométrique* ainsi :

> *La distance géométrique entre un instrument et l’objet ciblé est la longueur du plus petit segment reliant un point de l’instrument et un point de la cible.*
>
> *Une distance géométrique $d_G = 0$ signifie que l’instrument peut agir directement sur la cible. Une distance géométrique $d_G > 0$ signifie que l’instrument ne peut agir qu’indirectement sur la cible.*

Nous pouvons maintenant donner une définition formelle des instruments directs et indirects :

> *Un instrument est dit* direct*, pour une action donnée, s’il induit une distance géométrique d_G* et *une distance logique d_L tel que $(d_G = 0) \wedge (d_L = 0)$ pour la production de cette action. Un instrument est direct s’il est direct pour toutes les actions qu’il sait produire.*
>
> *Un instrument est dit* indirect*, pour une action donnée, s’il induit une distance géométrique d_G* et *une distance logique d_L* tel que *$(d_G > 0) \wedge (d_L > 0)$ pour la production de cette action. Un instrument est indirect s’il est indirect pour toutes les actions qu’il sait produire.*

Dans ces définitions, la distance logique est couplée à la distance physique du point de vue de l’indirection puisque nous avions $(d_G = 0) \wedge (d_L = 0)$ ou $(d_G > 0) \wedge (d_L > 0)$. Cependant, ces distances ne sont pas nécessairement couplées. Nous pouvons en effet avoir $(d_G = 0) \wedge (d_L > 0)$, cas qui correspond à un chaînage d’actions ayant lieu à l’endroit de l’instrument, ce qui est par exemple le cas pour le “toolglasses”. Par contre, le cas $(d_G > 0) \wedge (d_L = 0)$ est incohérent puisqu’il n’est pas possible d’appliquer directement une action depuis un endroit donné vers un autre endroit[11]. Nous introduisons le cas des instruments *semi-directs* pour satisfaire la troisième possibilité :

[11] La cas de l’instrument “raccourci clavier” induit $d_G = 0$ puisque, bien qu’invisible, un tel instrument est présent sur tout l’espace de travail.

Un instrument est dit semi-direct, *pour une action donnée, s'il induit une distance géométrique* d_G et *une distance logique* d_L *tel que* $(d_G = 0) \wedge (d_L > 0)$ *pour la production de cette action. Un instrument est semi-direct s'il est semi-direct pour toutes les actions qu'il sait produire.*

Les "toolglass" et les adaptateurs sont des exemples d'instruments semi-directs, lesquels seront abordés dans le chapitre A.

Notons enfin qu'un instrument direct ou semi-direct ($d_G = 0$) réalise un piqué ("picking") afin de préciser sa cible, alors qu'un instrument indirect ($d_G > 0$) utilise un mécanisme de sélection afin de préciser sa cible.

4.2.5.2 Les outils

Les instruments directs représentent la métaphore des "outils". La transmission de l'action de l'outil producteur vers le composant consommateur s'effectue à la suite d'un "piqué" ("*picking*") qui retourne le composant se situant sous l'outil *et* implantant l'interface de consommation de l'action.

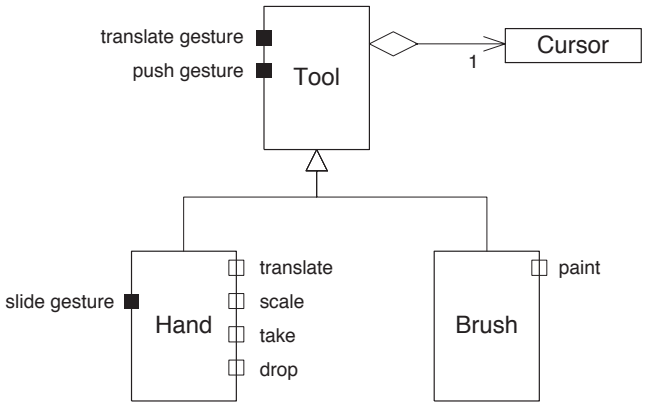


FIG. 4.23 – Outil "Main" et "Pinceau"

La figure 4.23 représente les parties logiques des deux outils "main" (classe Hand) et "pinceau" (classe Brush), lesquels héritent du composant abstrait "outil" (classe Tool). La partie logique contient la représentation de l'instrument réalisée *via* un curseur[12] et définit les différentes actions productibles.

La figure 4.24 illustre le principe de généricité pour les outils "main et "pinceau" : les actions de translation, de mise à l'échelle et de coloriage sont consommables par toute instance d'une classe héritant de graphique, permettant alors au rectangle composite d'être éditable en manipulant l'un de ses huit éléments constitutifs. La généricité est là naturellement induite par l'héritage de la classe des graphiques. Cependant, si l'héritage induit aisément la généricité, il n'en demeure pas une relation nécessaire (mais simplement suffisante). Notons que cette généricité induite très rapidement par l'héritage peut devenir indésirable dans certaines circonstances. Par exemple, le titre d'un bouton ne doit pas, en général, pouvoir être déplacé dans déplacer le bouton lui-même. Cet aspect pourra être traité facilement en utilisant le verrouillage de propriété abordé au chapitre 5.

[12] Dans la pratique, le curseur peut être complété pour afficher diverses informations, tel que l'état de l'instrument (la couleur pour le pinceau par exemple), le propriétaire de l'instrument, etc.

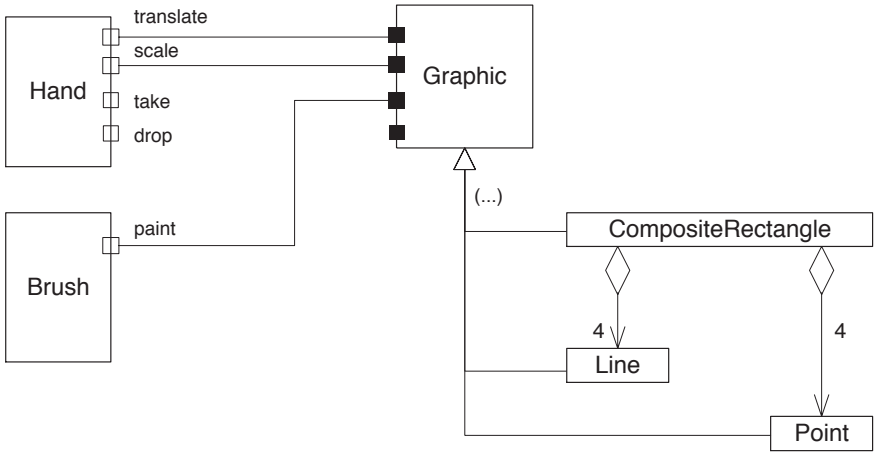


FIG. 4.24 – Généricité des outils “Main” et “Pinceau”

4.2.5.3 Les appareils

Un appareil est un instrument qui induit une distance logique $d_L > 0$: il produit l'action intentionnelle en consommant une action intermédiaire, cette dernière générant une indirection. Par exemple, un magnétoscope est un appareil auquel l'utilisateur *délègue* les actions de lire et d'enregistrer une séquence vidéo. En vertu des définitions faites plus haut, un appareil est un instrument soit indirect, soit semi-direct.

Un exemple d'instrument indirect est le bouton. Un bouton est un appareil capable de produire une action qui lui est associée sur une cible donnée. Son action est en général fixée une fois pour toute et la cible est spécifiée par un mécanisme de sélection. Lorsque l'action “d'appui” (classe Push, équivalente du geste PushGesture), générée par exemple par l'outil “main”, est consommée par le bouton, ce dernier produit l'action sur la cible spécifiée. L'action “d'appui” est l'action intermédiaire cause de l'indirection.

Un exemple d'instrument semi-direct est le bouton transparent gérant l'interaction “clic-au-travers”. Son principe de fonctionnement est identique à celui du bouton opaque usuel mais la spécification de la cible se fait au moment de l'interaction d'application de l'action par le mécanisme du clic-au-travers. Ce type de bouton est généralement transporté par une palette “toolglass”, elle-même déplacée *via* la main non dominante (interaction bimanuelle).

Un appareil peut être composé d'autres appareils élémentaires, ce qui justifie le choix du terme métaphorique “appareil” : dans notre vie courante, nous utilisons des outils pour les actions pouvant être traitées directement, et des appareils pour des actions ne pouvant être réalisées qu'indirectement. La description précédente de l'appareil correspond alors à l'appareil élémentaire, l'appareil parent n'étant alors qu'un “conteneur” d'appareils élémentaires (ce conteneur n'intervient pas dans le chaînage des actions). Cette approche métaphorique de l'appareil suggère l'idée de périphérique simulé mentionnée plus haut : le widget “bouton” peut être vu comme un bouton physique (d'une souris, d'un clavier ou de tout autre dispositif électronique), ou le lecteur de CD audio est un appareil qui existe aussi bien dans l'espace logique que dans l'espace physique.

La notion d'appareil coïncide, en première approximation, avec la notion de “widget” : un bouton et un “toolglass” sont deux exemples de widgets. Cependant les widgets, en particulier ceux à gros grain, ne sont pas nécessairement des appareils au sens de notre modèle. L'étude de cas proposée dans la section 3.4 illustre ce propos : l'affichage arborescent d'une

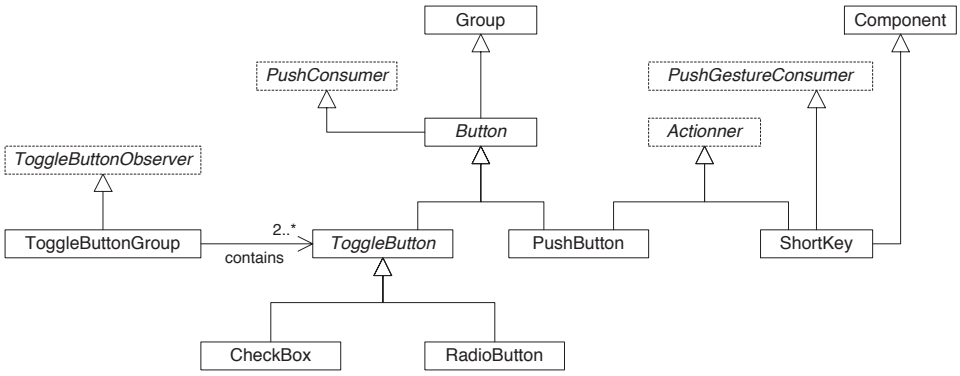


FIG. 4.25 – Widgets de type “bouton”

fenêtre du “finder” est une présentation et n’est donc pas un appareil. Si une telle application avait été étudiée sous l’angle des widgets, cet affichage aurait été pris en charge par un élément de type “TreeWidget” : il gérerait l’affichage, le traitement des données et l’interaction (navigation et édition). Dans notre approche, la présentation arborescente n’est pas un appareil *mais* contient, en plus des autres présentations imbriquées, des boutons triangulaires qui sont des appareils.

En conséquence, les seuls appareils qui coïncident avec la notion de widget sont les boutons. De tels appareils sont appelés *appareils élémentaires*. La figure 4.25 recense les principaux appareils élémentaires que sont les boutons (classe Button) et les raccourcis clavier (classe ShortKey). Remarquons que, en toute rigueur, une case à cocher n’est pas un appareil puisque son rôle n’est pas d’engendrer une action, mais de représenter l’état d’une donnée booléenne. Cependant, étant un cas particulier de bouton à bascule, il est considéré comme appareil. Les raccourcis clavier sont un cas particulier d’instrument avec une partie logique qui n’intègre pas spécifiquement de représentation. Ils ont une fonction analogue aux boutons : permettre la production d’une action sur une cible pré-sélectionnée (classe Actionner). Cependant, il s’agit d’instruments directs puisque $(d_L = 0) \wedge (d_G = 0)$.

Nous complétons maintenant la notion d’appareil par celle d’*appareil conteneur*. Une présentation a la capacité de jouer le rôle d’un appareil conteneur, *i.e.* d’un appareil construit (en autre) à partir d’appareils élémentaires. La présentation arborescente est un exemple d’appareil conteneur. L’exemple du lecteur de CD illustre également la notion d’appareil conteneur : le lecteur est un “boîtier” contenant (entre autre) différents boutons qui permettent de lancer différentes actions, tel que la lecture d’un morceau ou le passage au morceau suivant. La caractéristique essentielle d’un appareil conteneur est qu’il ne produit pas d’actions : il délègue cet aspect aux appareils élémentaires qu’il contient .

4.3 Conclusion

Le modèle DPI s’articule autour d’un modèle de composant générique. Un composant DPI est constitué de propriétés et de composants fils. Les propriétés d’un composant ainsi que sa structure hiérarchique peuvent être observées par d’autres composants *via* des interfaces d’observation. Les composants interagissent entre-eux par l’intermédiaire des actions. Chaque action peut être produite par un composant au travers d’une interface de production et peut être consommée par un composant au travers d’une interface de consommation. Un tel modèle permet de décrire le comportement de tous les composants intervenants dans notre modèle.

Composant	Interface d'observation	Interface de consommation	Interface de production
Périphérique	observabilité des capteurs	–	–
Détecteur de geste	observation des capteurs	–	production de gestes
Instrument direct	–	consommation de gestes	production d'actions
Instrument indirect	–	consommation d'actions	production d'actions
Représentation	observation des noeuds du document	consommation d'actions	–
Présentation	observation de la structure du document	–	–
Nœud	observabilité des propriétés	–	–
Document	observabilité de la structure	–	–

TAB. 4.2 – Synthèse des composants de DPI

Le tableau 4.2 synthétise comment le modèle de composant est utilisé pour chaque composant de DPI. Il illustre que chaque catégorie de composant propose une utilisation spécifique de ce modèle. Par exemple, un composant implantant une interface de consommation de gestes et une interface de production d'actions est typiquement un instrument direct.

La boîte à outil OpenDPI définit l'ensemble des aspects décrits dans ce chapitre. Elle compte à ce jour environ 250 classes, soit plus de 18000 lignes de code source. Ces chiffres assez importants sont principalement dus au fait que OpenDPI ne s'appuie que sur deux librairies externes : l'API[13] du langage Java, et la librairie Piccolo du HCIL (Bederson et al., 2000). La rapidité d'exécution n'est pas très élevée : elle est principalement induite par la lenteur de l'affichage du graphe de scène géré par Java[14]. Par ailleurs, l'emprunte mémoire peut s'avérer relativement élevée puisque chaque éléments de l'interface est un composant générique, lequel définit un ensemble non négligeable de données. Pour plus d'informations sur la réalisation de composants DPI complexe, le lecteur peut consulter les exemples fournis en annexe A.

[13] Application Program Interface

[14] Un test assez simple permet de justifier cette assertion : l'utilisation d'une résolution de 800 x 600 pixels implique un affichage fluide pour des situations assez complexes (comme celle de la figure A.3 on page 210), alors qu'une résolution 1600 x 1200 induit un affichage très saccadé.

Chapitre 5

Interaction bimanuelle - Collaboration locale

Contents

Dans ce chapitre, nous introduisons le problème de la collaboration locale conjointement à l'interaction bimanuelle (section 5.1). La section 5.2 précise la notion d'interaction consistante et le moyen proposé, appelé verrouillage de propriété, pour garantir la cohérence de l'interaction. La section 5.3 explique l'induction du verrouillage des propriétés et le coût engendré par cet aspect. La section 5.4 pose le problème de la simultanéité qui peut survenir lors de la demande de verrou, puis en propose une solution. La section 5.5 aborde un aspect particulier de la collaboration locale, le partage d'instrument, qui est dédié à l'apprentissage d'un instrument. La section 5.6 précise le problème de l'annulation et la ré-exécution dans un contexte multi-utilisateurs et en apporte une solution. Enfin, la section 5.7 effectue une synthèse du chapitre et met en perspective les évolutions possibles du modèle DPI de manière en prendre en compte les problèmes liés au verrouillage.

5.1 Introduction

L'utilisation de l'outil informatique dans un environnement collaboratif suggère bien souvent la notion de collaboration à distance. Cependant, la collaboration locale, qui consiste à coopérer *via* un unique écran, offre un intérêt significatif (Stewart et al., 1999). Nous souhaitons par conséquent que le modèle DPI intègre un modèle de partage local d'un écran.

Dans le cadre du "*Single Display Groupware*" (SDG), les utilisateurs manipulent typiquement leur propre souris : il existe donc plusieurs pointeurs sur l'écran partagé. Il peut alors arriver des situations dans lesquelles ces pointeurs manipulent un objet au même instant, ce qui doit être rendu possible ou impossible par le système en fonction du contexte. Par exemple, un rectangle composite peut être dimensionné en déplaçant simultanément deux de ses coins opposés mais ne peut l'être en déplaçant simultanément deux coins non opposés puisqu'il y a, dans ce cas, déplacement concurrent d'un segment commun. Cette situation peut également se produire dans le cadre de l'interaction bimanuelle pour laquelle typiquement deux périphériques de pointage sont manipulés par l'utilisateur. La simultanéité des actions est ainsi le point commun entre le SDG et l'interaction bimanuelle.

A notre connaissance, il n'a pas été défini d'approche générale de la concurrence d'accès ni pour le SDG, ni pour l'interaction bimanuelle, malgré le fort intérêt de ces deux domaines de l'interaction homme-machine. Notre modèle comble cette lacune en proposant une approche basée sur un verrouillage à granularité faible des objets "partagés". Nous présentons l'affinement du modèle DPI qui permet de prendre en compte la gestion des verrous.

5.2 Cohérence de l'interaction

5.2.1 Problème

Une gestion correcte de la concurrence en collaboration à distance permet d'éviter toute incohérence entre les différentes répliques des objets partagés[1]. Quand la collaboration reste locale, cette cohérence est nécessairement réalisée puisque l'écran est partagé ; la gestion de la concurrence peut alors sembler inutile. Cependant, il peut arriver des situations dans lesquelles l'interaction ne fournit pas un résultat conforme à celui attendu : nous parlons dans ce cas de d'*incohérence de l'interaction*. De telles situations peuvent être évitées par une gestion de la *concurrence d'accès locale*.

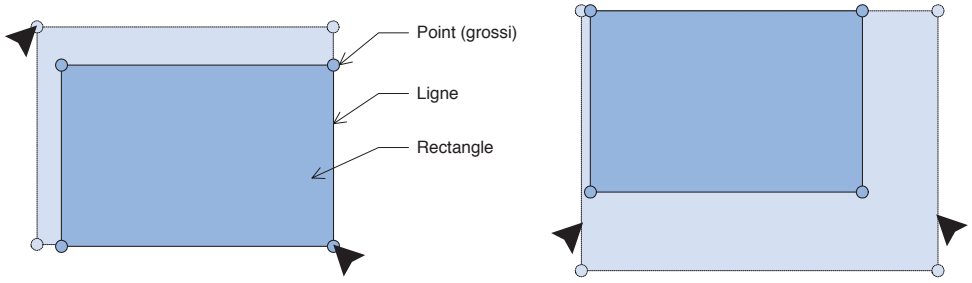
La notion de cohérence de l'interaction suggère celle de la préservation de l'intention de l'utilisateur (Prakash, 1999; Sun et al., 1998). Les intentions des utilisateurs peuvent ne

[1] Nous reviendrons sur la notion de réplique d'un objet partagé au chapitre 6.

pas être préservées, lors d'une collaboration temps réel à distance, quand le temps de transfert des messages sur le réseau devient supérieur au laps de temps séparant les actions (qualifiées alors de "simultanées")[2]. Il s'agit d'un problème cependant différent du problème de la cohérence des objets partagés (section 2.2.2.1) : l'objet partagé est dans un état consistant *mais* cet état n'est pas nécessairement celui voulu par l'utilisateur. Remarquons cependant qu'il subsiste une différence entre les notions respectives de préservation de l'intention et de cohérence de l'interaction : dans le premier cas, l'état final de l'objet n'est pas celui escompté par l'utilisateur alors que, dans le deuxième cas, l'intention est préservée mais n'engendre pas un rendu correct lors de l'interaction. De plus, la préservation de l'intention concerne la collaboration distante tandis que la cohérence de l'interaction concerne la collaboration locale.

5.2.2 Exemples

Un premier exemple d'interaction inconsistante concerne la translation d'un graphique : lorsque deux actions de translation $(dx_1,\ dy_1)$ et $(dx_2,\ dy_2)$ sont produites simultanément sur un graphique par deux outils "main", sa position s'incrémente de $(dx_1 + dx_2,\ dy_1 + dy_2)$. La position résultante du graphique n'est alors pas conforme aux positions respectives des deux mains. Dans ce cas, les deux translations sont équivalentes à une translation "glissante" qui a une sémantique différente (section 5.2.5).



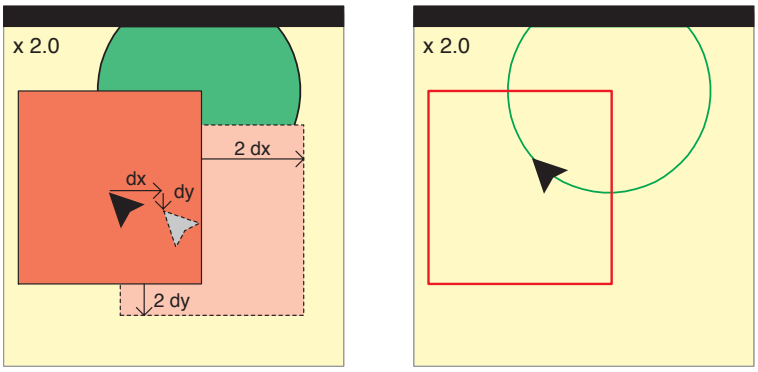
(a) Dimensionnement consistant (b) Dimensionnement inconsistant

FIG. 5.1 – Cohérence d'un dimensionnement

La figure 5.1 représente l'exemple plus subtil du dimensionnement d'un rectangle (classe CompositeRectangle) basée sur les translations simultanées de deux de ses coins. Ces translations peuvent être indifféremment réalisées par les deux mains d'un même utilisateur ou par deux utilisateurs. Dans le cas 5.1-a, le dimensionnement est consistant car les deux outils "main" ont des positions conformes à celles des points qu'ils translatent. Par contre, dans le cas 5.1-b, l'interaction est inconsistante car les translations respectives et simultanées du point bas-gauche de $(dx_1,\ dy_1)$ et du point bas-droit de $(dx_2,\ dy_2)$ impliquent une variation dy de la coordonnée *y commune* à ces deux points de $dy_1 + dy_2$. Il en résulte que les positions des deux outils "main" sont non conformes aux positions de ces deux points bien qu'ils soient à l'origine des deux translations. Nous jugeons cette situation confondante pour l'utilisateur.

La figure 5.2 représente des interactions inconsistantes qui ne sont pas le résultats d'actions simultanées sur un même objet, mais la conséquence d'un rendu altéré non conforme à l'état réel des objets manipulés. Le cas 5.2-a illustre l'incohérence d'une translation effectuée au travers d'une loupe possédant un facteur de grossissement de 2. Lorsque l'outil applique sa translation $(dx,\ dy)$ au carré, le carré se déplace effectivement de $(dx,\ dy)$ et

[2]Nous aborderons le problème de l'incohérence en collaboration temps réel dans le chapitre 6.



(a) Translation inconsistante (b) Piquer inconsistant

FIG. 5.2 – Actions inconsistantes au travers d'une lentille

la loupe affiche alors un déplacement du carré de $(2 \times dx,\ 2 \times dy)$. Par conséquent, la main agissant par-dessus la loupe, les positions respectives de la main et du carré ne sont pas conformes.

Le cas 5.2-b représente une lentille magique qui effectue le rendu des graphiques en affichant leur contour. Ainsi, lorsque l'outil se situe sur contour du cercle, l'utilisateur a l'impression qu'il pointe le cercle. Cependant, toute action réalisée à cette position sera produite sur le carré et non sur le cercle. Ceci provient du fait que le type des formes géométriques n'est pas conforme à celui rendu : les deux formes sont pleines mais sont rendues en tant que formes filaires.

5.2.3 Définition

Nous définissons la cohérence de l'interaction comme suit :

> *Une interaction est consistante si et seulement si elle produit des résultats conformes à ce qui est effectivement présenté à l'utilisateur.*
>
> *Une interaction peut être inconsistante si elle réalise des actions simultanées sur un objet. Dans ce cas, une gestion locale de la concurrence d'accès peut remédier à l'incohérence.*
>
> *Une interaction peut être inconsistante si elle concerne la manipulation d'un objet* via *un rendu altéré de l'objet. Dans ce cas, une gestion de la conformité objet manipulé / objet rendu peut remédier à l'incohérence.*

Dans ce chapitre, nous proposons une solution pour résoudre le problème de l'incohérence de l'interaction dans le cas des actions simultanées. Une solution pour le cas du rendu altéré sera fournie à la fin du chapitre 6.

5.2.4 Verrouillage des propriétés

Comme nous l'avons indiqué en introduction, nous utilisons le concept de verrou afin de gérer la concurrence d'accès. Notons que l'utilisation des verrous suggère là encore le contexte de la collaboration à distance, *i.e.* de la cohérence d'objets partagés. Nous utilisons ici ce procédé pour les objets *locaux*, afin de gérer la concurrence d'accès et, par voie de conséquence, la cohérence de l'interaction.

Nous affinons en conséquence le modèle des verrous proposé dans la section 5.4. Le but de cet affinement est d'autoriser ou non la production d'une action de manière à rendre consistante l'interaction qui l'engendre. Cette autorisation consiste à rendre possible ou non l'invocation de la méthode d'exécution associée à l'action. Le problème est résolu d'une manière générale par Riveill (1995) : en associant à chaque objet un ensemble de contraintes de synchronisation (ou conditions d'activation), l'exécution d'une méthode n'est possible que si les conditions précisées par la contrainte sont respectées.

Le mécanisme que nous proposons s'avère plus élémentaire et reste exclusivement basé sur notre modèle de chaînage des actions : la production de gestes engendre la productions d'actions et la production d'actions induit une modification des propriétés du consommateur . Nous définissons, pour chaque action, quelles sont les propriétés qu'elle modifie. De par le polymorphisme des actions, cette définition ne peut être réalisée que par les différents consommateurs de l'action. Nous utilisons alors, à cet effet, les méthodes de notification de début et de fin définies dans les interfaces de consommation des actions. Par exemple, l'action de translation modifiant les propriétés x et y pour la classe Graphic, ces deux méthodes y sont implantées de la manière suivante :

```
  public void beginTranslate(TranslateProducer producer) {
    setXLocked(true) ;
    setYLocked(true) ;
   //...
}
public void endTranslate(TranslateProducer producer) {
    setXLocked(false) ;
    setYLocked(false) ;
   //...
}
```

En procédant de la sorte, il n'est pas possible d'avoir deux actions de translation produites simultanément sur le même graphique, la méthode de faisabilité (canTranslate) testant si les propriétés x et y sont verrouillées[3]. Nous affinons le verrouillage en précisant également le *propriétaire* du verrou, ce qui s'avèrera indispensable par la suite :

```
  public void beginTranslate(TranslateProducer producer) {
   setXLocked(producer, true) ;
   setYLocked(producer, true) ;
   //...
}

public void endTranslate(TranslateProducer producer) {
   setXLocked(producer, false) ;
   setYLocked(producer, false) ;
   //...
}

public boolean canTranslate(TranslateProducer producer) {
   return isXEditable(producer) && isYEditable(producer) ;
}
```

La méthode isPpppEditable relative à la propriété pppp prend en conséquence en argument l'objet qui est susceptible d'éditer la valeur de la propriété. Une propriété est dans ces conditions éditable par un objet O si et seulement si la propriété n'est pas verrouillée *ou* si elle est verrouillée par O :

[3]Le problème de l'inter-blocage sera abordé dans la section 5.4.

```
// dans la classe Graphic :
public boolean  isXEditable(Object editor) {
   return isPropertyEditable("x", editor) ;
}

// dans la classe Component :
public boolean  isPropertyEditable(String propertyName, Object editor) {
   return !isPropertyLocked("x") || isPropertyLockedBy(“x”, editor) ;
}
```

Notons qu'il serait nécessaire d'affiner le modèle des propriétés en modifiant la forme des accesseurs en écriture de manière à vérifier, par une pré-condition, que l'objet qui modifie la valeur d'une propriété en a le droit. Par exemple, l'accesseur de la propriété x défini dans la classe Graphic pourrait s'écrire :

```
public void  setX(Object editor, double value) {
    assert  isXEditable(editor) ;
   // ... écriture de x
}
```

Nous n'avons cependant pas modifié le modèle des accesseurs dans OpenDPI à l'heure actuelle, jugeant cet aspect relativement contraignant pour le programmeur. N'oublions pas, en effet, que le verrouillage à pour but de permettre la garantie de la cohérence des interactions et n'est ainsi indispensable *que* pour le test de faisabilité d'une action.

5.2.5 Relâchement de la politique de verrouillage

La section précédente propose un mécanisme qui permet de garantir la cohérence de l'interaction dans les cas simples, tel que celui des translations simultanées d'un rectangle. Cependant, le verrouillage de propriété peut être considéré, dans certains cas, comme trop contraignant du point de vue de l'utilisateur[4]. Il peut être souhaitable, du point de vue de l'utilisateur, de relâcher la politique de verrouillage imposée par l'environnement interactif. Ceci est possible en utilisant le concept d'adaptateur (section 3.3.1.2).

Dans l'exemple du rectangle, il pourrait être tout à fait utile de pouvoir effectuer deux translations simultanées lesquelles équivaudraient à une unique translation dite “glissante”. L'utilisation d'un adaptateur translation $\rightsquigarrow$ translation glissante (classe SlidingTranslationAdapter) permet, d'une manière simple, de relâcher la politique de verrouillage pour l'action de translation. Cet adaptateur consomme plusieurs actions de translations (simultanées) et ne fournit en sortie qu'une seule translation glissante. Cet instrument étant le producteur de la translation glissante, il est l'unique propriétaire des verrous posés sur les propriétés x et y du graphique translaté. Ceci l'autorise alors à effectuer la translation glissante résultant des translations simultanées consommées. Ce type d'instrument peut être inséré à l'intérieur même du graphique (et au-dessus) de manière à modifier son comportement de la translation en une translation glissante.

5.3 Induction du verrouillage

5.3.1 Principe

Si la technique de verrouillage précédente est suffisante pour résoudre, par exemple, le problème de la translation d'un graphique, elle reste insuffisante pour résoudre l'incohé-

[4] Du point de vue du programmeur, le verrouillage de propriété sera *toujours* contraignant !

rence du dimensionnement du rectangle composite (figure 5.1). Il est en effet nécessaire de pouvoir *propager* le verrouillage d'un composant à l'autre.

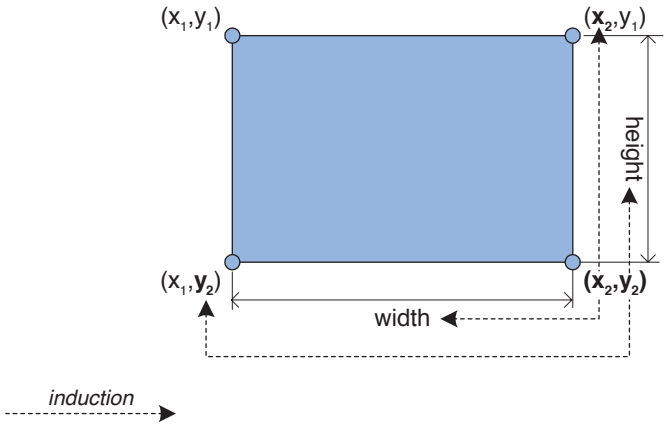


FIG. 5.3 – Induction du verrouillage lors d'un dimensionnement

Dans l'exemple du rectangle composite, il s'agit de propager le verrouillage des propriétés x et y du point bas-droit aux propriétés x, y, width et height du rectangle qui le contient, ainsi qu'aux propriétés respectives x et y des points haut-droit et bas-gauche (figure 5.3). Nous voyons sur cet exemple que la propagation ne se fait pas en suivant l'axe de la composition des composants (des composants contenants vers les composants contenus) mais selon l'axe de l'observabilité des composants (des composants observés vers les composants observateurs).

5.3.2 Définition

La propagation du verrouillage d'une propriété vers d'autres propriétés est un mécanisme appelé *induction du verrouillage* :

> *L'induction du verrouillage d'une propriété P consiste à propager le verrouillage et le déverrouillage de P vers d'autres propriétés qui sont en relation de dépendance avec P.*
>
> *L'induction du verrouillage suit l'axe d'observation des composants, transversal par rapport au graphe de scène, par opposition à l'axe vertical relatif à la composition des composants.*

5.3.3 Granularité des propriétés

Nous n'avons pas, jusqu'à présent, justifié formellement le choix de la granularité quant à la définition des propriétés. Par exemple, la classe Graphic définit la position d'un graphique au travers des propriétés x et y de type double, ses dimensions par les propriétés width et height de type double, et sa couleur par la propriété color instance de Color. Cependant, il aurait été possible de choisir une autre granularité pour définir ces trois éléments : la position pourrait être définie plus grossièrement par une propriété location instance de Location, les dimensions par une propriété size instance de Size, et la couleur pourrait être définie plus finement par trois propriétés red, green, blue de type float.

Le choix d'un niveau de granularité pour les propriétés est en fait dicté par celui relatif à leur verrouillage, *i.e.* par la finesse recherchée de l'interaction (consistante). Par exemple, en supposant que la position d'un graphique soit précisée par la propriété location précédente, le dimensionnement du rectangle composite (figure 5.1) conduirait à un verrouillage de la propriété location du point bas-droit soit, par induction, au verrouillage des propriétés location des trois autres points. Nous constatons ainsi, sur cet exemple, que le choix de la granularité d'une propriété a une incidence non négligeable sur la finesse des interactions (consistantes) envisageables. Dans OpenDPI, le choix de définir la couleur avec la propriété color implique qu'il n'est pas possible de modifier simultanément plus d'une des trois composantes $(r,\ v,\ b)$ d'un graphique. Un tel choix impose des limites pour les concepteurs et devrait, par conséquent, être réalisé avec leur consentement.

5.3.4 Affinement du modèle d'observation

La réalisation de l'induction du verrouillage implique la possibilité *d'observer les verrous* posés sur les propriétés. Dans l'exemple précédent, le rectangle composite observe le verrouillage du point bas-droit de manière à induire le verrouillage comme l'indique la figure 5.3. Nous proposons de définir le mécanisme d'observation des verrous par dessus celui de l'observation des propriétés, en vertu du deuxième point de la définition de l'induction.

Par exemple, l'interface d'observation des propriétés de la classe Graphic est augmentée des *méthodes de notification du verrouillage* comme suit :

```
public interface GraphicObserver extends PropertyObserver {
  public void newX(double x) ;
  public void newY(double y) ;
  public void  newXLocked(Producer owner, boolean locked) ;
  public void  newYLocked(Producer owner, boolean locked) ;
  //...
}
```

5.3.5 Exemple : le rectangle composite

La figure 5.4 illustre l'induction du verrouillage lorsque le point "bas-droit" est translaté :

1. Le premier niveau d'induction se situe à l'intérieur même du rectangle composite. Lorsqu'il y a déplacement (*i.e.* x et y verrouillés), le rectangle contenant verrouille les propriétés y du point "bas-gauche" et x du point "haut-droit". Ceci garantit la cohérence de l'interaction lors des déplacements des quatre points du rectangle. Il y a également induction vers les quatre segments du rectangle composite, et vers le rectangle lui-même (non représentée).
2. Le second niveau d'induction concerne le verrouillage des propriétés relatives aux données du domaines. Lorsque la propriété x du point "bas-droit" est verrouillée, les propriétés xCenter et width des données du rectangle (données du domaine) sont verrouillées. De même, le verrouillage de y induit celui de yCenter et de height.
3. Le troisième niveau d'induction concerne le verrouillage des propriétés de la seconde présentation. Lorsque la propriété xCenter de la donnée du domaine est verrouillée, la propriété string du champ de saisie "X centre" est verrouillée. Cette induction touche de même les autres propriétés de la donnée du domaine.

L'induction pour les deux premiers niveaux est réalisée par l'observation des verrous des quatre points, comme suit :

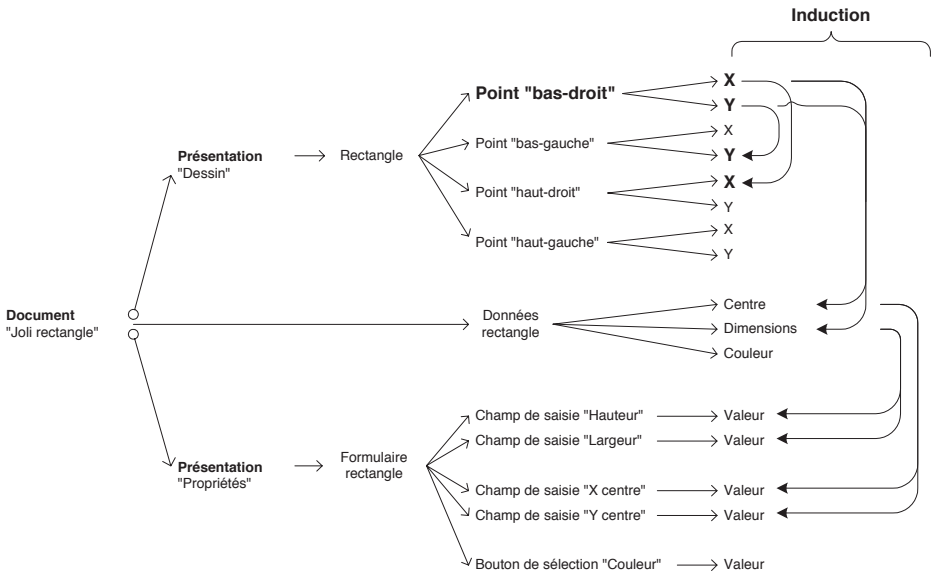


FIG. 5.4 – Induction du verrouillage *via* un document

```
public class CompositeRectangle extends Rectangle {

  public CompositeRectangle(RectangleData data) {
    super(data) ;
    points = new Point[4] ;
    for (int p = 0 ; p < points.length ; p++) {
      Point point = new Point() ;
      points[p] = point ;
      addChild(point) ;
      point.addPropertyObserver(new PointObserver(p)) ;
    }
    //...
  }

  protected class PointObserver extends GraphicBasicObserver {
    //...
    public void newX(double x) {
      //...
    }
    public void newY(double y) {
      //...
    }
    public void  newXLocked(Producer owner, boolean locked) {
      //...
      // induction vers les points
      for (int n = 0 ; n < 4 ; n++)
        if ((n != p) && haveCommonX(n, p))
          points[n].setXLocked(points[p], locked) ;
      // induction vers les segments ...
      // induction vers le rectangle
       setXLocked(points[p], locked) ;
      // induction vers le document
      if (data != null) {
```

```
        data.setXCenterLocked(points[p], locked) ;
        data.setWidthLocked(points[p], locked) ;
      }
    }
    public void  newYLocked(Producer owner, boolean locked) {
      // meme principe que pour newXLocked...
    }
    protected int p ;
  }

  protected Point[] points ;
  //...
}
```

Lors de la construction du rectangle composite, les propriétés x et y des quatre points sont observées par quatre instances internes de PointObserver. Quand le verrouillage de la propriété x de l'un des points a lieu, la méthode de notification de verrouillage (newXLocked) est invoquée. Elle détermine alors quels sont les points qui ont une propriété x commune et, pour chacun de ces points, verrouille la propriété x. Elle réalise une opération semblable pour les quatre segments (non représentée) et pour le rectangle lui-même. De plus, si le rectangle composite est attaché à une donnée du domaine (data != null), le point verrouille ses propriétés xCenter et width.

5.3.6 Coût de l'induction

Les exemples d'interaction donnés dans la section 5.2 illustrent l'intérêt qu'il y a, du point de vue de l'utilisateur et du concepteur, à gérer la concurrence d'accès aux propriétés des composants. Cependant, du point de vue du programmeur, la réalisation d'une telle gestion de la concurrence induit un effort de programmation non négligeable. L'exemple (simplifié) de la section précédente illustre ce propos. De plus, l'approche ouverte de DPI, basée sur des composants *indépendants*, ne facilite pas l'implantation de l'induction.

Cette remarque est importante car elle peut influencer le choix de la granularité des propriétés. Ce choix est en effet essentiel car il est lié à la complexité de la programmation de l'induction : un grain fin implique une induction complexe à mettre en œuvre et, inversement, un grain plus élevé simplifie l'induction. Il peut ainsi, dans certains cas, s'avérer préférable d'utiliser une granularité forte au détriment de la finesse de l'interaction, plutôt que de proposer une interaction fine mais dont l'implantation risque de générer des erreurs.

5.4 Simultanéité forte

5.4.1 Problème

Deux actions a_1 et a_2, produites dans les intervalles de temps respectifs $[t_{1d},\ t_{1f}]$ et $[t_{2d},\ t_{2f}]$, sont dites simultanées si et seulement si $[t_{1d},\ t_{1f}] \cap [t_{2d},\ t_{2f}] \neq \emptyset$. La simultanéité de deux actions n'implique pas nécessairement que les actions commencent (et se terminent) au même instant. La simultanéité est dite *forte* lorsque les deux actions commencent au même instant, *i.e.* si et seulement si $t_{1d} = t_{2d}$; elle est dite *faible* dans le cas contraire.

Les considérations précédentes traitent convenablement le cas des actions faiblement simultanées. Par contre, un problème peut survenir dans le cas d'une simultanéité forte, analogue à l'inter-blocage[5] que l'on rencontre dans les approches basées sur les verrous (Habermann, 1969).

[5] "*Deadlock*"

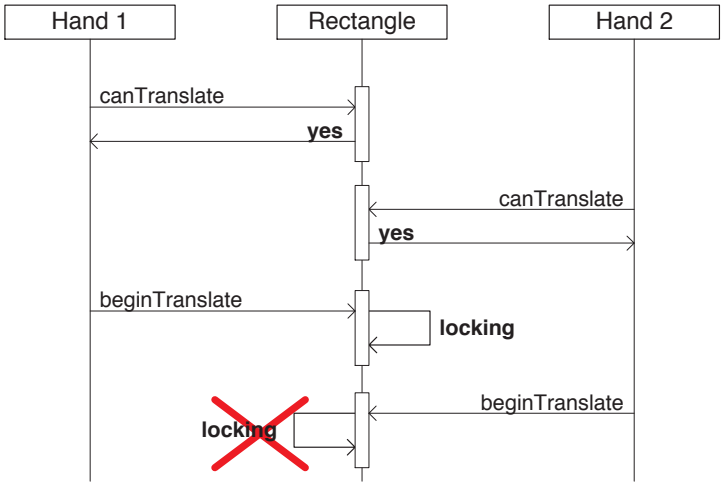


FIG. 5.5 – Verrouillage direct en simultanéité forte

La figure 5.5 illustre le problème lors des translations fortement simultanées d'un rectangle. Une première main réalise le test de faisabilité de la translation par une invocation de la méthode canTranslate sur le rectangle, laquelle confirme cette possibilité. Au même instant, une deuxième main effectue un test de faisabilité identique. La deuxième invocation de la méthode canTranslate n'est alors effectuée qu'après la première en utilisant le mécanisme de synchronisation de Java (mot clé synchronized). Puisque la première action de translation n'a pas démarré, le résultat de cette deuxième invocation est également positif. Les deux mains peuvent alors commencer la production de la translation. La première main invoque la méthode beginTranslate, laquelle réalise le verrouillage des propriétés x et y du rectangle. Cette méthode étant également synchronisée, sa deuxième invocation a lieu juste après ; elle tente alors de verrouiller les propriétés x et y du même rectangle, ce qui n'est plus possible. La deuxième main a ainsi démarré l'action de translation puisqu'elle lui semblait faisable, mais sa production va alors conduire à une interaction inconsistante.

5.4.2 Solution

Une solution simple au problème précédent consisterait à définir les deux méthodes de faisabilité et de début d'une action en une seule méthode synchronisée. En procédant ainsi, le verrouillage des propriétés a lieu immédiatement après le test de verrouillage, sans interruption possible. Cependant, s'il y a induction de verrouillage, il n'est plus possible de définir une telle synchronisation.

La figure 5.6 reprend l'exemple du rectangle (composite) dimensionné par des translations fortement simultanées de ses deux points bas adjacents (figure 5.1-b). Les deux mains démarrent les translations puisque ces dernières sont faisables. Ce faisant, les propriétés x et y des deux points "bas-gauche" et "bas-droit" sont verrouillées. Ceci conduit alors, par le biais de l'induction, à deux tentatives respectives de verrouillage de la propriété y des points adjacents. Ces deux tentatives échouant, l'interaction devient inconsistante.

La solution précédente ne peut fonctionner ici puisque, les invocations de canTranslate s'effectuant sur deux points *différents*, elles ne peuvent pas être synchronisées (le diagramme 5.6 affiche un tel parallélisme). Nous percevons alors qu'il est nécessaire que l'induction ait opéré pour pouvoir *constater* l'impossibilité du verrouillage. Cette remarque suggère d'utiliser le mécanisme des exceptions afin d'*essayer* (mot clé try) de démarrer la production d'une action (Eckel, 2002). Si une exception de verrouillage (classe LockingException)

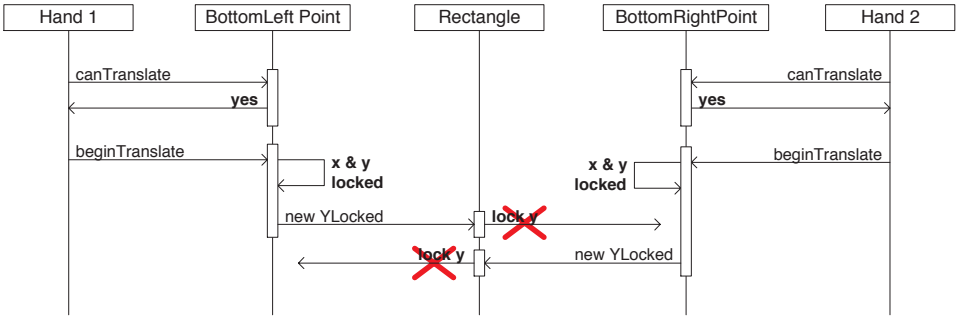


FIG. 5.6 – Verrouillage par induction en simultanéité forte

est levée lors du démarrage de l'action (mot clé throw), cela signifie qu'une simultanéité forte a eu lieu et que ce démarrage doit être interrompu. Dans ce cas, il est nécessaire de déverrouiller les propriétés qui ont été verrouillées, induisant du même coup l'annulation des verrous posés par induction. Cette solution n'a pas été implantée à l'heure actuelle dans OpenDPI.

Notons que, du fait de l'absence de vérification du droit d'écriture par les accesseurs, les exemples précédents ne conduisent pas à proprement parler à un inter-blocage puisque les actions peuvent continuer à être produites, *i.e.* à modifier des propriétés *via* les accesseurs. Ainsi, si le mécanisme d'exception précédent n'est pas utilisé, une simultanéité forte ne bloque pas à proprement parler l'interaction mais la rend probablement inconsistante, tandis qu'une simultanéité faible conduit nécessairement à une interaction consistante.

5.5 Partage d'instrument

Le fait que tous les objets présents à l'écran soient partagés (puisque l'écran lui-même l'est[6]) nous permet d'envisager de partager un outil. Ce type de partage peut être utile lors d'une phase d'*apprentissage* d'un instrument complexe. Nous le retrouvons notamment dans l'exemple de la conduite automobile. L'élève, plutôt que d'utiliser un simulateur de conduite, conduit le véhicule *en situation* réelle sous la responsabilité du maître. Le maître dispose d'un pédalier analogue à celui de l'élève, ce qui lui permet d'ajuster les commandes en cas d'erreur de conduite de l'élève. Dans cet exemple, les deux pédaliers sont les périphériques d'entrée des deux utilisateurs, le véhicule étant l'instrument partagé.

Le partage d'un outil consiste ainsi à ce que les périphériques de pointage des utilisateurs constituent la partie physique de l'outil. Par exemple, si deux utilisateurs partagent l'outil "main", leur périphérique de pointage respectif génère les gestes de translation et d'appui vers cet unique outil. La figure 5.7 fournit le diagramme de composant d'un main partagée entre deux utilisateurs A et B. Nous voyons que les deux détecteurs ont été étendus de manière à permettre l'observation des états de *plusieurs* périphériques d'entrée.

Le partage d'un outil est une opération qui doit pouvoir être réalisée dynamiquement à partir de deux outils de même type, et tant que les périphériques associés sont de même nature. En effet, il semble peu opportun, du point de vue du concepteur, d'autoriser l'apprentissage d'un instrument avec deux parties physiques de nature différente.

[6]Nous parlerons d'induction du partage dans le chapitre 6.

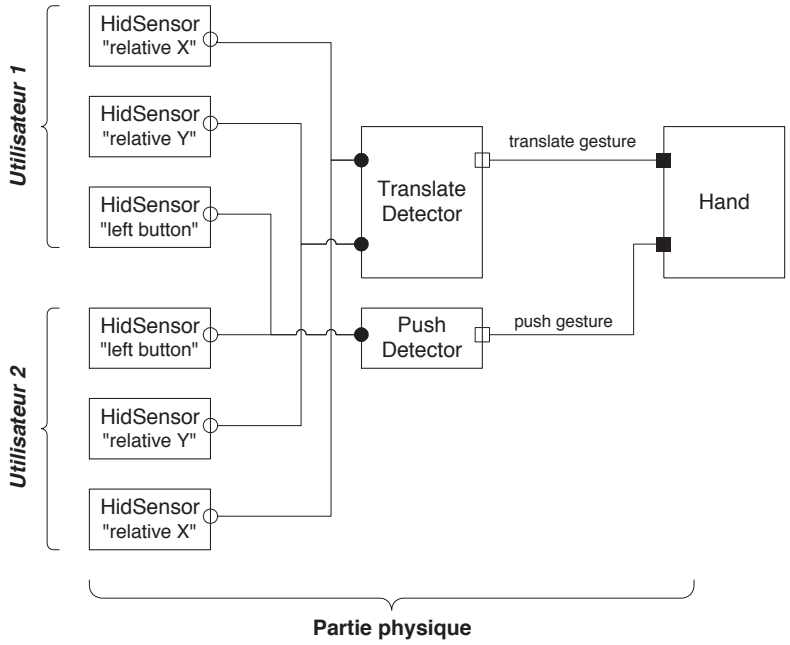


FIG. 5.7 – Partage d'un instrument

5.6 Faire, défaire et refaire

5.6.1 Prise en compte des utilisateurs

Dans une approche collaborative, il est important de bien identifier qui fait quoi, c'est-à-dire avoir *conscience* des activités d'autrui. Cette remarque vaut aussi bien pour la collaboration locale que pour la collaboration distante, bien qu'elle devienne plus problématique dans le deuxième cas (chapitre 6).

Nous définissons par conséquent la classe User de manière à pouvoir *identifier* les différents utilisateurs qui agissent ensemble sur un écran partagé. Elle précise l'identité de l'utilisateur en donnant son surnom, sa couleur d'identification ainsi que sa main dominante. Lorsqu'un second périphérique de pointage est connecté, le système peut par exemple demander à l'utilisateur de préciser si ce périphérique sera utilisé par sa main non dominante ou bien par un second utilisateur. Dans le premier cas, le système détermine si la main non dominante de l'utilisateur est sa main droite ou gauche de manière à adapter en conséquence la forme de l'outil affiché. Dans le second cas, il peut de plus afficher sur l'outil associé au second utilisateur son surnom ou/et sa couleur d'identification.

De plus, nous devons affiner le modèle de production des actions de manière à prendre en compte ce qu'un utilisateur est *autorisé* à faire. Nous modifions légèrement la méthode de test de faisabilité des interfaces de consommation de manière à ce que l'utilisateur générant l'action soit clairement identifié. Par exemple, la méthode canTranslate de l'interface TranslateConsumer s'écrit maintenant :

```
public boolean canTranslate(TranslateProducer producer,  User user) ;
```

Un composant gérant des droits d'accès peut alors vérifier qu'un utilisateur peut produire une action donnée en invoquant la méthode de faisabilité. Par exemple, les pièces blanches d'un jeu d'échec peuvent être translatées uniquement par le joueur qui a commencé la partie *et* si c'est à son tour de jouer, *ou* par le système lui-même.

5.6.2 Annulation des actions simultanées

Dans l'exemple du dimensionnement consistant du rectangle composite (figure 5.1-a), les deux actions de translation peuvent avoir lieu "en même temps", c'est-à-dire que l'une se termine alors que l'autre se poursuit. Cependant, rien n'indique que ces deux translations sont simultanées *et* liées : si la simultanéité peut être testée en comparant leurs dates de début et de fin, il est impossible de déterminer si les actions sont liées, *i.e.* si elles font partie d'une unique interaction.

La seule solution qui nous semble envisageable est de définir explicitement les actions qui peuvent être jouées simultanément dans une unique classe. Par exemple, les actions de translation des quatre points du rectangle composite peuvent fusionner en une *unique* action de "quadruple translation". La liaison des quatre actions est ainsi explicitement déclarées par la classe. Nous n'avons pas implantée, à l'heure actuelle, cette solution dans OpenDPI.

Notons que les instruments à caractère purement bimanuel ne posent pas nécessairement de problème. Dans l'exemple des toolglasses, la translation du toolglass lui-même et le clic-au-travers sont en effet des actions non liées (section A.3).

5.6.3 Droit d'annulation

Dans une approche multi-utilisateurs, les actions ne sont plus anonymes. Puisqu'elles ne peuvent être produites que par des utilisateurs autorisés, elles ne peuvent être annulées et ré-exécutées que sous certaines conditions. Nous distinguons alors deux politiques possibles :

1. Une action est annulable par n'importe quel utilisateur.
2. Une action est annulable uniquement par l'utilisateur qui l'a générée.

Le cas 1 laisse la possibilité aux utilisateurs de défaire une action qu'ils n'ont pas faite eux-même. Le cas 2 est plus contraint et permet de préserver les actions de chacun. Le choix d'une politique est du ressort du concepteur.

Du point de vue du programmeur, le cas 1 ne pose pas de problème particulier. La cas 2 implique par contre de pouvoir :

– Identifier l'utilisateur qui annule l'action ;
– Bloquer le processus d'annulation si l'utilisateur n'est pas autorisé à annuler.

Le premier point s'appuie sur le fait que les entrées de l'historique précisent les utilisateurs à l'origine des actions. Notons cependant qu'il n'est pas toujours aisé d'identifier l'utilisateur qui demande une annulation ou une ré-exécution. Par exemple, OpenDPI fournit un instrument d'annulation (classe Undoer) qui peut avoir pour partie physique les touches Ctrl+Z (annulation) et Ctrl+Z (ré-exécution) d'un clavier, ou bien la molette d'une souris (un déplacement négatif de la molette impliquant une annulation, et un déplacement positif une ré-exécution). Dans le premier cas, le clavier n'est pas nécessairement associé à un utilisateur en particulier et la sélectivité de l'annulation pose alors problème. Ce problème disparaît plus facilement dans le deuxième cas. Cette difficulté est là encore du ressort du concepteur.

5.7 Conclusion

Nous avons défini, dans ce chapitre, le problème de l'incohérence de l'interaction qui peut survenir dans le cas d'actions produites simultanément sur un objet. Nous avons proposé une solution basée sur le verrouillage des propriétés afin de pouvoir garantir la cohérence

de l'interaction. Si le principe du verrouillage reste simple, le phénomène d'induction du verrouillage peut rendre son implantation assez coûteuse. Il est cependant possible de s'affranchir, dans certains cas, du verrouillage. Dans notre exemple du rectangle composite, une action de rotation du rectangle peut se *substituer* à la translation de son point bas-gauche lorsque l'action de translation se produit sur le point bas-droit. Cette idée devrait être explorée après la thèse.

Nous avons également défini le mode de collaboration basé sur le partage d'instruments et permettant l'apprentissage collaboratif d'un instrument. Enfin, le problème de l'annulation des actions simultanées et liées a été abordé et une solution a été proposée.

Chapitre 6

Collaboration à distance : le partage de composants

Contents

La collaboration à distance consiste à rendre possible le *partage* de composants DPI entre des utilisateurs géographiquement répartis. Contrairement à la collaboration locale, l'espace physique des utilisateurs n'est plus commun. Par conséquent, leurs écrans ne sont plus partagés et la communication informelle naturellement induite par leur présence réciproque n'a plus cours. Afin que notre modèle rende possible la collaboration à distance, nous nous attachons à définir la manière dont il est possible de "coupler" les différents écrans des utilisateurs, *i.e.* de partager les éléments qu'ils affichent. La facette relative à la communication "informelle", laquelle est du ressort des mediaspaces, sort du cadre de cette thèse.

Ce dernier chapitre est composé de sept sections. Le section 6.1 introduit la notion de collaboration à distance par le biais d'un élément unique, la place, qui joue le rôle de pivot pour la collaboration du point de vue de l'utilisateur. La section 6.2 présente le problème général du partage des composants en abordant le principe de la réplication des composants ainsi que le couplage temporel et spatial des répliques. La section 6.3 précise comment notre modèle a été affiné afin de permettre le partage par réplication des composants DPI. Elle présente en particulier le concept de point de partage qui est, du point de vue du développeur, au cœur de la facette collaborative du modèle DPI. Les sections 6.4 et 6.5 abordent les problèmes classiques de la collaboration à distance, à savoir la concurrence d'accès, la cohérence des répliques et la conscience mutuelle des actions faites par autrui. La section 6.6 présente comment le concept de point de partage peut être utilisée afin de définir une gestion consistante du rendu graphique altéré. La section 6.7 conclut ce chapitre sur la facette collaborative du modèle et précise en particulier son état d'avancement dans OpenDPI.

6.1 Un élément pivot pour la collaboration : la place

Afin d'étendre le modèle DPI afin qu'il puisse prendre en compte les aspects de la collaboration synchrone et asynchrone, il est important de d'identifier les situations typiques de la collaboration à distance. Nous illustrerons ces situations au travers de scénarios de conception, lesquels seront repris dans la section 6.3 de manière à décrire comment ils peuvent être réalisés en utilisant le modèle DPI. La collaboration a été imaginée autour d'un unique composant interactif, appelé "place" (Beaudoux, 2002), qui permet d'initier tous les types de collaboration que nous avons pu identifier (sections 1.4 et 2.2).

6.1.1 Partager des types d'objet variés

Une place est, du point de vue de l'utilisateur, un lieu dédié au partage d'objets divers et variés entre plusieurs utilisateurs. Cette notion est très proche de la considération de "place sans espace" (Harrison & Dourish, 1996) : une place n'est pas un espace réel partagé dans lequel la collaboration peut être physiquement engagée, mais plutôt une place abstraite dans laquelle différents types d'artefacts sont accessibles à des utilisateurs par des possibilités de partage variées.

Une place est un composant permettant à des utilisateurs inscrits d'accéder aux objets répliqués qu'elle contient (figure 6.1). Elle est constituée de deux parties : l'ensemble des utilisateurs enregistrés et l'ensemble des objets partagés. Les utilisateurs peuvent y jouer différents rôles (administrateur, participant ou invité) et les objets partagés peuvent être des documents, des dossiers, des fenêtres, des espaces de travail, des instruments ou d'autres places. La place est le point d'entrée pour toute collaboration basée sur le partage d'objet.

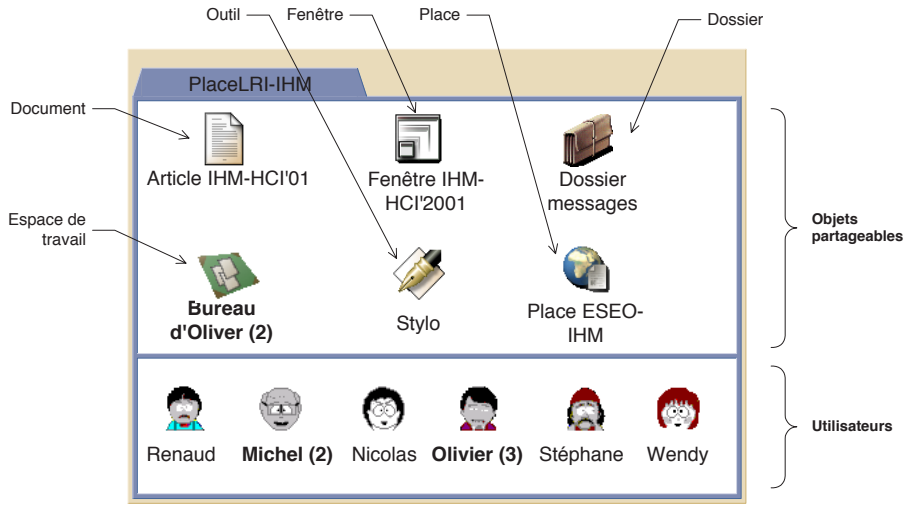


FIG. 6.1 – Un exemple de place

6.1.2 Collaborer selon des styles variés

Le partage d'une grande variété d'objets *via* une place offre un large panel de styles de collaboration.

Un document partagé peut être édité de manière synchrone en utilisant la version courante du document, laquelle correspond à une réplique synchrone du document. Il peut également être édité de manière asynchrone en utilisant la version de travail du document, laquelle correspond à une réplique asynchrone du document.

L'utilisateur peut choisir d'accéder au contenu du document et ainsi partager les données du domaine et les présentations avec d'autres utilisateurs. Il peut également partager ce document en partageant une fenêtre qui contient l'une de ses présentations. Le premier cas correspond à un travail en mode WYSIWIS relâché puisque seul le contenu du document est partagé. Dans le second cas, le travail s'effectue en mode WYSIWIS strict puisque la fenêtre et la présentation du document qu'elle contient sont partagées : toute action, tel que le défilement de la présentation à l'intérieur de la fenêtre ou son dimensionnement, est répliquée sur les différents postes, affichant ainsi le contenu de la fenêtre d'une manière semblable.

La possibilité d'inclure un espace de travail dans une place publique fusionne la métaphore du "bureau" (*i.e.* un espace personnel) avec la métaphore de la "salle de réunion" (*i.e.* un espace commun) en une unique métaphore : celle d'un bureau personnel qui peut accueillir d'autres participants. Le fait de travailler dans un espace partagé induit naturellement un mode synchrone et WYSIWIS strict puisque l'espace est "physiquement" partagé.

Le fait de pouvoir travailler en mode synchrone ou asynchrone, en WYSIWIS relâché ou strict, dans un espace de travail personnel ou partagé, permet à l'utilisateur d'ajuster le degré de couplage, et donc le niveau d'intimité, selon ses souhaits. Ce niveau peut varier du plus faible, ce qui est le cas en travaillant de manière asynchrone dans un espace personnel, au plus élevé, ce qui est le cas en travaillant dans un espace partagé.

Les vues iconiques des objets partageables permettent, par exemple, de donner conscience de certaines des activités des utilisateurs en fonction du degré d'intimité désiré. Par exemple, un nom d'utilisateur en gras signifie que cet utilisateur accède au nombre d'objets indiqué entre parenthèses. De même, le label d'un objet en gras indique que cet objet est accédé

par le nombre d'utilisateur indiqué entre parenthèses. Ainsi, un utilisateur qui travaille dans son espace personnel sur une version d'un document le fait en toute discrétion, ce qui n'est pas le cas (par défaut) lorsqu'un utilisateur partage un espace de travail.

Le fait de pouvoir insérer un instrument dans une place permet aux utilisateurs d'accéder à des instruments qu'ils n'ont pas nécessairement à disposition depuis leur propre espace de travail. Ceci peut s'avérer utile afin de collaborer à l'édition d'un document en utilisant les mêmes instruments. Par exemple, un stylo qui permet, en plus des fonctions usuelles liées à l'édition d'un texte (frappe clavier et application d'attributs de style), de surligner une partie du texte et donc d'annoter le document, peut s'avérer utile aux autres participants qui ne posséderaient pas d'instrument de type "surligneur". Cet "échange" d'instrument rend alors possible l'édition homogène de documents partagés[1].

Enfin, une place B peut être insérée dans une autre place A de manière que chaque utilisateur référencé dans A puisse accéder aux objets insérés dans B. Cette organisation permet de créer facilement un modèle de délégation entre les places. Par exemple, la place "LRI-IHM", qui concerne les activités du groupe IHM du LRI, fournit un accès aux objets de la place publique "ESEO-IHM", qui concerne l'activité IHM de l'ESEO. Il est de la même manière possible de définir l'accès croisé à la place publique "LRI-IHM" depuis "ESEO-IHM".

6.1.3 Scénarios

La section précédente synthétise ce qu'il est possible de faire au travers d'une place sans pour autant préciser comment cela est effectivement possible. Nous donnons, dans cette section, un exemple de conception qui caractérise une façon possible de travailler avec une place. Le scénario proposé a ainsi été imaginé du point de vue du *concepteur*.

La figure 6.2 fournit la "photographie" d'un espace de travail relative au scénario illustratif suivant :

> "Michel et moi-même (Olivier) travaillons ensemble à la rédaction d'un article pour la conférence IHM-HCI'2001."

Nous détaillons chaque "sous-scénario" de ce scénario principal dans les sous-sections suivantes, en relation avec le contenu de figure 6.2.

6.1.3.1 Travailler dans un espace personnel

Dans un premier temps, l'article relatif à la conférence IHM-HCI'2001 est construit dans l'espace personnel d'Olivier :

> "Je commence par travailler seul dans mon espace de travail en utilisant un classeur qui contient trois pages affichant les documents suivants : l'article *Article IHM-HCI'01* qui contient la structure initiale de la soumission et quelques lignes de texte, l'article *Instrument CHI'00* qui décrit les fondements de l'interaction instrumentale, et le document *Architecture DPI* qui fournit les explications de l'implantation en cours de DPI. J'édite l'article *Article IHM-HCI'01* en utilisant l'instrument *Stylo* tout en me référant aux deux autres documents de manière à rester cohérent."[2]

Remarquons ici qu'Olivier utilise un seul classeur qui contient les trois documents. Les autres classeurs représentés dans la figure 6.2 apparaissent ultérieurement.

[1] Cet aspect rejoint la notion de compatibilité instrument / document (section 3.3.1.2).

[2] Les noms en italique font référence à des éléments présentés dans la figure 6.2.

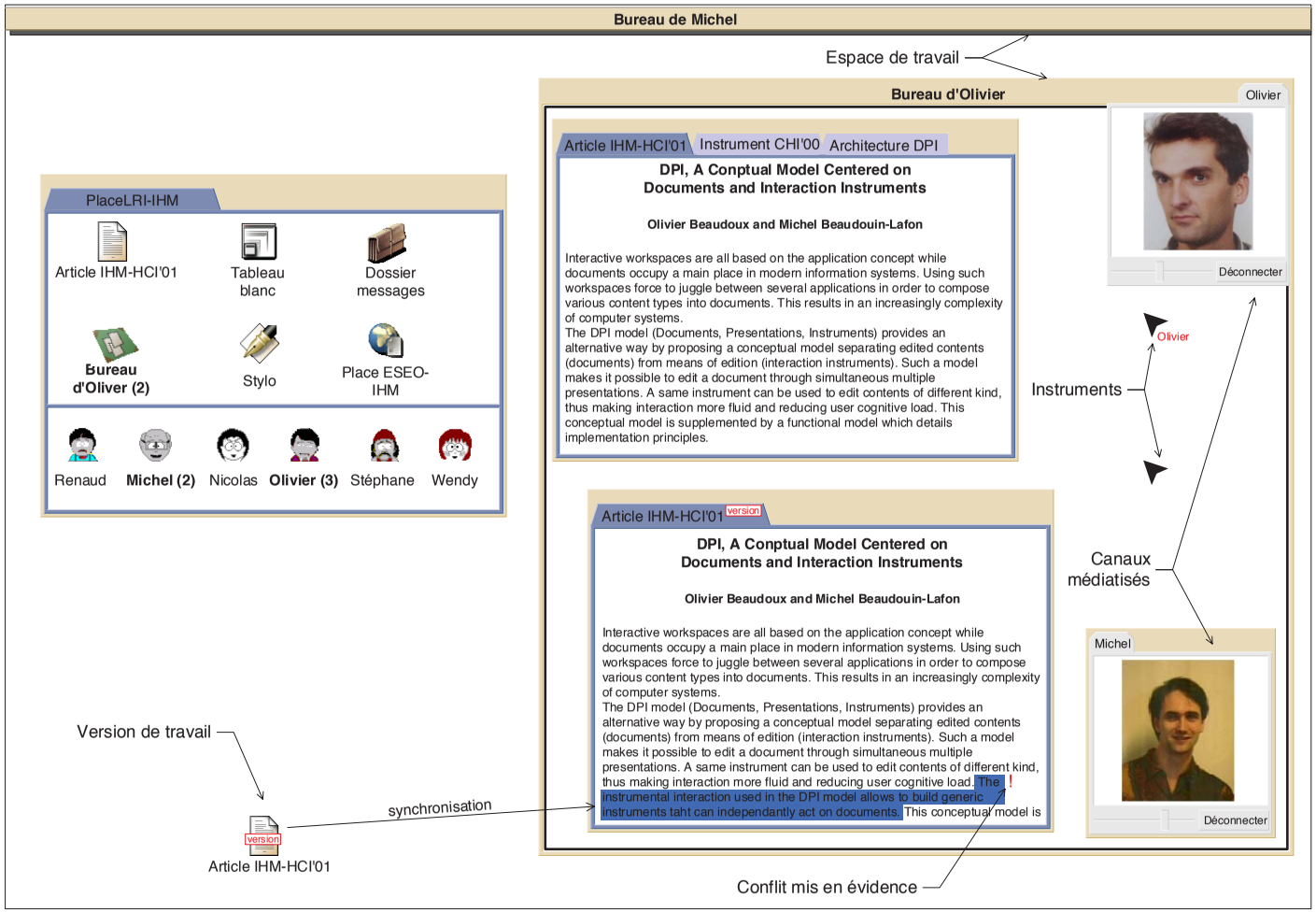


FIG. 6.2 – Scénario d'utilisation d'une place

6.1.3.2 Partager des objets *via* une place publique

Maintenant que le document existe, il est possible de le partager de manière à permettre son édition collaborative :

> "Une fois que j'ai créé un premier contenu à l'article, je décide de le rendre accessible par la *Place LRI-IHM*."

Cette place est accessible aux six collègues qui travaillent sur le thème de l'IHM au LRI. Il contient à cet instant :

- Un *Tableau Blanc* qui permet de dessiner librement ses idées en mode WYSIWIS strict. Il s'agit d'une page contenant un document de type dessin à main levée.
- Un *Dossier Messages* qui contient la file des messages relatifs à l'activité des utilisateurs dans la place, renseignée par les utilisateurs eux-mêmes.
- La *Place ESEO-IHM* qui fournit l'accès aux objets de cette place aux utilisateurs enregistrés dans la *Place LRI-IHM*.

De manière à permettre l'édition partagée du document, Olivier réalise les interactions suivantes :

> "Je glisse l'*Article IHM-HCI'01* dans la place puis j'écris un petit message afin d'initier le travail collaboratif. Je fournis également l'instrument *Stylo* que j'ai utilisé pour écrire l'article : si nécessaire, les utilisateurs enregistrés pourront l'installer en double-cliquant sur l'icone. Enfin, je notifie les utilisateurs de la place que son contenu a changé en cliquant sur le bouton Notifier (non montré)."

6.1.3.3 Collaborer *via* une version de travail d'un document

Les différentes personnes inscrites à la place LRI-IHM ont maintenant accès au document partagé. En s'apercevant que cette place possède un nouvel élément, elles peuvent décider d'interagir avec ce dernier :

> "Michel reçoit la notification (non montrée) qui concerne le changement du contenu de la place et lit mon message. Comme il est intéressé par mon travail, il glisse le document *Article IHM-HCI'01* sur son espace de travail personnel *Bureau de Michel*. Cette action crée une version du document. Il installe le *Stylo* dans son propre espace de travail puis ouvre le document dans un classeur. "

La version de travail du document est une réplique non synchronisée du document : lorsque Michel effectue ses actions sur le document, celles-ci ne sont pas répliquées sur document d'origine *via* la place mais sont stockées localement. Ces actions sont ultérieurement répliquées sur le document par le système, lors de la phase de synchronisation :

> "Après quelques actions d'édition, Michel valide sa version en cliquant sur l'icone *version*. Un conflit est détecté puis est mis en évidence par le système : l'une des modifications de Michel concerne une phrase qui a été entre temps supprimée du document courant (les différences sont surlignées sur la figure)."

Puisqu'il y a conflit d'intention, c'est aux utilisateurs qui ont appliqué ces intentions différentes de décider qu'elles actions doivent effectivement être appliquées au document.

6.1.3.4 Collaborer *via* un espace de travail partagé

Pour engager la résolution du "conflit", il est intéressant de discuter à l'intérieur d'un espace de travail partagé :

> "Michel a observé que j'étais en train de faire des actions sur un objet partagé de la place."

La place affiche en effet le prénom d'Olivier en gras et indique le nombre d'objets partageables qu'il est en train de manipuler.

> "Michel décide donc d'entrer en contact avec moi. Il double-clique sur l'icone me représentant et ouvre ainsi un canal médiatisé (vidéo + voix). Nous décidons de résoudre le conflit mis en évidence dans mon propre espace de travail. Michel joint cet espace en cliquant sur l'icone du *Bureau d'Olivier*. Il y déplace ensuite le classeur qui contient la version conflictuelle : le classeur devient partagé. Nous discutons de la version de Michel et nous nous mettons d'accord pour effectivement supprimer la phrase."

Le scénario se termine alors ainsi :

> "Michel et moi continuons à travailler sur la soumission et, une fois d'accord sur son contenu, nous déconnectons les canaux médiatisés. Michel quitte finalement l'espace partagé en le fermant depuis son espace personnel."

6.1.3.5 Conclusion

Ce scénario de conception illustre les possibilités de collaboration offertes par le concept de place. Nous pouvons constater que cette notion de place permet de jouer le rôle de *pivot* pour la collaboration : elle est l'élément central et unique qui permet d'engendrer les styles de collaborations que nous avons pu identifier à ce jour. Dans la section suivante, nous allons proposer un modèle de partage des composants DPI qui permettra de construire un tel élément, sans pour autant imposer le partage des composants par l'unique vecteur que représente la place.

6.2 Partage de composants

6.2.1 Architecture répliquée

6.2.1.1 Réplication *versus* centralisation

Il existe typiquement deux approches possibles pour la gestion du partage d'objets : l'approche centralisée et l'approche répliquée. Elles se distinguent par la manière dont les objets partagés sont distribués sur les différents postes connectés au réseau (Dewan, 1999).

Dans l'approche centralisée, le système met à disposition l'ensemble des objets partagés sur un poste dédié appelé *serveur*. Les différents collaborateurs accèdent alors à ces objets au travers de leurs propres postes, appelés postes *clients*, lesquels communiquent avec le serveur afin de *simuler* la présence locale des objets partagés. Dans l'exemple de la figure 6.3, le poste client A accède à l'objet partagé O_1 par le biais du point d'accès O_{1A} et le poste client B accède aux objets partagés O_1 et O_2 *via* les points d'accès O_{1B} et O_{2B}. Les

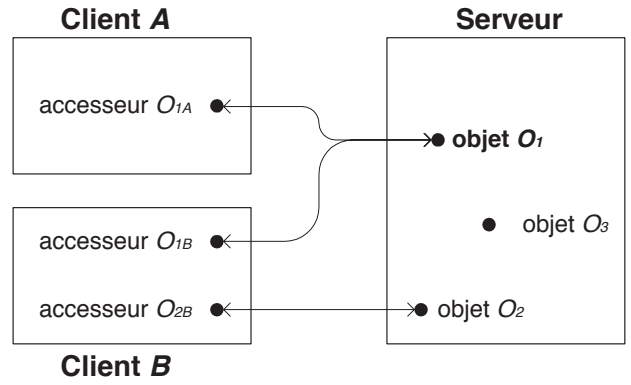


FIG. 6.3 – Architecture centralisée

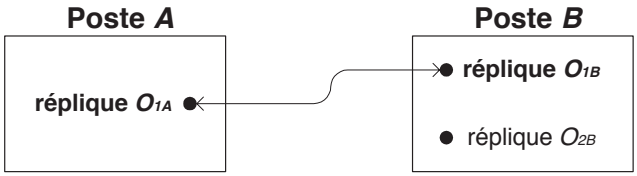


FIG. 6.4 – Architecture répliquée

utilisateurs des postes respectifs A et B peuvent alors travailler sur l'objet partagé O_1 tout en ayant l'impression que cet objet est localement présent sur leur poste.

Dans l'approche répliquée, il n'existe plus d'asymétrie entre les postes clients et serveurs : chaque poste possède une copie de l'objet partagé appelée *réplique*. Le terme d'objet partagé fait ainsi référence à l'ensemble de ses répliques réparties. Le problème principal de cette approche consiste à *synchroniser* les différentes répliques d'un même objet partagé de manière à s'assurer qu'elles sont dans le même état. Si tel n'est pas le cas, l'objet partagé est qualifié d'*inconsistant*. Dans l'exemple de la figure 6.4, l'objet partagé O_1 existe au travers de ses deux répliques O_{1A} et O_{1B} sur les postes respectifs A et B. D'une certaine façon, chaque poste joue à la fois le rôle de client (permanent) et de serveur (occasionnel). Lorsque qu'une action est effectuée sur O_{1A}, le poste A joue le rôle de serveur et effectue une notification au groupe constitué des autres postes, soit ici le poste B, ces derniers jouant alors le rôle de clients

Le choix entre une architecture répliquée et une architecture centralisée est sujet à beaucoup de débats. Il est cependant souvent admis qu'une architecture répliquée offre des avantages quant à l'efficacité d'exécution des applications, ce qui est un point important en interaction homme-machine. En effet, une architecture centralisée implique un dialogue permanent avec le serveur, même si l'objet partagé est accédé par un unique utilisateur. L'architecture répliquée ne possède pas un tel inconvénient puisque les actions sont d'abord appliquées localement *avant* d'être propagées sur les postes distants. De plus, une architecture répliquée est plus robuste de part la présence de multiples répliques. En contre partie, elle est plus complexe à mettre en œuvre en ce qui concerne la gestion de la cohérence.

6.2.1.2 Définitions

Nous donnons ci-dessous les définitions respectives d'une réplique, d'un objet partagé et de la cohérence d'un objet partagé, dans le cadre que nous envisageons pour notre architecture répliquée :

> *Une* réplique r *d'un objet partagé* O*, notée* $r \sim O$*, est une copie de* O *qui reste en permanence liée à* O*.*
>
> *Un* objet O partagé *par l'ensemble* U *des utilisateurs autorisés, noté* O/U*, est l'ensemble des répliques* $r_u \sim O$ *accessibles par tout utilisateur de* U *:* $O/U = \{r_u \sim O / u \in U\}$*. L'existence de* O *est ainsi abstraite : l'objet partagé n'existe pas à un endroit particulier mais est dilué au travers des répliques de chaque utilisateur.*
>
> *L'objet partagé* O/U *est dit globalement consistant à un instant* t *lorsque toutes ses répliques sont identiques à cet instant,* i.e. *si et seulement si :* $\forall (i,j) \in U^2,\ r_i \sim O =_t r_j \sim O$*.*

Nous affinerons, dans la section 6.2.2, la définition de la réplique en ce qui concerne sa liaison permanente avec l'objet partagé.

6.2.1.3 Réplication *"multicast"*

La collaboration distante nécessite le transport d'informations d'un utilisateur à l'autre par l'envoi de messages sur le réseau. Le *routage* spécifie les chemins que suivront les différents paquets de données qui transitent sur le réseau afin d'être acheminés à bon port.

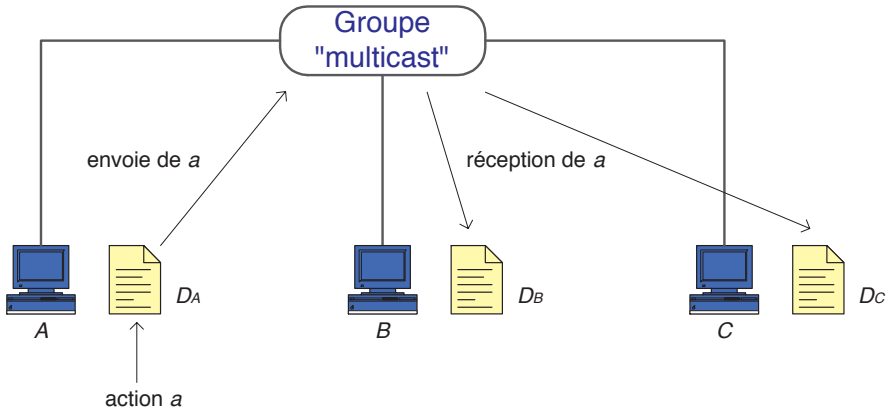


FIG. 6.5 – Réplication par "multicast"

La technique de routage la plus adaptée à notre problème est le "multicast". Le multicast est un moyen simple pour envoyer des messages à plusieurs destinataires en un seul envoi, par le biais des groupes (Wittmann et al., 2000). Un *groupe* est un ensemble de postes distants qui se sont enregistrés comme y appartenant. Il s'agit d'une entité logique car elle n'a pas d'existence physique sur un poste en particulier, mais est disséminée sur l'ensemble des postes du groupe : nous voyons là une très forte similitude avec la notion même d'objet partagé, ce dernier n'existant qu'au travers de ses différentes répliques. La figure 6.5 illustre la simplicité de la réplication d'une action a, faite depuis le postes A sur un document répliqué D, vers les deux autres postes B et C du groupe multicast.

6.2.1.4 Réplication et mode déconnecté

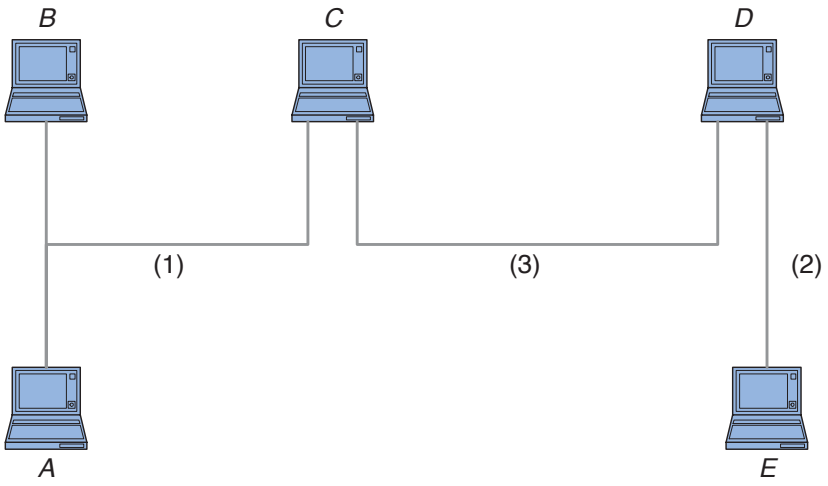


FIG. 6.6 – Réplication épidémique

Nous avons supposé jusqu'alors que les différents utilisateurs étaient en permanence connectés au réseau. Dans ce cas, dès qu'une action est produite sur une réplique d'un objet partagé, l'ensemble des autres répliques de l'objet partagé peut être mis à jour afin de refléter les changements imposés par ladite action. Cependant, il peut être intéressant pour l'utilisateur de pouvoir travailler sur un objet partagé *via* sa réplique locale, alors qu'il lui est impossible de se connecter au réseau. L'utilisateur travaille alors sur sa réplique qui reste désynchronisée de toutes les autres : l'utilisateur propagera *ultérieurement* ses modifications, dès lors qu'une nouvelle connexion réseau sera disponible.

La technique du multicast n'est alors plus satisfaisante en mode déconnecté car elle peut conduire à des problèmes de concurrence *au fil du temps*. La figure 6.6 donne un exemple de problème typique du mode de travail déconnecté. Les cinq postes de travail A à E collaborent à l'édition d'un document partagé d :

1. Dans un premier temps, les postes A, B et C sont connectés au réseau, les postes D et E étant déconnectés. Ces trois postes peuvent synchroniser leurs répliques respectives d_A, d_B et d_C de d de telle sorte que $d_A = d_B = d_C$.
2. Dans un second temps, les postes A, B et C sont déconnectés du réseau, les postes D et E étant maintenant connectés. Les deux postes D et E peuvent maintenant synchroniser leurs répliques de telle sorte que $d_D = d_E$.
3. Dans un troisième et dernier temps, seuls les postes C et D sont connectés au réseau. Ils peuvent alors tenter de synchroniser leurs répliques de manière à avoir $d_C = d_D$. Ceci nécessite une *concertation* afin de prendre en considération la fait que $d_C \neq d_D$ à l'issue du second temps.

Nous voyons sur cet exemple que l'objet partagé d n'est pas globalement consistant à la fin de chacun des trois temps. Par ailleurs, la phase de concertation (troisième temps) n'est pas aisée à réaliser puisque les postes A et E à l'origine des modifications sont déconnectés. Cette phase oblige alors à définir des points de rencontre entre les différents utilisateurs afin de pouvoir effectuer des validations *après coup*.

La notion de la propagation *épidémique* de Demers et al. (1987) répond au problème du routage quand les déconnexions sont autorisées. Dans une approche épidémique de la

réplication, les modifications se propagent de proche en proche aux différents postes au fur et à mesure de leur connexion sur le réseau, à la façon dont une épidémie se propage. Dans l'exemple de la figure 6.6, une telle propagation s'effectue selon trois temps. Bien entendu, cette approche ne saurait résoudre le problème de la non cohérence qui est intrinsèque au mode déconnecté. Par contre, elle garantit la *convergence* vers la cohérence globale au fur et à mesure que les postes se connectent sur le réseau.

L'approche épidémique est bien adaptée aux collecticiels asynchrones. Le système DOORS est un bon exemple de boîte à outils dédiée à la collaboration asynchrone qui utilise un routage épidémique des différentes versions des objets partagés (Preguiça et al., 2000). Cependant, l'utilisation du mode déconnecté dans un approche synchrone rejoint, du point de vue des concertations, celle relative à la collaboration asynchrone ; nous soulignerons ce points commun dans la section 6.2.2. Saito & Shapiro (2002) proposent une synthèse très exhaustive du problème de la convergence dans une approche répliquée, laquelle illustre la similitude entre la collaboration synchrone et la collaboration asynchrone. Étant donnée la complexité du problème, nous préférons nous cantonner à un routage de la réplication multicast simple, *i.e.* sans gestion du mode déconnecté. Nous envisagerons l'approche épidémique ultérieurement à cette thèse.

6.2.2 Couplage des répliques

Le couplage précise dans quelle mesure une réplique d'un objet partagé est effectivement identique à l'objet partagé (Dewan & Choudhary, 1995). Tout comme la matrice des collecticiels (tableau 1.1 on page 29) met en évidence les deux axes distance et temps, le couplage se décompose indépendamment en un couplage spatial et en un couplage temporel. En effet, lorsque des utilisateurs collaborent au même endroit, les objets et leurs représentations sont nécessairement identiques alors que ces représentations *peuvent* devenir différentes quand les utilisateurs collaborent à des endroits différents. Ce constat reste semblable quand les utilisateurs collaborent au même instant ou à des instants différents. Le *degré de couplage* définit la métrique (qualitative) du couplage sur chacun des deux axes.

6.2.2.1 Couplage temporel

Le couplage temporel consiste à préciser si les différentes répliques d'un objet partagé restent semblables au fil du temps. Si le degré de *couplage temporel* est fort, les répliques sont dans des états synchronisés, c'est à dire semblables à tout instant. Lorsque ce degré devient faible, les répliques sont dans des états désynchronisés tant qu'une mise à jour n'a pas été faite. Le premier cas correspond à la facette synchrone de la collaboration à distance (section 2.2.2.1), et le second cas à sa facette asynchrone (section 2.2.2.2).

La figure 6.7 illustre la notion de synchronisation des répliques. Le rectangle rouge, situé à l'intérieur d'un groupe d'objets partagés, est translaté sur le poste A. Cette action est alors répliquée sur le poste B de manière à maintenir la synchronisation entre les deux répliques du rectangle. Nous voyons sur cette figure que ce synchronisme temporel peut avoir lieu alors que les deux répliques du rectangle ne s'affichent pas dans un même *contexte* : les objets partagés n'ont pas les mêmes positions relativement au groupe, le facteur d'échelle du groupe d'objets partagés n'est pas identique et le rendu des graphiques diffère[3]. Nous reviendrons sur le couplage quant à l'affichage d'un objet partagé dans la section suivante.

Nous définissons la notion de réplique synchrone et asynchrone comme suit :

[3]L'altération du rendu est abordé dans la section 6.6.2.2.

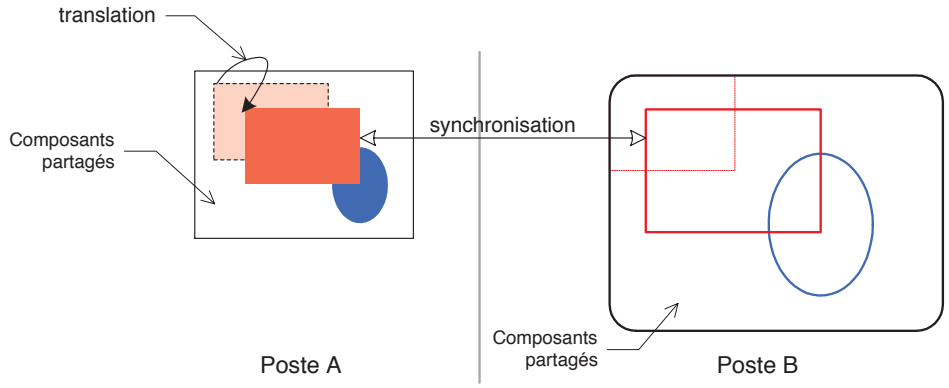


FIG. 6.7 – Synchronisation des répliques d'un composant partagé

Un objet partagé O/U définit l'ensemble $O/_{synch}U$ des répliques synchrones *et l'ensemble $O/_{asynch}U$ des* répliques asynchrones *tel que $O/U = O/_{synch}U \cup O/_{asynch}U$ et $O/_{synch}U \cup O/_{asynch}U = \emptyset$.*

Une réplique se déclare synchrone (respectivement asynchrone) en se plaçant dans l'ensemble $O/_{synch}U$ (respectivement dans $O/_{asynch}U$).

Toute action consommée par une réplique $r_i \in O/_{synch}U$ est également consommée par les autres répliques $r_j \in O/_{synch}U$, de telle sorte que $\forall(i,j) \in U^2$, $r_i = r_j$ à tout instant.

Toute action consommée par une réplique $r_i \in O/_{asynch}U$ sera consommée par les répliques $r_j \in O/_{synch}U$ dès lors que r_i se déclare synchrone.

La collaboration temps réel (ou collaboration synchrone) consiste alors à partager des objets *via* des répliques exclusivement synchrones, tandis que la collaboration en temps différé (ou collaboration asynchrone) consiste à partager des objets *via* des répliques asynchrones, lesquelles seront synchronisées à l'initiative de l'utilisateur. Il est possible d'envisager une collaboration mixte, par le biais de répliques synchrones pour certains objets partagés *et* de répliques asynchrones pour d'autres.

Notons que la définition de la synchronisation des répliques devraient être affinée afin de prendre en compte le mode déconnecté. En effet, nous voyons que, dans l'exemple de la figure 6.6 on page 157, le document partagé d/U où $U = \{A, B, C, D, E\}$ n'est jamais consistant durant les trois temps, soit $d/U = d/_{asynch}U$. Cependant, les trois sous-ensembles d/U_k de d/U, pour lesquels $U_1 = \{A, B, C\}$, $U_2 = \{D, E\}$ et $U_3 = \{C, D\}$, sont consistants, c'est à dire qu'ils contiennent des répliques synchronisées. Ils peuvent donc être notés $d/_{synch}U_k$. Ces cohérences "locales" sont importantes car elles caractérisent la possibilité qu'a le système de faire converger d/U vers la cohérence globale : elle est assurée dès lors que $d/_{synch}U = \bigcup_{k=1}^{3} d/_{synch}U_k$, c'est-à-dire lorsque les trois sous-ensembles sont synchrones entre eux.

6.2.2.2 Couplage spatial

Le couplage spatial précise dans quelle mesure le partage d'un objet implique de partager le *contexte de représentation* de cet objet. La figure 6.7 donne l'exemple d'un couplage spatial faible entre les deux représentations des répliques (synchrones) du rectangle partagé. Ce couplage est faible du fait des *contextes différents* quant à la représentation du rectangle.

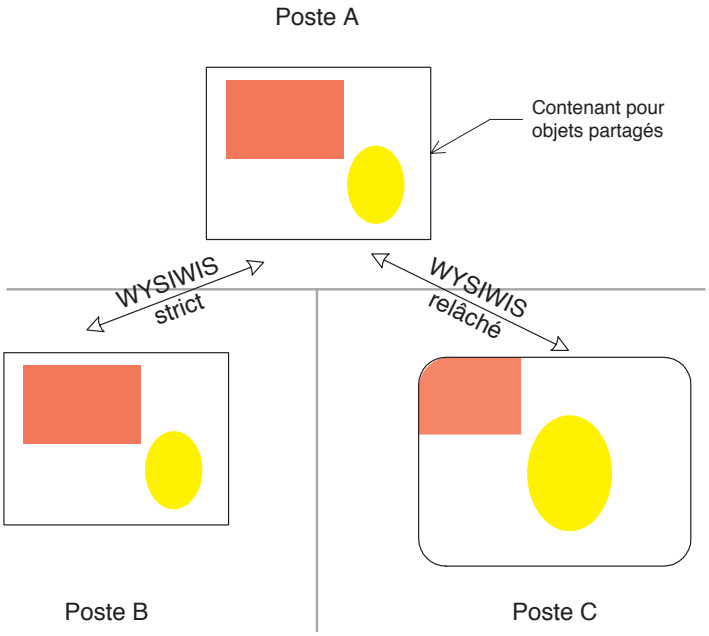


FIG. 6.8 – WYSIWIS strict et WYSIWIS relâché

Le couplage spatial permet de définir plus formellement [4] le principe du WYSIWIS[5] strict et du WYSIWIS relâché (Stefik et al., 1987). La figure 6.8 reprend l'exemple précédent du rectangle partagé dont les répliques sont maintenant réparties sur les trois postes A, B et C. Le couplage entre A et B est de type WYSIWIS strict : les deux utilisateurs voient le contenu du conteneur de la même manière ; par contre, le couplage est de type WYSIWIS relâché entre B et C : les conteneurs n'ont pas le même facteur de zoom ni la même position d'origine (le défilement n'est pas identique).

6.2.2.3 Couplage flexible

Dewan & Choudhary (1995) introduisent la notion de *couplage flexible*, c'est-à-dire d'un degré de couplage que l'utilisateur peut faire varier selon ses intentions. Les auteurs fournissent un modèle qui précise comment le couplage peut être défini entre l'objet partagé de référence et ses répliques.

Il est tout à fait possible d'imaginer, pour le couplage temporel flexible, une collaboration *via* des objets partagés dont le degré de couplage est modifiable par l'utilisateur. Par exemple, la synchronisation des répliques d'un objet partagé peut avoir lieu à intervalle de temps régulier, ou bien lorsqu'un nombre prédéfini d'actions effectuées sur l'objet partagé été atteint. La collaboration synchrone correspond alors au cas d'une synchronisation avec une granularité la plus fine possible, c'est-à-dire que la synchronisation est effectuée dès lors que l'objet est modifié. La collaboration asynchrone correspond, quant à elle, au cas d'une synchronisation à granularité élevée, cette synchronisation étant laissée à l'initiative de l'utilisateur et pouvant ainsi être tout à fait occasionnelle.

[4] Nous verrons plus loin que le degré de couplage spatial peut être formellement caractérisé par la position relative au graphe du contenant des objets partagés : plus sa position est élevée, *i.e.* plus il est proche de la racine du graphe de scène, plus le couplage est fort.

[5] What You See Is What I See

Par ailleurs, il est possible d'imaginer de faire varier le couplage spatial flexible. Dans l'exemple de la figure 6.7, le couplage spatial peut être augmenté en rendant les graphiques partagés d'une manière identique d'une part, et en imposant des positions identiques pour ces graphiques relativement aux contenants d'autre part. Si le degré de couplage spatial est fort, les représentations de l'objet sont fortement semblables du point de vue des utilisateurs. Lorsque ce degré devient faible, les représentations peuvent différer alors que l'état de l'objet représenté reste identique (le couplage spatial implique la synchronisation des répliques).

Nous retrouverons le côté flexible des couplages temporel et spatial dans les scénarios typiques de la collaboration que nous abordons dans la section suivante.

6.3 Modèle de partage des composants DPI

Nous précisons dans cette section comment les aspects de l'architecture répliquée décrits dans la section précédente sont mis en œuvre dans le modèle DPI et la boîte à outils OpenDPI. En particulier, nous présentons le concept de point de partage et sa réalisation concrète dans OpenDPI, et nous montrons comment le concept de place peut être réalisé par le biais des points de partage.

6.3.1 Réplication des composants

Dès lors qu'un objet préexistant devient partageable par des utilisateurs, il doit pouvoir être répliqué sur leurs postes respectifs. Cette réplication s'effectue en deux temps : la construction initiale des différentes répliques, puis la mise à jour des répliques au fur et à mesure où des actions y sont effectuées.

Le premier point concerne la sérialisation des composants DPI, laquelle inclut la sérialisation de la structure composite et des propriétés des composants, ainsi que leurs relations d'observation. Le mécanisme utilisé dans OpenDPI reste classique et n'est pas détaillé ici.

Le second point concerne la *réplication des actions* : toute action produite sur la réplique d'un objet partagé O/U doit pouvoir être (re)produite sur les répliques de $O/_{synch}U$.

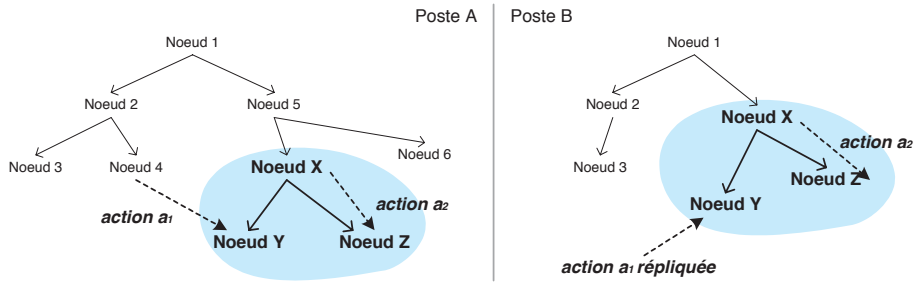


FIG. 6.9 – Modèle pour la réplication des actions

Le processus de réplication d'une action se réalise différemment selon les deux contextes possibles reproduits sur la figure 6.9. Les deux répliques, présentes sur les postes A et B, sont constituées de trois nœuds X, Y et Z définissant l'objet partagé. Deux actions $a_1$et a_2 sont produites, sur la réplique gauche, par les nœuds respectifs 4 et X, et consommées par les nœuds respectifs Y et Z. Elles ne sont par contre pas produites de la même manière sur le poste B :

– L'action a_1 est répliquée, c'est à dire quelle est (re)produite sur le poste B pour être consommée par le nœud Y.
– Par contre, l'action a_2 est inductivement répliquée du fait que sa source est le nœud X et que ce nœud est lui-même répliqué : il produira lui-même l'action a_2 vers Z. La réplication de l'action ne nécessite donc pas, dans ce cas, de transport réseau.

Nous disons que les actions a_1 et a_2 sont respectivement externes et internes à l'objet partagé. Nous donnons la définition suivante de la réplication d'une action :

> *La* réplication d'une action a, *produite sur une réplique d'un objet partagé* O, *consiste à reproduire* a *sur les autres répliques de* O. *La reproduction de l'action est* effective *lorsque l'action est externe à l'objet partagé, ou bien est* inductive *lorsque l'action reste interne à l'objet partagé*.

Notons que la reproduction d'une action consiste à reproduire l'intégralité du cycle de production de l'action, c'est à dire à invoquer certaines des méthodes définies dans son interface de consommation.

Nous donnons un exemple de réplication des actions dans la section suivante.

6.3.2 Point de partage d'un composant

6.3.2.1 Définition

Un point de partage est un élément attaché à un composant DPI et qui permet de partager, par réplication, le *contenu* de ce composant.

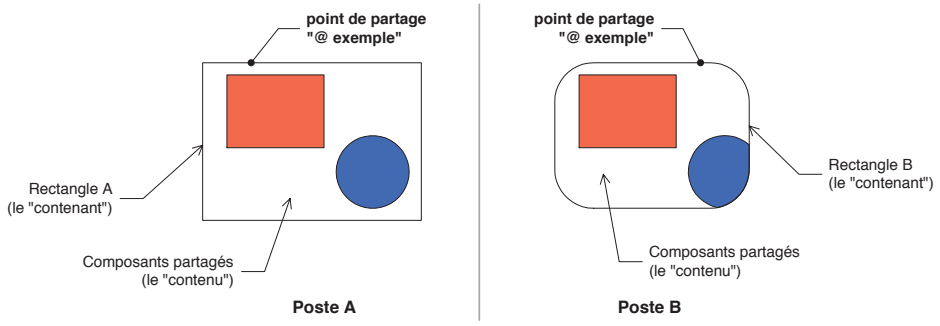


FIG. 6.10 – Exemple de point de partage

La figure 6.10 illustre le principe du point de partage sur un exemple simple. Sur le poste A, le rectangle A est un composant DPI présent sur l'espace de travail et auquel est attaché un point de partage dont l'identifiant réputée unique est "@ exemple". Le rectangle et le cercle contenus dans le rectangle A sont, de par l'attachement du point de partage au rectangle A, répliqués sur l'ensemble des postes qui utilisent le même point de partage (même identifiant). Ainsi, sur le poste B, le rectangle B étant attaché au point de partage "@ exemple", les éléments qu'il contient sont identiques à ceux du rectangle A. Il est important de noter que, si les contenus des deux rectangles A et B sont identiques, les "contenants" n'ont pas nécessairement les mêmes caractéristiques (le rectangle B est plus petit que le rectangle A et possède des coins arrondis).

Nous définissons le point de partage comme suit :

Un point de partage *est une entité identifiée de manière unique par son nom et qui prend en charge la réplication des composants partagés. Un point de partage peut être localement* attaché *à un composant DPI qui joue alors, sur le poste en question, le rôle de* contenant *: l'ensemble des composants* contenus *dans ce contenant sont partagés sur tous les autres postes qui mettent en jeu un point de partage de même nom.*

Le point de partage assure la synchronisation des éléments de son contenant : la gestion de la construction initiale des différentes répliques, la réplication des actions ainsi que la réplications des ajouts et retraits de composants dans le contenant attaché.

6.3.2.2 Réalisation

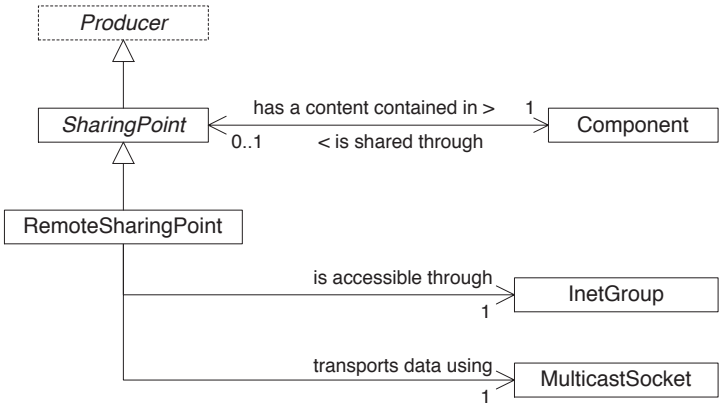


FIG. 6.11 – Modèle du point de partage

La figure 6.11 fournit le modèle d'un point de partage. Nous y retrouvons le fait qu'un point de partage est associé à un contenant (typiquement un graphique) qui définit le contenu du point de partage (classe abstraite SharingPoint). Le lien inverse représente la capacité pour un composant à être partagé lorsqu'un de ses ancêtres est associé à un point de partage (induction du partage, section 6.3.3). Il existe deux types de point de partage : le point de partage local (classe LocalSharingPoint, non représentée) et le point de partage distant (classe RemoteSharingPoint). Le premier sera abordé dans la section 6.6 et le second correspond au point de partage que nous présentons dans cette section. Le point de partage distant possède un nom unique qui est une adresse IP de groupe multicast (section 6.2.1.3), et il utilise un "socket" pour le transport des messages sur le réseau. Notons qu'un point de partage hérite de la classe Producer puisqu'il prend en charge la reproduction des actions externes dont il devient alors le producteur.

Le point de partage prend en charge la réplication des actions en reproduisant les invocations des méthodes de notification de début, de notification de fin, d'exécution, d'annulation et de ré-exécution, à l'intérieur du composant contenant associé. Dans la figure 6.9, le point de partage est associé au nœud 5 pour le poste A et le nœud 1 pour le poste B. Le producteur de l'action a_1 étant externe au contenant représenté par le nœud 5, le point de partage reproduit alors l'action sur le nœud Y du poste B.

Un objet de l'espace personnel d'un utilisateur devient alors un objet partagé dès lors qu'il est déplacé à l'intérieur du composant contenant associé au point de partage. Le point de partage prend alors en charge la création des répliques sur tous les postes accédant au même point de partage. Dans l'exemple de la figure 6.9, le nœud X (puis inductivement les nœuds

Y et Z) sont répliqués sur le poste B dès lors que l'utilisateur l'a placé sous le nœud 5, ce dernier étant le contenant du point de partage.

Les classes SharingPoint et RemoteSharingPoint sont implantées dans OpenDPI. Cependant, les mécanismes de gestion de la concurrence et de la cohérence (section 6.4) ne sont pas intégrés dans la classe RemoteSharingPoint. L'écho et la localisation (section 6.5) y sont par contre présents par des mécanismes *ad hoc*.

6.3.2.3 Point de partage et place

Nous pouvons constater que le concept de point de partage est analogue au concept de place (section 6.1), puisqu'une place joue également le rôle d'un contenant pour objets à partager. Cependant, la notion de point de partage concerne le niveau développeur tandis que la notion de place concerne le niveau utilisateur (et le concepteur). Une place serait à réaliser en définissant d'une part un composant graphique représentant la place et, d'autre part, en attachant à ce composant un point de partage. Comme nous le verrons dans les sous-sections suivantes, un point de partage définit (entre autre) le mode de synchronisation des répliques (mode synchrone ou asynchrone). Par conséquent, le réalisation d'une place devra définir d'autres points de partage pour chaque élément de la place, de manière à pouvoir partager ces éléments dans des modes variés.

Nous n'avons pas implanté dans OpenDPI le concept de place. Nous avons à la place réalisé le concept de point de partage qui en est une version simplifiée. Dans la section 6.3.4, nous considérons comment le placement du point de partage joue sur le couplage spatial des composants partagés. Ces considérations devraient être utiles pour la réalisation de le concept de place.

6.3.3 Induction du partage

Le principe de l'induction de partage est un aspect essentiel des points de partage : lorsqu'un composant devient un descendant du composant contenant attaché à un point de partage, il devient partagé. Dans l'exemple de la figure 6.9, le nœud X est partagé dès qu'il est inséré sous le nœud 5, et les nœuds Y et Z deviennent inductivement partagés puisqu'ils sont fils du nœud X. Cette propagation du partage est appelée induction du partage des composants :

> *L'*induction du partage *consiste, pour un composant donné, à propager automatiquement l'état de partage à ses composants fils*.

L'induction du partage confère aux points de partage un usage simple. Sa définition permet de déterminer si un composant est partagé :

> *Un composant est* partagé *si et seulement s'il existe, parmi ces ancêtres, un composant jouant le rôle de contenant d'un point de partage*.

Le principe de fonctionnement de la réplication des composants consiste à transmettre toutes les invocations des méthodes insertChild et removeChild faites sur la réplique d'un composant partagé aux autres répliques de ce composant *via* le point de partage. Ce principe est semblable pour la réplication des actions, laquelle consiste à transmettre les invocations des méthodes de notification de début, de fin, d'exécution, d'annulation et de re-exécution *via* le point de partage. La figure 6.12 illustre la réplication d'une action a produite par un outil et consommée par un graphique :

1. L'outil teste localement si l'action a est faisable sur le graphique.

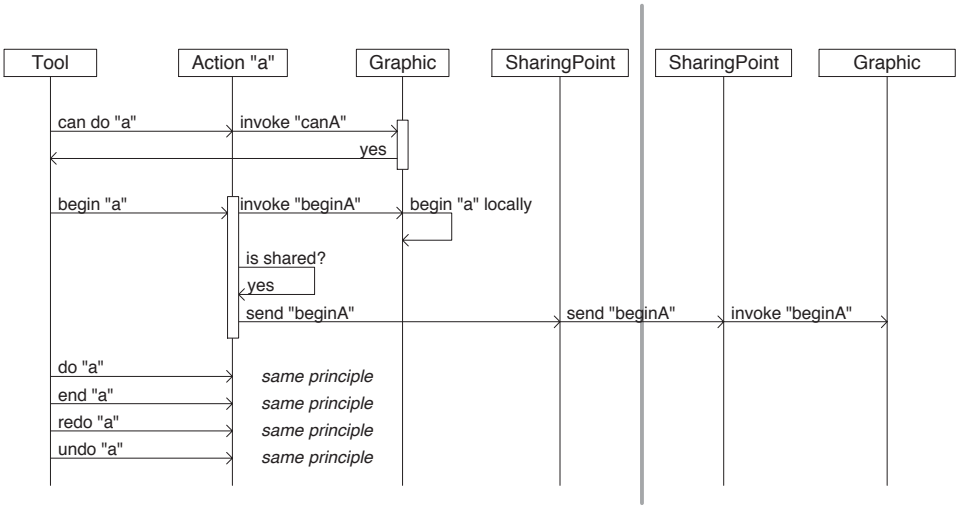


FIG. 6.12 – Principe du partage et induction

2. Si a est faisable, l'outil lui demande de démarrer l'action a en invoquant la méthode beginA sur le graphique ciblé. Le graphique verrouille alors les propriétés que l'action est susceptible de modifier et créé une entrée dans l'historique (non représenté).
3. L'action a teste si le graphique ciblé est partagé. Si tel est le cas, a envoie au point de partage gérant la réplication du graphique la demande de réplication de l'invocation beginA.
4. Le point de partage transmet alors cette demande aux autres points de partage (utilisation du groupe multicast).
5. Quand la demande est reçue par ces autres points de partage, ils localisent le graphique répliqué et y invoquent la méthode beginA.

6.3.4 Couplage spatial

Dans cette section, nous allons montrer comment le degré de couplage spatial peut varier en jouant sur la "hauteur" du point de partage relativement au graphe de scène. Nous reprenons à cet effet les exemples issus du scénario de conception (section 6.1.3). Notons que ces exemples n'ont pas été implantés dans OpenDPI.

La figure 6.13 montre comment il est possible de partager un document. Le document est placé, pour les deux postes représentés, à l'intérieur d'un composant (non visuel) qui définit le contenu du point de partage "@ exemple 1". Le document possède deux présentations Présentation 1 et Présentation 2, la première étant affichée sur le poste A et la seconde sur le poste B (les graphes de scène sont représentés en gras). Lorsque l'utilisateur côté A effectue une action sur la Présentation 1, elle est répliquée sur le poste B, ce qui met à jour les données du domaine du document et, par conséquent, la Présentation 2. Le principe reste identique quand l'utilisateur de droite effectue une action sur la Présentation 2. Nous voyons sur cet exemple que l'induction du partage doit avoir également lieu au travers des liens document / présentation de manière à ce que les présentations 1 et 2 soient effectivement partagées par induction.

Dans la figure 6.14, les utilisateurs souhaitent travailler d'une manière fortement couplée sur la même présentation du document précédent. Ils partagent alors la fenêtre qui contient leur réplique de la Présentation 1. Cette fenêtre est simplement positionnée dans un groupe

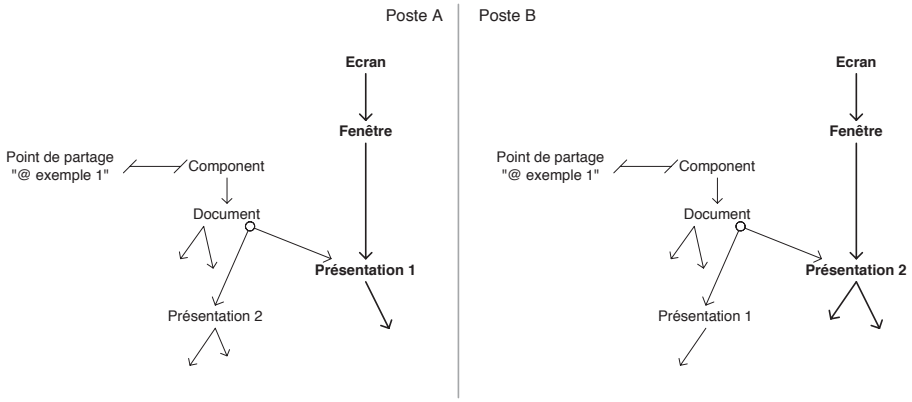


FIG. 6.13 – Partage d'un document

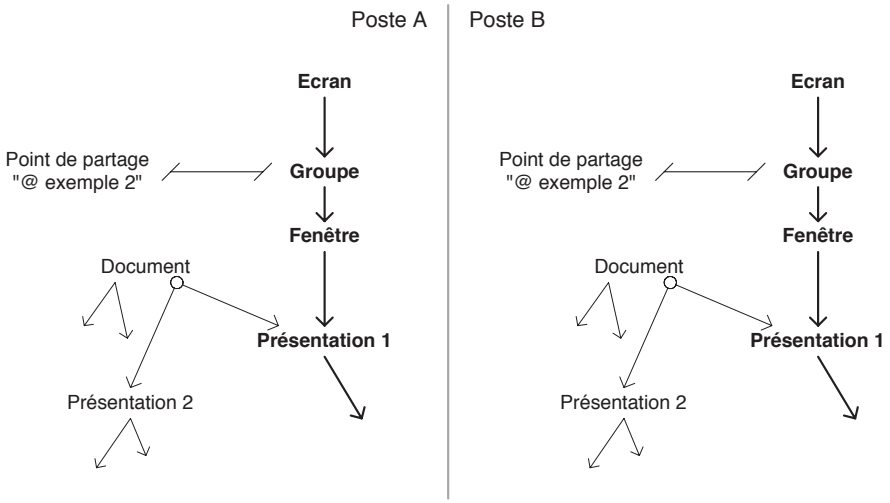


FIG. 6.14 – Partage d'une fenêtre

qui définit le contenu du point de partage "@ exemple 2". Nous voyons sur cet exemple que le partage d'une présentation doit induire, *dans le sens vertical ascendant*, le partage du document (et donc de sa seconde présentation). Le fait d'induire un tel partage simplifie le processus de réplication : les notifications liées à l'observation document / présentations sont *localement* répliquées une fois que les actions faites sur les présentations ont été répliqués au travers du réseau. Les notifications associées au mécanisme d'observation n'ont ainsi pas à être transmises sur le réseau, ce qui permet de rendre ce mécanisme indépendant des temps de transit réseau.

Dans la figure 6.15, les utilisateurs souhaitent collaborer d'une manière encore plus couplée et partagent alors un espace de travail commun. Pour se faire, il suffit d'attacher à cet espace de travail un point de partage : tout élément de l'espace de travail du poste A sera présent dans l'espace du poste B, et vice-versa.

Les exemples relatifs à ces différents cas de figure correspondent à un travail synchrone. Une collaboration asynchrone consisterait à définir l'un des points de partage comme asynchrone. Ceci permet d'envisager, dans l'exemple de la fenêtre partagée (figure 6.14), de travailler en toute discrétion sur le document présenté dans la fenêtre puis de discuter, peu

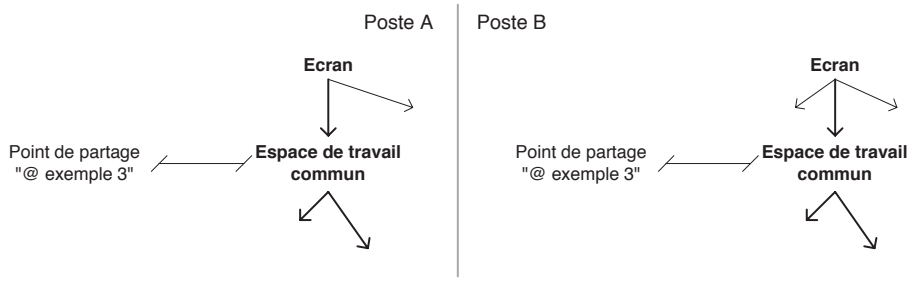


FIG. 6.15 – Partage d'un espace de travail

de temps après, avec les autres utilisateurs des modifications apportées.

6.3.5 Couplage temporel

Le point de partage prend en charge la synchronisation des répliques en définissant s'il fonctionne de manière synchronisée ou non. Le point de partagé est dit *synchronisé* s'il assure que l'ensemble des répliques qu'il contient soient synchronisées. Inversement, le point de partage est dit *désynchronisé* s'il assure que l'ensemble des répliques désynchronisées pourront être ultérieurement synchronisées.

Du fait de l'induction du partage, le synchronisme d'une réplique est induit sur l'ensemble de tous ses descendants. Nous pouvons donc parler d'induction du synchronisme. Par conséquent, du fait que le point de partage impose lui-même le synchronisme, tous les descendants du composant contenant d'un point de partage ont le même niveau de synchronisme.

La figure 6.16, qui complète la figure 6.12, illustre la manière dont les points de partage gèrent le synchronisme des répliques :

1. Le point de partage Pa fonctionne en asynchrone et le point Pb en synchrone :
   (a) L'outil exécute une action sur le graphique répliqué.
   (b) Le point de partage Pa qui contient cette réplique étant désynchronisé, les points de partage distants ne sont pas sollicités et l'action est sauvegarde dans une pile.
   (c) Lorsque Pa passe en mode synchronisé, les différentes actions sauvegardées dans cette pile sont transmises aux points de partages distants.
   (d) Le point de partage Pb étant synchronisé, il (re)produit les actions reçues immédiatement sur les répliques (synchrones) qu'il contient.
2. Le point de partage Pa fonctionne en synchrone et le point Pb en asynchrone :
   (a) L'outil exécute maintenant une autre action sur le même graphique répliqué au travers du même point de partage Pa.
   (b) Le point de partage Pa étant maintenant synchronisé, les points de partage distants sont sollicités.
   (c) Le point de partage Pb étant maintenant désynchronisé, l'action reçue est sauvegardée dans une pile.
   (d) Lorsque Pb se synchronise, les actions sauvegardées localement sont produites sur la réplique du graphique partagé.

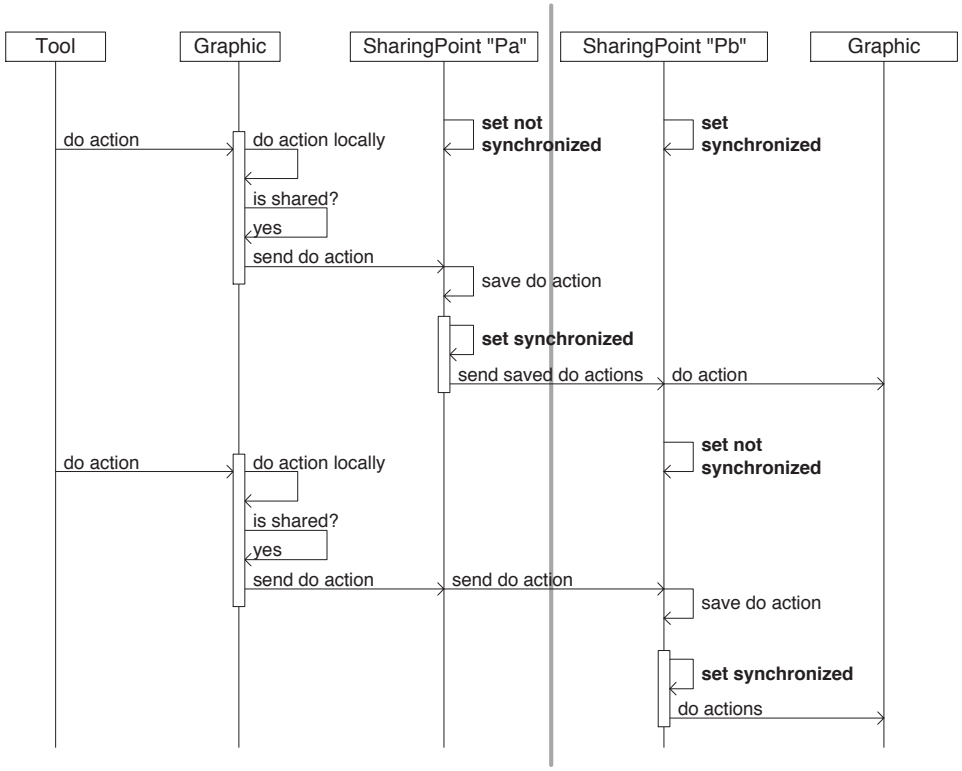


FIG. 6.16 – Réplication et synchronisation

Ce principe est très simple à implanter dans OpenDPI mais, en contre-partie, il implique d'être connecté au réseau pour travailler en "mode asynchrone" sur un objet partagé. En effet, lors de la phase 2.(c) précédente, le point de partage Pa étant en mode synchrone, il envoie au point de partage Pb les actions qui y sont faites : le point Pb sauvegarde ces actions puisqu'il fonctionne en mode asynchrone. Le poste côté Pb *doit* être connecté au réseau car le point Pa n'a *pas* à savoir si Pb est en mode asynchrone ou non : c'est donc bien à Pb de sauvegarder les actions faites du côté de Pa en attendant la prochaine synchronisation.

Ceci est une limite forte qui peut paraître paradoxale dans une approche asynchrone de la collaboration. Toutefois, nous rappelons que nous n'implantons pas le mode déconnecté dans le cadre de cette thèse. Nous jugeons que l'aspect asynchrone du travail collaboratif devrait être géré d'une manière "unifiée" avec la gestion du mode déconnecté, en considérant le cas asynchrone comme un cas particulier du mode déconnecté : dans les deux cas, il s'agit de pouvoir travailler d'une manière déconnectée (ou désynchronisé) sur les répliques (*i.e.* sur l'ensemble $O/_{synch}U$) ainsi que de pouvoir effectuer des "rendez-vous" de manière à pouvoir résoudre les conflits après-coup.

6.4 Concurrence et cohérence

Nous présentons dans cette section l'approche de la concurrence d'accès et la cohérence des composants répliqués dans le modèle DPI. Notons que les principes évoqués ne sont pas implantés dans les points de partage d'OpenDPI.

6.4.1 Problème

Nous avons défini dans la section 6.2.2 la cohérence globale d'un objet partagé comme caractérisant le fait que *toutes* ses répliques sont identiques. En utilisant les notations introduites dans le couplage temporel, nous avons :

> O/U *est* globalement consistant *si et seulement si* $O/U = O/_{synch}U$ *(ce qui équivaut à* $O/_{asynh}U = \emptyset$*).*

Cette définition de la cohérence globale de l'objet partagé est caractéristique du mode déconnecté : dès qu'il y a modification d'un objet partagé alors qu'au moins un poste possédant une réplique de cet objet partagé n'est pas connecté au réseau, il y a *nécessairement* incohérence globale de cet objet. Aucun système ne saurait garantir la cohérence globale des objets partagés en mode déconnecté. Il se doit par contre d'en *garantir la convergence*.

La définition de l'ensemble $O/_{synch}U$ (section 6.2.2) indique que toutes les répliques de cet ensemble sont identiques. Cependant, lorsqu'une action est produite sur $r \in O/_{synch}U$ et, par conséquent, (re)produite sur l'ensemble $O/_{synch}U - \{r\}$, il peut arriver que certaines répliques synchrones de $O/_{synch}U$ ne soient plus égales entre elles, contredisant du même coup la définition de cet ensemble $O/_{synch}U$. Nous parlons dans ce cas d'*incohérence partielle* (ou, par abus de langage, d'incohérence) de l'objet partagé. Si le système ne peut garantir la cohérence globale, il est *nécessaire* qu'il garantisse la cohérence (partielle), *i.e.* que les répliques synchronisées puissent rester identiques tant qu'elles ne se désynchronisent pas de l'objet partagé.

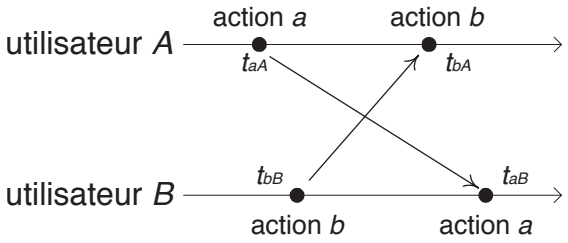


FIG. 6.17 – Actions croisées

Une telle situation d'incohérence peut se produire du fait du temps de transport réseau lors de la réplication des actions. La figure 6.17 donne un exemple typique d'un transport réseau générant une incohérence de l'objet partagé. L'utilisateur A réalise une action a sur l'objet partagé O, laquelle est exécutée localement et transmise sur le réseau à l'instant t_1. De même, l'utilisateur B réalise une action b sur cet objet O, laquelle est exécutée et transmise à l'instant t_2 proche de t_1. Le laps de temps $t_2 - t_1$ restant petit devant le temps de transit réseau, l'utilisateur B reçoit l'action de A après l'instant t_2, action qui est alors appliquée localement. Il en résulte que les deux actions de A et de B ne sont pas effectuées dans le même ordre sur les deux répliques synchrones de l'objet O. Le fait d'inverser l'ordre de la consommation des actions a et b peut alors conduire à une incohérence partielle. Ceci est en particulier vrai si les actions considérées ne sont pas commutatives[6].

Pour notre modèle des actions, nous devons prendre en compte l'étalement temporel lié à la production d'une action sur la cible consommatrice (figure 6.18). Nous analyserons en particulier les différents cas de figure lorsque des actions "longues" sont répliquées au travers le réseau (section 6.4.4).

[6]Deux actions a et b sont dites commutatives lorsque, pour tout objet O, l'application de a puis b sur O donne un résultat identique à l'application de b puis a sur O.

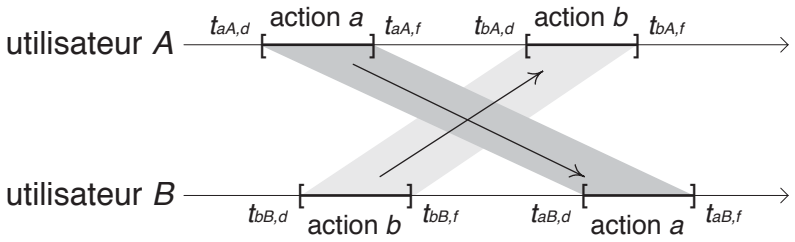


FIG. 6.18 – Actions “longues” croisées

6.4.2 Solutions existantes

Le problème de l’incohérence se résout par une gestion de la concurrence d’accès aux répliques. Dans le cadre des collecticiels temps réel, il existe deux méthodes pour gérer la concurrence d’accès aux objets partagés, la sérialisation et le verrouillage, lesquelles peuvent être déclinées en deux approches, l’approche pessimiste ou l’approche optimiste (Greenberg & Marwood, 1994). En ce qui concerne les collecticiels asynchrones, nous aborderons la gestion des conflits dans la section 6.4.4.

6.4.2.1 Sérialisation

La gestion par sérialisation se propose d’assurer le bon ordre des actions selon deux approches différentes : l’approche pessimiste et l’approche optimiste.

Dans l’approche pessimiste, il est supposé que les actions sont susceptibles de parvenir sur les postes dans un mauvais ordre et vise alors à *empêcher* cela. Pour ce faire, le système va “sérialiser” les différentes actions des utilisateurs au niveau d’un poste qui joue le rôle de “coordinateur” : chacune des actions est d’abord reçue par le coordinateur sans être exécutée localement, puis elle est éventuellement réordonnée pour être finalement envoyée sur les différents postes afin d’y être exécutée. Cette approche à pour défaut majeur la lenteur d’exécution des actions. En effet, les actions faites par un utilisateur ne se répercutent pas immédiatement sur son poste puisqu’elles sont d’abord propagées vers le coordinateur. Ainsi le choix de la sérialisation pessimiste dans une architecture répliquée fait perdre l’avantage même de cette architecture, alors qu’il devient naturel dans une architecture centralisée puisque le serveur peut jouer naturellement le rôle de coordinateur.

Dans l’approche optimiste, il est postulé que les conflits ont assez peu de chance de se produire et qu’il est plus efficace de tenter de *réparer* les éventuels problèmes d’ordre que de vouloir en permanence les éviter. Dans cette approche, les actions sont reçues dans un ordre éventuellement erroné et, si besoin, sont réordonnées *localement* en utilisant un mécanisme d’annulation et de ré-exécution des actions. Cette approche ne concerne que l’architecture répliquée. L’application GroupDesign utilise une sérialisation optimiste dans une architecture répliquée (Beaudouin-Lafon & Karsenty, 1992). Elle propose notamment une optimisation des annulations /ré-exécutions basée sur la commutativité des opérations, c’est-à-dire la possibilité pour certaines opérations de s’effectuer dans un ordre quelconque, le résultat final demeurant identique (Karsenty & Beaudouin-Lafon, 1993). Néanmoins, il n’est pas toujours possible de réordonner des actions automatiquement. Dans ce cas, le système laisse le choix à l’utilisateur quant à la réplique à effectivement conserver.

6.4.2.2 Verrouillage

La gestion de la concurrence par *verrouillage* est un autre moyen de garantir le bon ordre des actions et donc la cohérence des répliques des objets partagés. Un objet partagé est modifiable uniquement si le demandeur a obtenu un verrou de la part du coordinateur. Dans le cas où un demandeur désire acquérir un verrou pour un objet déjà verrouillé, il y a refus et le demandeur doit attendre la libération du verrou actuellement posé sur l'objet. La politique de verrouillage est qualifiée de pessimiste quand aucune action ne peut être effectuée tant que la demande d'acquisition du verrou n'a pas donné satisfaction. Elle est qualifiée d'optimiste lorsque le système autorise l'exécution *locale* de l'action avant de recevoir la notification d'acceptation ou de refus du verrouillage[7]. Dans le cas d'un refus, l'application a alors en charge d'annuler localement l'action exécutée. La motivation de la version optimiste est de fournir un temps de réponse du système plus acceptable que dans la version pessimiste.

La notion de verrou est simple à implanter dans une architecture centralisée puisqu'un verrou peut être physiquement positionné sur l'objet de référence situé sur le serveur. Cette simplicité d'implantation disparaît notablement dans le cas d'une architecture répliquée puisqu'aucun poste n'a le statut spécifique de "coordinateur". Il est donc relativement complexe et coûteux de répliquer le verrouillage des objets.

6.4.3 Solution retenue

Dans notre modèle, nous optons pour une architecture répliquée et, en conséquence, pour une gestion de la concurrence d'accès optimiste basée sur la sérialisation dans le cas de la collaboration à distance. Notons que l'approche optimiste vise à *éviter* le plus possible les accès concurrents. Afin de prévenir de tels risques, il est essentiel de soigner tout particulièrement la restitution de la conscience mutuelle (section 2.2.3.3). De plus, nous veillons à ce que cette gestion de la concurrence d'accès à distance reste cohérente avec celle locale basée sur le verrouillage des propriétés.

6.4.3.1 Sérialisation optimiste

Le ré-ordonnancement des actions, nécessaire pour garantir la cohérence partielle des objets partagés, concerne le ré-ordonnancement inter-actions ou intra-action.

Le ré-ordonnancement *inter-action* est relatif au cas présenté sur la figure 6.17 : les deux actions simultanées `a` et `b` doivent être ordonnées. Le processus se déroule alors en deux phases : la détection d'un ordre erroné puis la remise en ordre. La détection d'un ordre erroné utilise la datation par les horloges logiques de Lamport (section 6.4.3.2). Le remise en ordre est basée sur les mécanismes d'annulation, lesquels ont été présentés dans la section A.2, ainsi que la notion d'ordre total des horloges de Lamport.

Le ré-ordonnancement *intra-action* est relatif au cas présenté sur la figure 6.19 : l'action "longue" est répliquée de A vers B avec un synchronisme erroné des différentes invocations de la commande associée, voire une perte de demandes d'invocation. Nous parlons dans ce cas d'*incohérence de l'action* reproduite. Les points de partage doivent nécessairement garantir la cohérence des actions qu'ils reproduisent sur les répliques. Notons que cet aspect n'a aucun rapport avec la gestion de la concurrence d'accès puisqu'une seule action est ici concernée. La solution à ce problème est similaire au ré-ordonnancement inter-actions en ce

[7] Dans (Greenberg & Marwood, 1994), il est fait mention des trois niveaux d'optimisme : pessimiste, semi-optimiste et pleinement optimiste. Nous avons ici simplifié dans un unique but de clarté, le principe restant inchangé.

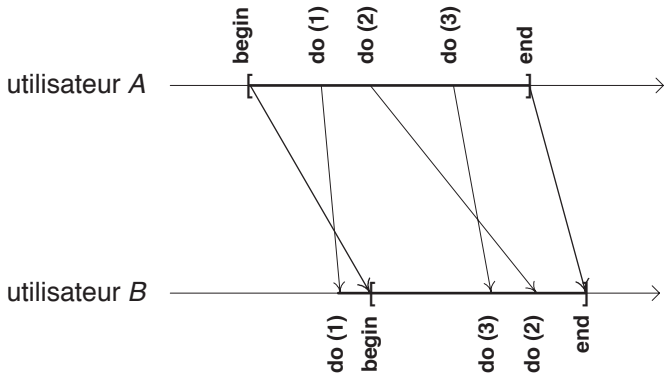


FIG. 6.19 – Action non consistante

qui concerne la détection d'un ordre erroné. Cependant, la solution pour le remise en ordre ne peut être basée sur l'annulation des invocations des différentes méthodes : dès qu'un ordre erroné est détecté, les différentes invocations sont enregistrées sans être réalisées, et leur réalisation sera effectuées dans le bon ordre une fois la méthode de notification de fin reçue. Notons que ce problème est induit par l'utilisation du routage "multicast" en mode "non connecté", *i.e.* selon le protocole de transport UDP (Wittmann et al., 2000).

6.4.3.2 Horloges logiques

Il est très difficile que des postes géographiquement répartis sur un réseau puissent avoir des horloges "physiques" synchronisées. Cette synchronisation des horloges est presque impossible car les temps de latence réseau, variables et non déterministes, n'autorisent pas l'envoi de tops de synchronisation fiables. De plus, une horloge peut s'avérer localement trop imprécise pour déterminer l'ordre des actions. Ainsi, ce qui importe, c'est l'idée d'une horloge *logique* qui permette de mesurer l'*ordre* des actions de telle sorte que cet ordre soit le même pour tous.

Lamport (1978) définit mathématiquement une *horloge logique* qui permet de caractériser la relation d'ordre entre les événements d'envoi et de réception. L'horloge logique de Lamport, définie pour chaque poste P_i concerné par la réplication, est un nombre H_i initialement nul et mis à jour par P_i au fur et à mesure de la production des événements. Le calcul des H_i doit respecter les deux règles suivantes :

1. Si les événements a et b ont lieu sur un même poste P_i tel que a précède b alors $H_i(a) < H_i(b)$ [8].
2. Si l'événement a correspond à l'envoi d'une action depuis le poste P_i et l'événement b correspond à sa réception sur le poste Pj alors a précède b et $H_i(a) < H_j(b)$.

La règle 1 précise que l'horloge doit prendre en compte l'ordre des événements locaux à chaque poste. La règle 2 précise que les horloges doivent prendre en compte le temps de transit sur le réseau. L'implantation de ces deux règles se fait simplement par l'algorithme suivant :

[8] $H_i(e)$ représente la valeur de H_i au moment où e se produit sur P_i.

– Quand un nouvel événement e_i (de réception ou d'envoi d'une action) se produit dans P_i, H_i est incrémenté :

$$H_i \leftarrow H_i + 1$$

– Si l'événement e_i correspond à l'envoi d'une action depuis P_i, le message d'envoi est *estampillé* par H_i :

$$H(e_i) \leftarrow H_i$$

– Sinon, la réception de l'action sur P_i provient de l'événement d'envoi e_j depuis P_j et l'horloge est mise à jour comme suit[9] :

$$H_i \leftarrow max(H(e_j) + 1, H_i)$$

Cette horloge permet, par l'estampille insérée dans les messages, de fournir l'*ordre partiel* des événements. Par exemple, dans la figure 6.17 on page 169, l'estampille associée à la réception de l'action a sur le poste B a la même valeur que l'estampille de l'action b émise (vers A) par B. De manière à pouvoir ordonner correctement les événements, il est alors nécessaire de définir un ordre arbitraire associé à chaque poste. Lamport (1978) propose de définir un *ordre total* à partir de l'ordre partiel et d'un ordre arbitraire affecté aux différents postes de la manière suivante :

1. Les postes P_i sont ordonnés arbitrairement avec la relation $<$.
2. Si un événement a se produit sur le poste P_i et l'événement b se produit sur le poste P_j tel que $H_i(a) = H_j(b)$, alors a précède b si et seulement si :

$$P_i < P_j$$

La relation du point 2 permet ainsi d'ordonner les événements, quand ils sont concurrents, en fonction de l'ordre attribué aux différents postes. L'algorithme global de vérification de l'ordre des événements devient donc :

> *Si un événement a se produit sur le poste P_i et l'événement b se produit sur le poste P_j, a précède*[10] *b si et seulement si :*
>
> $$H_i(a) < H_j(b) \; ou \; (H_i(a) = H_j(b) \; et \; P_i < P_j)$$

Nous utilisons ce type l'horloge dans notre implantation de l'architecture répliquée. Notons cependant que les horloges logiques de Lamport ne sont pas adaptées aux infrastructures de réseau dynamiques ("mobile computing") ainsi qu'au mode déconnecté. Il devient dans ce cas nécessaire de définir d'autres horloges dont la réalisation peut devenir complexe. Citons les horloges hiérarchiques de Prakash & Singhal (1997) qui sont une extension des horloges vectorielles de Fidge (1988) et Mattern (1989) aux infrastructures dynamiques, elles-même extension des horloges de Lamport (1978).

6.4.3.3 Cohérence de l'interaction

Si nous avons rejeté une gestion de la concurrence d'accès basée sur le verrouillage distant, nous avons opté pour un verrouillage *local* afin de garantir la cohérence (locale) de l'interaction. Ceci nous amène alors à soulever deux questions :

1. La cohabitation entre le verrouillage local et la sérialisation optimiste est-elle possible ?

[9] N'oublions pas que H_i a préalablement été incrémenté au point 1.

[10] *au sens de l'ordre total*, i.e. *avec le côté arbitraire lié au choix de l'ordre des postes P_i.*

2. Si cette cohabitation est possible, dans quelle mesure peut-on conserver la cohérence de l'interaction lors d'une collaboration à distance ?

Nous pouvons répondre à la première question en considérant le cas des deux actions distantes, de même type et simultanées de la figure 6.18 on page 170. Dans le cas où le temps de transit réseau est inférieur à la durée de l'action[11], la reproduction de l'action b sur le poste A a lieu avant que l'action a ait été entièrement consommée localement en A, *i.e.* à un instant tel que $t_{bA,d} \leq t_{aA,f}$. Par conséquent, l'action b ne peut pas commencer à être produite en A puisque a n'a pas encore déverrouillé les propriétés qu'elle est susceptible de modifier. Ainsi, lorsque la méthode de notification de début sera invoquée pour b, la tentative de verrouillage va échouer. Contrairement à une tentative de verrouillage qui échoue dans le cas d'une simultanéité forte (section 5.4), il faut ici produire l'action b en A. Le problème est dû au fait que le test de faisabilité de b en A n'a pas été demandé par le poste B : nous avons délibérément rejeté une telle possibilité car elle nécessiterait une interrogation de toutes les répliques, ce qui équivaudrait à une approche pessimiste. Par conséquent, dans notre approche optimiste, nous considérons qu'une action peut être répliquée dès lors qu'elle est faisable en local : les postes où l'action est reproduite doivent donc prendre en charge les éventuels conflits liés aux verrous.

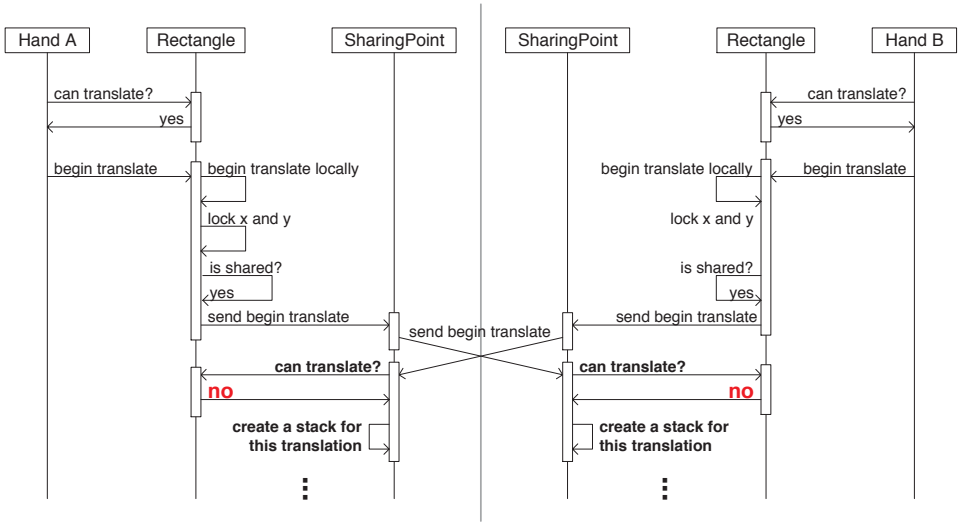


FIG. 6.20 – Cohérence de l'interaction en collaboration distante

Une solution simple est d'invoquer localement le test de faisabilité des actions à reproduire juste avant de démarrer la reproduction de l'action. La figure 6.20 illustre cette solution lorsqu'un rectangle partagé est translaté simultanément par deux outils "main A" et "main B" :

1. Les deux outils entament localement la translation du rectangle.
2. Ce rectangle étant partagé, les points de partage relatifs aux deux répliques du rectangle prennent en charge l'envoi de la demande d'invocation de la méthode beginTranslate.
3. Lorsque les points de partage reçoivent cette demande, plutôt que d'invoquer immédiatement la méthode beginTranslate, ils effectuent un test de faisabilité de la translation par une invocation de la méthode de faisabilité canTranslate.

[11] ce qui peut être souvent le cas

4. Ces invocations de canTranslate retournent une non faisabilité signifiant qu'il y a alors conflit.
5. Les points de partage définissent alors, pour l'action de translation conflictuelle, une pile qui recevra les demandes ultérieures d'invocation relatives à cette action.
6. Quand les points de partage reçoivent les demandes ultérieures, elles sont stockées dans la pile si la translation est toujours non faisable (non représenté).
7. Dès que la translation devient faisable, les différentes demandes d'invocations stockées sont alors exécutées sur la cible consommatrice (non représenté).

Cette solution gère la cohérence partielle des objets partagés, ce qui permet de répondre positivement à la première question. Nous pouvons de plus constater que l'exécution du point 7 conduit à des invocations successives de la méthode doTranslate encadrées par les invocations des méthodes beginTranslate et endTranslate. L'interaction reste donc consistante sur les deux postes A et B, à ceci près que les translations "distantes" sont jouées d'un seul tenant[12]. Notons que le système détecte le conflit et offre donc les deux alternatives (pour le concepteur) : appliquer les deux actions conflictuelles l'une après l'autre (avec une annulation / ré-exécution éventuelle en cas d'ordre non conforme), ou appliquer l'une des deux actions (choix réalisé par les utilisateurs).

Notons enfin que cette solution n'est ici qu'ébauchée et qu'elle nécessiterait un affinement pour tenir compte de situations plus complexes (nombre de sites supérieur à deux par exemple). En particulier, l'idée d'utiliser une pile dès qu'il y a conflit devrait être approfondie. Cet aspect n'a pas été implanté dans OpenDPI.

6.4.4 Asynchronisme

Les considérations précédentes concernent la collaboration synchrone. Dans le cadre de la collaboration asynchrone, les utilisateurs manipulent des répliques désynchronisées, équivalentes du point de vue utilisateur aux versions d'un objet partagé (section 6.1.3). La gestion de la concurrence concerne alors la possibilité, pour un utilisateur, d'appliquer les modifications qu'il a effectuées sur sa réplique asynchrone, *i.e.* de *resynchroniser* la réplique.

Nous abordons dans les sections suivantes les politiques possibles que le concepteur peut utiliser pour resynchroniser des répliques, ainsi que la manière dont il est possible de détecter puis de résoudre les conflits.

6.4.4.1 Politiques de synchronisation

Afin d'illustrer les différentes politiques qu'un concepteur pourrait mettre en application lors d'un conflit, nous prenons l'exemple d'un rectangle partagé dont la réplique R_B de l'utilisateur B est désynchronisée (figure 6.21) :

– L'utilisateur A effectue d'abord une modification de la structure du rectangle (réplique synchrone R_A) en lui "ôtant" son fond et en lui appliquant la couleur bleue pour son contour (action c_A). Il dimensionne ensuite ce rectangle (action d_A).
– L'utilisateur B effectue quant à lui un dimensionnement du rectangle (réplique synchrone R_B) de manière à lui donner la forme d'un carré (action d_B) et le peint ensuite en orange (action p_B).

[12] Il est cependant tout à fait possible de reproduire l'action distante en respectant son étalement dans le temps en adjoignant aux demandes d'invocations une heure physique.

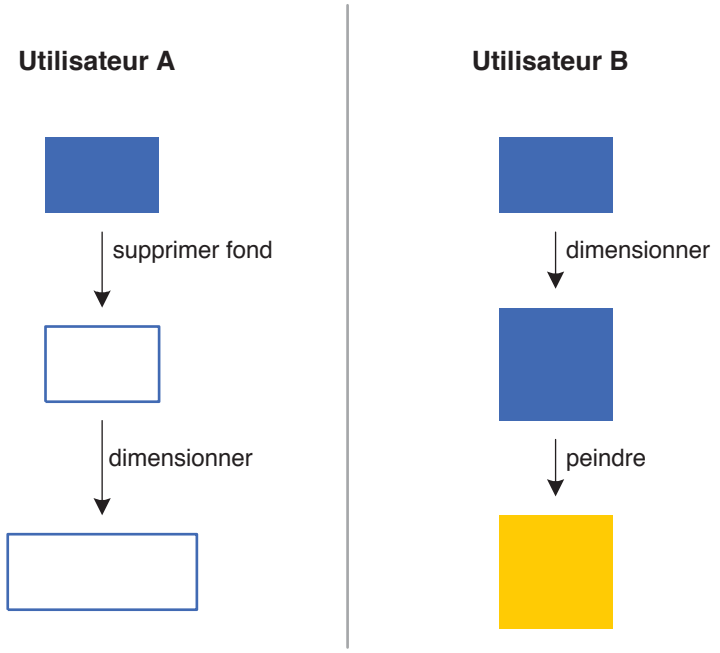


FIG. 6.21 – Concurrence en asynchrone

La synchronisation de la réplique R_B vers la réplique R_A consiste alors déterminer comment appliquer les quatre actions c_A, d_A, d_B et p_B, de manière à avoir finalement $R_B = R_A$. Cette synchronisation est conflictuelle puisque les actions manipulent des propriétés semblables. Par exemple, le fait d'appliquer c_A ne permet plus d'appliquer la couleur de fond *via* p_B puisque le fond n'existe plus, et le fait d'appliquer p_B puis c_A annule l'effet de p_B. De même, le fait d'appliquer d_A puis d_B ne fournit pas le même résultat que si d_B est appliqué avant d_A. Par conséquent, le système doit, pour chaque ensemble d'actions conflictuelles, demander quelle *action de substitution* doit être appliquée aux répliques R_A *et* R_B.

Le concepteur doit prendre en considération les différents cas de figure possibles afin de permettre aux utilisateurs de définir (d'une manière simple) quelles actions de substitution doivent être appliquées à la place des actions conflictuelles. Il doit ainsi proposer à l'utilisateur différentes *politiques de synchronisation*. Nous pouvons, par exemple, envisager les politiques suivantes :

Versionnage : La réplique asynchrone devient une version qui est alors adjointe aux répliques synchrones. Les utilisateurs peuvent ainsi résoudre les conflits par la suite (en utilisant l'une des politiques de synchronisation suivante).

Synchronisation forcée prioritaire : Les actions réalisées de manière asynchrone sont appliquées telles quelles sur les répliques synchrones, donc de manière prioritaire sur celles réalisées entre temps sur les répliques synchrones. Dans l'exemple du rectangle, cela signifie que $c + p_B$ se substitue à $\{c_A, p_B\}$ et $d_A + d_B$ à $\{d_A, d_B\}$. [13].

Synchronisation forcée non prioritaire : Les actions réalisées de manière asynchrone sont appliquées avant celles réalisées entre temps sur les répliques synchrones. Dans l'exemple du rectangle, cela signifie que $p_B + c_A$ se substitue à $\{c_A, p_B\}$ et $d_B + d_A$ à $\{d_A, d_B\}$.

[13] Le symbole $+$ dénote la succession d'actions : $a + b$ se lit *a puis b*.

Synchronisation interactive : Le système demande à l'utilisateur de préciser explicitement les actions de substitution pour chaque ensemble d'actions conflictuelles. Dans notre exemple, l'ensemble $\{c_A, p_B\}$ peut être substitué par l'action $p_B + c'_A$ qui consiste à peindre le rectangle en orange (action p_A) *puis* à lui définir un contour bleu sans pour autant supprimer le fond (action c'_A), et l'ensemble $\{d_A, d_B\}$ peut être substitué par la seule action d_B.

Cette gestion de la concurrence liée aux versions d'objet partagés (*i.e.* au répliques asynchrones) est présente dans les systèmes purement asynchrones tels que le système de fichiers Coda (Kistler & Satyanarayanan, 1992), ou le système de base de données répliquées Bayou (Edwards et al., 1997b). Quand ils détectent un problème de concurrence, ces systèmes adjoignent la version à l'objet partagé de telle sorte qu'il est possible, soit par une stratégie prédéfinie de fusion, soit par le choix de l'utilisateur, d'appliquer sur l'objet partagé certaines des actions définies dans la version.

Ces stratégies ne sont pas implantées dans OpenDPI. Il est cependant important qu'elles le soient dans le futur de manière à ce que le concepteur puisse choisir une politique parmi l'ensemble des politiques disponibles. Nous décrivons dans la section suivante comment les actions conflictuelles peuvent être détectées.

6.4.4.2 Détection des conflits

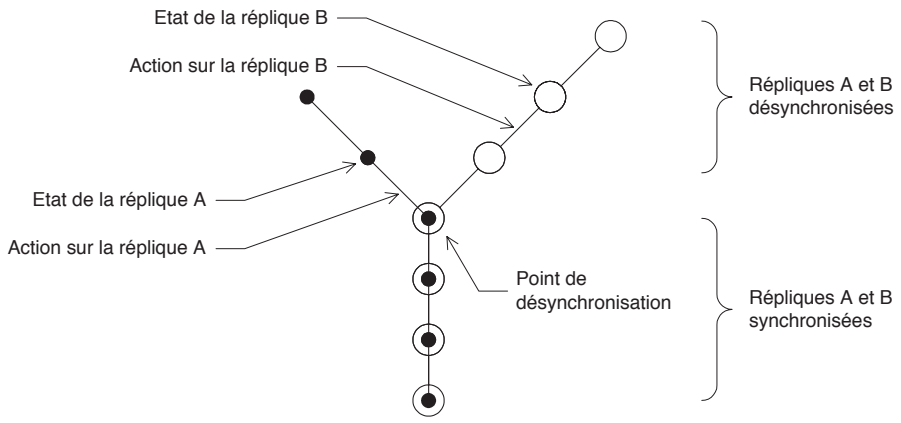


FIG. 6.22 – Historique de répliques désynchronisées

La détection des conflits consiste à analyser les conflits potentiels entre les différentes actions qui ont été produites sur les répliques synchrones $r_A = O/_{synch}U$ et une réplique asynchrone $r_B \in O/_{asynch}U$ à partir du *point de désynchronisation*, ce dernier caractérisant le moment où $r_B$est passé dans $O/_{asynch}U$ (figure 6.22). Nous définissons le conflit entre deux actions de la manière suivante :

> *Soit deux actions a_1 et a_2 consommables par un consommateur C et notons $\mathcal{P}(a, C)$ l'ensemble des propriétés de C susceptibles d'être modifiées lors de la consommation de a par C.*
>
> *L'action a_1 est en conflit avec l'action a_2 pour le consommateur C , noté $a_1 \parallel_C a_2$, si et seulement si $\mathcal{P}(a_1, C) \cap \mathcal{P}(a_2, C) \neq \emptyset$.*
>
> *Puisque $\forall(a_1, a_2)$, $\mathcal{P}(a_1, C) \cap \mathcal{P}(a_2, C) = \mathcal{P}(a_2, C) \cap \mathcal{P}(a_1, C)$, la relation $\parallel$ est commutative.*

L'algorithme de synchronisation d'une réplique r_B vers une réplique r_A consiste alors à déterminer la liste $\omega_{A\|B}$des actions faites sur r_A qui entrent en conflit avec celles faites sur r_B :

- Soit Ω_A (respectivement Ω_B) la liste des actions produites sur la réplique $r_A = O/_{synch}U$ (respectivement sur $r_B \in O/_{asynch}U$) depuis la désynchronisation de r_B.
- Pour chaque action $\alpha \in \Omega_A$, le test de conflit est réalisé avec chacune des action $\beta \in \Omega_B$. Soit $\omega_B(\alpha)$ la sous-liste de Ω_B des actions β entrant en conflit avec α, *i.e.* $\omega_B(\alpha) = \{\beta \in \Omega_B \,/\alpha \parallel \beta\}$. La liste $\omega_{A\|B}$ des actions de Ω_B entrant en conflit avec les actions de Ω_A est alors l'ensemble défini par $\omega_{A\|B} = \{\omega_B(\alpha) \,/\alpha \in \Omega_A\}$, soit finalement :

$$\omega_{A\|B} = \{\{\beta \in \Omega_B \,/\alpha \parallel \beta\} \,/\alpha \in \Omega_A\}$$

Une fois cet ensemble déterminé, l'algorithme peut synchroniser les répliques selon les politiques de synchronisation envisagées dans la section précédente, pour chaque élément de $\omega_{A/B}$.

L'algorithme réalisant le test de conflit $\alpha \parallel_C \beta$ utilise la méthode de faisabilité de β (méthode "canβ") comme suit :

- Soit C' un consommateur instance de la classe de C et sur lequel on a invoqué la méthode de notification de début de α.
- $\alpha \parallel_C \beta$ si et seulement si la méthode de faisabilité de β invoquée sur C'retourne vrai.

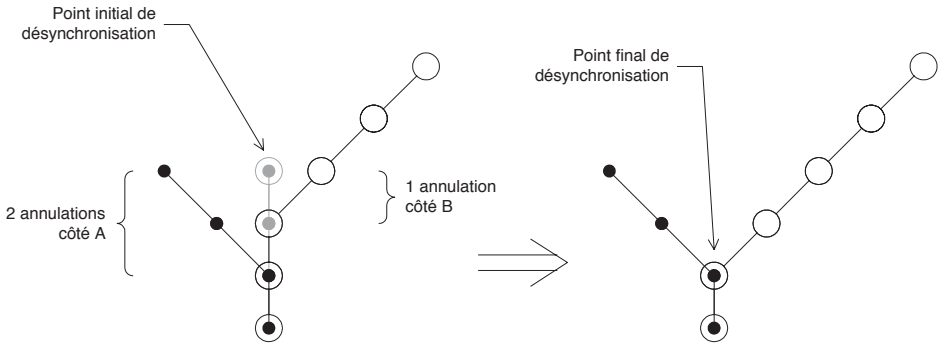


FIG. 6.23 – Répliques avec annulations désynchronisées

Notons que la hauteur du point de désynchronisation n'est pas immuablement défini au moment où r_B se désynchronise de r_A mais peut varier si des annulations ont eu lieu entre temps sur r_A ou/et r_B. Dans l'exemple de la figure 6.23, deux actions ont été annulées puis deux autres produites côtés r_A, alors qu'une action a été annulée et trois autres produites côté r_B (les points grisés représentes les états antérieurs annulés). L'algorithme de calcul de $\omega_{A\|B}$ doit alors ajuster la hauteur du point de désynchronisation de manière à ce que les deux ensembles Ω_A et Ω_B soient cohérents par rapport à la nouvelle position de la divergence entre r_A et r_B. Cet ajustement est trivial :

> La position du point de désynchronisation est décrémentée de $\delta = max(n_A, n_B)$ où n_A (respectivement n_B) le nombre d'annulations effectuées sur r_A (respectivement r_B) depuis la désynchronisation.

Pour la figure 6.23, $n_A = 2$ et $n_B = 1$ soit $\delta = 2$.

6.5 Conscience mutuelle

Dès qu'il y a collaboration à distance, la conscience mutuelle des actions faites par les utilisateurs devient un aspect essentiel et critique. Le système doit alors être capable de fournir différentes techniques, telles que celles décrites dans la section 2.2.3.3, permettant aux utilisateurs de posséder une telle conscience. Nous présentons dans les sections suivantes l'implantation de l'écho et de la localisation dans DPI.

6.5.1 Écho

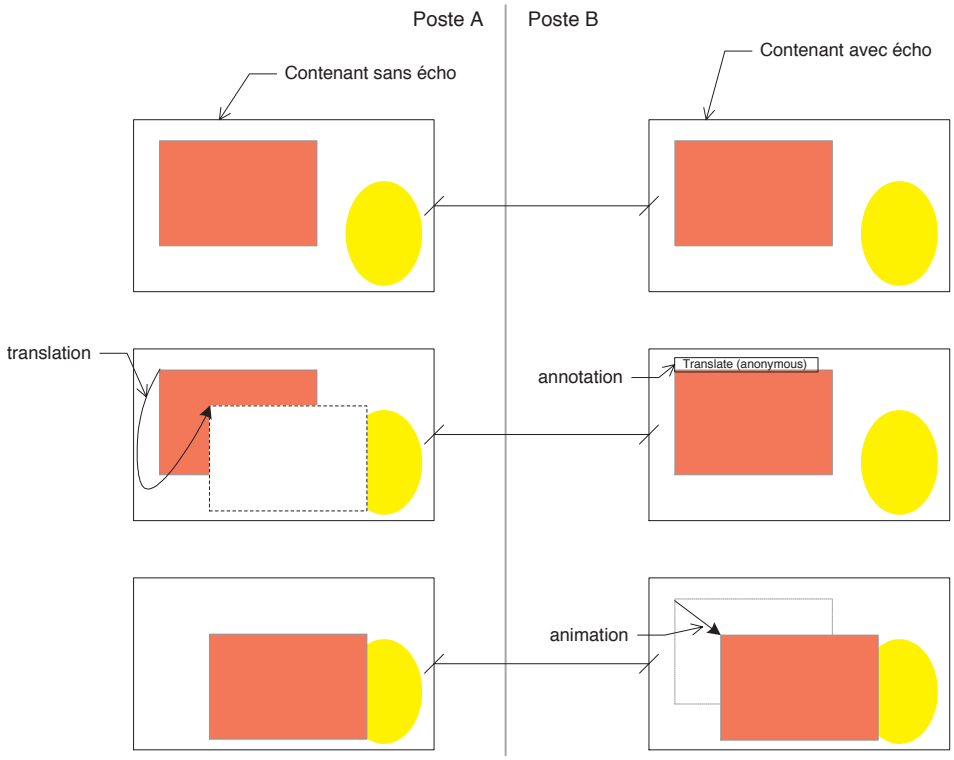


FIG. 6.24 – Exemple d'écho pour une translation

La figure 6.24 illustre un écho possible pour une action de translation réalisée sur un rectangle :

- Contexte :
  - La translation est effectuée sur le rectangle intérieur au contenant associé à un point de partage du poste A.
  - Sur le poste B, un contenant est associé à ce même point de partage.
- Cycle de la translation :
  - Du début à la fin de la translation, une annotation associée au rectangle à translater indique le type d'action en cours de production ainsi que son auteur.
  - A la fin de la translation, une translation linéaire depuis le point de départ jusqu'au point d'arrivée est effectuée sous la forme d'une *animation*.
  - Une fois l'animation terminée, la réplique du rectangle est synchronisée (la réalisation de l'écho *doit* naturellement garantir une telle synchronisation)

Tout comme la synchronisation, l'écho est un mode défini localement au point de partage. Dans la figure 6.24, le point de partage A n'est pas en mode écho tandis que le poste B l'est. Le principe de réalisation de l'écho est assez simple :

1. La translation démarre sur le rectangle répliqué R_A du poste A :
   - La méthode beginTranslate est invoquée sur R_A.
   - Le point de partage P_A du poste A effectue une demande d'invocation de beginTranslate.
   - Le point de partage P_B du poste B reçoit la demande d'invocation de beginTranslate :
     - Il annote le rectangle ciblé R_B comme indiqué ci-dessus.
     - Il invoque la méthode beginTranslate sur R_B.
2. La translation de R_A s'exécute :
   - La méthode doTranslate est invoquée sur R_A.
   - P_A effectue une demande d'invocation de doTranslate.
   - Le point de partage de droite P_B reçoit la demande d'invocation de doTranslate. La méthode doTranslate n'est alors *pas* invoquée sur R_B.
3. La translation de R_A se termine :
   - La méthode endTranslate est invoquée sur R_A . Celle-ci insère, puisque R_A est partagé, une entrée de type écho (classe EchoEntry) dans le point de partage P_A, laquelle contient la position finale de R_A.
   - P_A effectue une demande d'invocation de endTranslate en passant l'entrée précédente de l'écho en argument[14].
   - Le point de partage P_B reçoit la demande d'invocation de endTranslate :
     - La méthode endTranslate est invoquée sur R_B.
     - L'écho est alors joué par invocation d'une méthode, appelée méthode d'écho, sur R_B. Il consiste à réaliser l'animation décrite précédemment.

La *méthode d'écho* se substitue ainsi, en mode écho, à la méthode d'exécution. Cette méthode est naturellement définie dans l'interface de consommation de l'action. Ainsi, l'interface de consommation de l'action de translation s'écrit maintenant :

```
public interface TranslateConsumer extends Consumer {
  public boolean canTranslate() ;
  public void beginTranslate(TranslateProducer producer) ;
  public void endTranslate(TranslateProducer producer) ;
  public void doTranslate(double x, double y) ;
  public void undoTranslate(ArgumentStack arguments) ;
  public void redoTranslate(ArgumentStack arguments) ;
   public void echoTranslate(ArgumentStack arguments) ;
}
```

Si la méthode d'écho n'est pas définie dans l'interface, le mode d'écho conduit à l'appel de la méthode d'exécution. Dans ce cas, l'action est dite "sans écho".

L'annotation, réalisée au début de l'action par le point de partage en mode écho, est un élément graphique inséré dans la réplique même du rectangle partagé. Ceci conduit alors à la réplication de l'annotation elle-même. Cet effet étant indésirable, nous définissons le concept de *point d'arrêt* de l'induction du partage comme suit :

> *Un* point d'arrêt *de l'induction du partage est un point qui, lorsqu'il est associé à un composant, stoppe l'induction du partage sur ce composant et tous ses descendants*.

[14] L'entrée de type écho pour une action est construite par le poste où a lieu l'action. Ce poste n'a, en effet, pas à savoir si l'action va être reproduite en mode écho sur les autres postes.

Nous fournissons ainsi une solution *ad hoc* au problème de l'annotation dans un point de partage. Notons qu'il est important que l'utilisation des points de partage, des points d'arrêt ainsi que la hiérarchisation possible des points de partage (possibilité pour un composant d'être partagé par le biais de plusieurs points de partage) soit étudiée de manière plus précise après la thèse.

6.5.2 Localisation

La localisation, qui permet de percevoir quelle partie de l'objet partagé est visualisée par un tiers utilisateur (section 2.2.3.3), est triviale à implanter au travers des points de partage. Il suffit d'ajouter un graphique représentant le contour du contenant, appelé *contour de localisation*, associé au point de partage à l'intérieur même de ce composant. Ce contour est alors inductivement répliqué sur les autres points de partage qui peuvent alors jouer sur son apparence locale : si le mode est sans localisation, le contour est rendu invisible (propriété visible), et si le mode est avec localisation, le contour peut être affiché avec différents attributs tel que le nom de l'utilisateur ou sa couleur. Le rendu du contour de localisation joue sur le fait que les changements des propriétés des objets partagés ne sont pas répliqués par les points de partage (section 6.6.2.2).

6.5.3 Communication médiatisée

La communication médiatisée est un élément essentiel de la collaboration à distance. Elle favorise entre autre la conscience mutuelle. Cependant, nous ne développons pas dans cette thèse les aspects liés aux mediaspaces. Nous ne définissons dans OpenDPI que les classes essentielles qui permettent, d'une part, la gestion des périphériques audio et vidéo et, d'autre part, le transport réseau des flux associés (Sun, 1999).

6.6 Points de partage locaux

Nous avons défini dans OpenDPI, initialement dans un but purement pratique, la notion de *point de partage local*. Un point de partage local fonctionne exactement selon les principes du point de partage évoquées plus haut, mais ne nécessite pas l'utilisation d'un réseau : ils permettent alors de définir des contenants associés aux points de partage ayant des contenus "identiques". L'adresse IP de groupe est dans ce cas un simple nom, réputé unique, représentant le "groupe local". L'intérêt initial du point de partage local est de permettre la mise au point du partage de composants, tels que les documents, les fenêtres, ou les espaces de travail, en s'affranchissant, dans un premier temps, des problèmes d'incohérence et de concurrence d'accès liés à la collaboration distante.

La boîte à outils OpenDPI définit en conséquence la classe de base abstraite SharingPoint, et les deux classes dérivées LocalSharingPoint et RemoteSharingPoint. La classe SharingPoint implante l'ensemble des mécanismes de dialogues entre les points de partage qui consiste en l'envoi et en la réception de messages. Les classes dérivées LocalSharingPoint et RemoteSharingPoint définissent quant à elles la manière dont ces messages sont envoyés d'un point de partage à un autre.

Les points de partage locaux nous semble pertinents pour compléter le modèle DPI en dehors des aspects de la collaboration à distance, en particulier en ce qui concerne le modèle de rendu des graphiques conservant la cohérence de l'interaction.

6.6.1 Application à la loupe consistante

Il est important de remarquer que les changements des valeurs des propriétés des composants partagés ne sont *pas* répliqués par les points de partage. Nous avons en effet considéré que les propriétés étaient exclusivement modifiées *via* les actions. Ceci nous à alors donner l'idée de définir des contenus synchronisés aux valeurs des propriétés près. Les répliques sont considérées dans ce cas comme *pseudo-synchrones*.

Nous avons souligné, dans la section 5.2, le problème de incohérence de l'interaction qui peut survenir lorsque l'interaction a lieu à travers la loupe ou la lentille magique (figure 5.2 on page 135). Le modèle associé, proposé en annexe (section A.7) ne garantit effectivement pas la cohérence de l'interaction pour les loupes. En utilisant les points de partage locaux, nous proposons un modèle qui permet de garantir une telle cohérence.

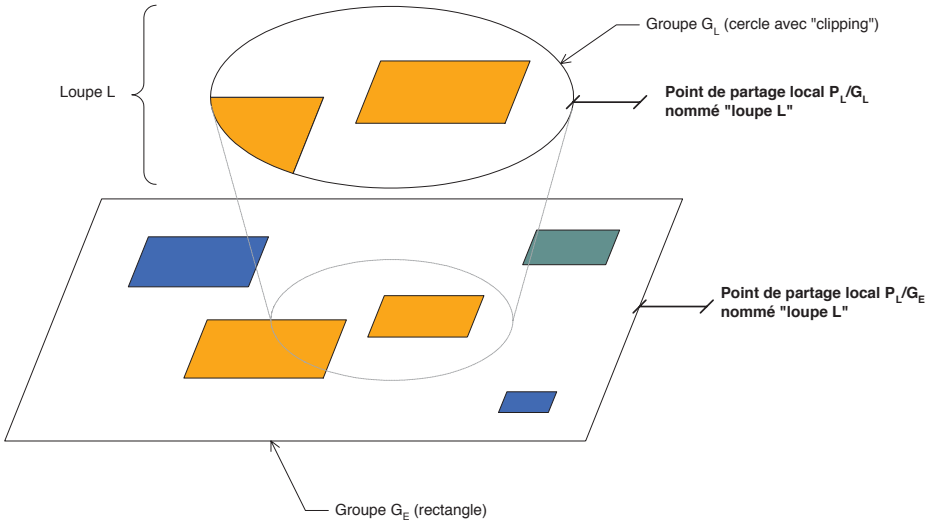


FIG. 6.25 – Éléments d'une loupe consistante

Nous construisons une loupe grossissante L effectuant le rendu d'un groupe de graphique G_E créé par ailleurs dans l'espace de travail (figure 6.25) comme suit :

1. La loupe L créé un point de partage P_L/G_E nommé "loupe L" et ayant pour contenant G_E.
2. Elle créé un second point de partage P_L/G_L, également nommé "loupe L" et ayant pour contenant un nouveau groupe de graphiques G_L. Le composant G_L sera celui affiché par la loupe et joue donc le rôle de *composant de rendu*[15]. Les deux contenants G_E et G_L ont alors le même contenu : ils sont synchronisés.
3. La loupe L définit alors son facteur de zoom (égal à 1,5 sur la figure) en appliquant un facteur d'échelle à G_L *via* la méthode setScale. Le composant G_L est donc pseudo-synchrone puisque la valeur de sa propriété scale est *a priori* différente de celle de G_E.

Pour une telle loupe, l'interaction reste *consistante* lorsque, par exemple, un composant qu'elle affiche est déplacé *par dessus* la loupe. En effet, quand l'utilisateur effectue la translation (dx, dy), elle est produite sur le graphique piqué qui est contenu dans le composant de rendu de la loupe (G_L dans notre exemple), et non sur le graphique rendu par la

[15] Notons que, lorsque la loupe se déplace, elle applique un déplacement adéquat au composant de rendu.

loupe (G_E dans notre exemple). Le point de partage prend alors en charge la réplication de l'action de manière à rendre les répliques (pseudo)-synchrones.

Notons que le concept de point de partage local suggère celle des présentations multiples d'un document. Cependant, la synchronisation des présentations multiples passe nécessairement par le document qui doit, à cet effet, intégrer un nombre de données du domaine suffisant. En utilisant le points de partage locaux, il est possible de (pseudo)synchroniser des présentations sans avoir recours aux documents.

6.6.2 Application au rendu altéré

Nous allons là encore mettre à profit le pseudo-synchronisme offert par les points de partage pour définir un modèle de rendu qui conserve la cohérence de l'interaction, ce qui n'était pas le cas avec celui proposé dans la section A.7. Nous pourrons alors construire des lentilles magiques consistantes.

6.6.2.1 Techniques existantes

Edwards et al. (1997a) proposent trois approches combinables pour la transformation du rendu et qu'ils ont implantées avec la boîte à outils SubArtic[16]. La première approche consiste à dériver les classes des widgets (mécanisme d'héritage de la programmation à objet). Les classes dérivées redéfinissent alors la manière dont le rendu et effectué. La seconde approche est basée sur la technique de "wrapping" : un composant *A* dont on souhaite modifier le rendu est associé à un autre composant *B* qui prendra en charge le rendu altéré de *A*. Ce mécanisme est assez semblable à celui du "portable look & feel" de l'API Swing (Eckstein et al., 1998) pour lequel chaque type composant graphique est associé un objet qui effectue le rendu graphique ainsi que la gestion des interactions du composant. Le première méthode basée sur la dérivation possède les limites de l'héritage : il ne peut être spécifié qu'au moment de la construction de l'application (*i.e.* statiquement). La seconde approche ne possède pas un tel inconvénient : le "wrapping" peut s'effectuer lors de l'exécution de l'application (*i.e.* dynamiquement). Cependant, elle impose que le composant *B* connaisse les caractéristiques du composant *A* pour lequel il effectue le rendu. Enfin, une troisième technique est envisagée lorsque le rendu ne peut altéré qu'après le rendu initial (alors effectué d'une manière non visible), comme dans le cas d'un flou par exemple. Le rendu s'effectue sans l'approche de SubArtic au niveau du pixel. Dans notre approche basée sur une représentation vectorielle, ce type d'altération du rendu n'est pas facile à réaliser et nécessite de travailler directement au niveau du moteur de rendu vectoriel (qui effectue la transformation graphiques vectoriels $\rightarrow$ pixels). Aucune de ses trois techniques ne nous semble satisfaisante pour résoudre notre problème d'altération du rendu (vectoriel).

L'implantation du rendu spécifique pour les lentilles magiques de Bier et al. (1993) utilise trois autres approches différentes, plus adaptées à une représentation vectorielle des composants graphiques. La première méthode, appelée "recursive ambush", est adaptée aux modèles décrits de manière procédurale par un langage tel que PostScript : la lentille utilise un interpréteur de ce langage différent de celui utilisé pour le rendu du graphe de scène. L'avantage de cette méthode est de rester indépendante des applications. Par contre, elle est relativement lourde à mettre en œuvre et est assez mal adaptée à la superposition de lentilles magiques. De plus, la cohérence de l'interaction ne peut pas être garantie avec cette méthode puisque seul le rendu est altéré. Le seconde méthode, appelée "model-in model-out", impose à la lentille de travailler sur une *copie* des objets affichés. Le rendu consiste alors à modifier ces objets, la modification pouvant être légère (substitution d'une couleur par exemple) ou élevée (changement de la structure de l'objet). Bien que les auteurs ne le

[16] http ://www.cc.gatech.edu/gvu/ui/sub_arctic

mentionnent pas, ce principe garantit la cohérence de l'interaction. La solution que nous fournissons est basée sur cette idée. La troisième méthode proposée, appelée "reparametrize and clip" consiste à paramétrer le gestionnaire de rendu qui est associé à la lentille. Elle reste simple mais est limité par les paramètres pré-définis par le gestionnaire de rendu.

6.6.2.2 Modèle de rendu altéré

L'incohérence de l'interaction, induite par le modèle de rendu proposé dans la section A.7, provenait du fait que l'utilisateur manipulait les objets au travers de leur rendu altéré : l'objet manipulé n'était alors plus conforme à ce qui était présenté par la loupe. Nous allons en conséquence bâtir un nouveau modèle de rendu basé sur l'altération des propriétés des composants.

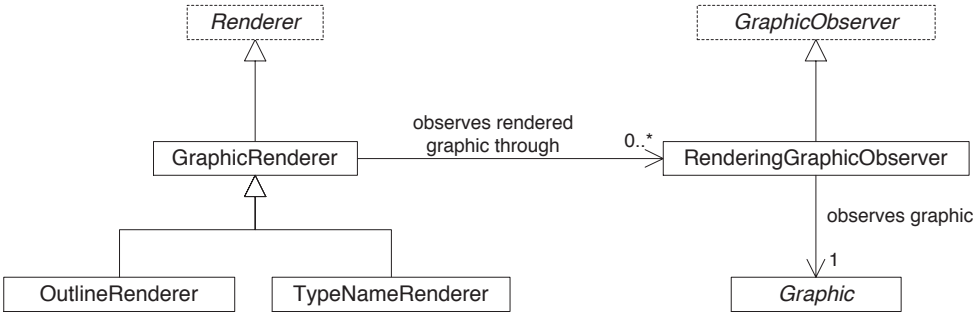


FIG. 6.26 – Modèle de rendu

La figure 6.6.2.2 présente les classes intervenant dans le mécanisme de rendu altéré des graphiques DPI. Le rendu altéré est réalisé par des gestionnaires de rendu (interface Renderer) en modifiant les propriétés des graphiques qu'il doit rendre : quand une propriété pppp d'un graphique est modifiée, le gestionnaire de rendu invoque la méthode renderPppp et utilise à cet effet une classe d'observation du graphique (classe interne RenderingGraphicObserver).

Reprenons l'exemple du gestionnaire de rendu OutlineRender qui effectue l'affichage du contour d'un graphique avec la couleur de fond du graphique (figure A.14 on page 223) :

```
public class OutlineRenderer extends GraphicRenderer {

  public void  renderGraphic(Graphic graphic) {
   if (graphic instanceof Shape) {
     Shape shape = (Shape) graphic ;
     shape.setStrokeWidth(1) ;
     renderColor(graphic) ;
   }
  }

  public void  renderColor(Graphic graphic) {
   if (graphic instanceof Shape) {
     Shape shape = (Shape) graphic ;
     shape.setStrokeColor(shape.getColor()) ;
     shape.setColor(null) ;
   }
  }
}
```

La méthode renderGraphic est appelée quand le graphique est rendu pour la première fois par le gestionnaire (méthode startRendering). Elle réalise alors l'initialisation du rendu spécifique : si le graphique est une forme vectorielle, il est *transformé* en une forme ayant un contour de un pixel, puis il est peint *via* un appel de renderColor. La méthode renderColor (déclarée dans la classe GraphicRenderer) est appelée par renderGraphic et à chaque fois que la valeur de la propriété color du graphique passée en paramètre est modifiée. Elle spécifie, dans notre exemple, la couleur du contour (propriété strokeColor) comme étant la valeur de la couleur du graphique (propriété color), cette dernière étant alors forcée à zéro (*i.e.* le graphique n'est formé que du contour et n'a pas de "fond").

Par ce procédé simple, il est possible de définir des rendus différents pour chaque composant (typiquement des graphiques). Nous complétons notre modèle en définissant la notion de *point de rendu* comme suit :

> *Un* point de rendu *est un point qui associe un composant à une liste de gestionnaires de rendu. Lorsque le composant doit être rendu, chaque gestionnaire de la liste est invoqué de manière à éventuellement altérer les valeurs des propriétés du composant.*
> *L'*induction du rendu *est le mécanisme par lequel les composants descendants d'un composant associé à un point de rendu sont rendus par les gestionnaires de rendu associé à ce point.*

Dans l'exemple de la figure 6.7 on page 159, le code suivant permet de spécifier le rendu "par contour" du point de partage du poste B, noté sharingPointB :

```
RoundedClipRectangle areaB = new RoundedClipRectangle(100,100) ;
sharingPointB = new RemoteSharingPoint("230.0.0.1", 6789, areaB) ;
OutlineRenderer outlineRenderer = new OutlineRenderer() ;
sharingPointB.setRenderingPoint(new Renderer[] { outlineRenderer }) ;
```

L'interaction conserve là encore sa cohérence puisque les graphiques rendus ont des propriétés conformes à ce qui est affiché.

6.6.2.3 Lentille magique consistante

Il est maintenant aisé de modifier, avec le mécanisme de rendu décrit précédemment, la loupe grossissante en une lentille magique. La lentille magique construit, comme la loupe grossissante un point de partage P_L/G_E associé au graphique à visualiser G_E et un point de partage P_L/G_L associé à son contenant de rendu G_L. Elle se différencie de la loupe magique par le fait qu'elle permet de définir un ensemble de gestionnaires de rendu associé, *via* un point de rendu, à G_L. Par exemple, une lentille magique qui n'affiche que le contour des graphiques de la couche "main layer" de l'écran se construit comme ceci :

```
OutlineRenderer outlineRenderer = new OutlineRenderer() ;
MagicLens lens = new MagicLens() ;
lens.addLayer(
  Screen.self().getMainLayer(),
  new Renderer[] { outlineRenderer }) ;
```

La ligne 3 réalise la définition d'un point de rendu pour la couche "main layer", ce point de rendu utilisant un unique gestionnaire de rendu instance de OutlineRenderer.

Notons que les lentilles et loupes grossissantes peuvent avoir une forme quelconque (typiquement circulaire ou carrée). Leur réalisation reste relativement indépendante de Piccolo.

6.7 Conclusion

Nous avons introduit dans le modèle DPI la notion de point de partage de manière à rendre possible la réplication des composants DPI, point clé pour la collaboration distante.

Un point de partage définit la manière dont un nœud du graphe de scène peut être répliqué sur des postes distants, en gérant en particulier les problèmes de concurrence et de cohérence. Il permet de préciser le degré de couplage spatial en fonction de la hauteur du point de partage dans le graphe de scène. Il gère le couplage temporel en permettant un travail en mode asynchrone ou synchrone, avec ou sans écho.

Nous avons implanté dans OpenDPI une version simplifiée des points de partage basée sur les groupes “multicast”. Cette version ne gère pas la concurrence ni la cohérence des répliques. Le concept de place n’a pas été implantée dans OpenDPI et nécessiterait une étude plus approfondie de la multiplicité des points de partage dans un même graphe de scène (en particulier, dans une même branche du graphe).

Enfin, le concept de point de partage a été utilisée pour résoudre le problème de l’altération du rendu des composants graphiques. Les points de partage locaux ont été entièrement implantés dans OpenDPI et permettent de définir, par exemple, des lentilles magiques consistantes.

Conclusion

Synthèse

Cette thèse a été motivée par le triple constat que les systèmes actuels restent centrés sur les applications, qu'ils réduisent la richesse potentielle de l'interaction en imposant le tandem clavier + souris, et qu'ils ne permettent pas une mise en œuvre simple des mécanismes de collaboration. Nous avons imaginé, pour contrer ce constat, un modèle pour la conception de systèmes interactifs et collaboratifs centrés sur les documents et les instruments.

Le modèle proposé s'appuie sur différents fondements jugés pertinents. Il a adopté certains principes fondamentaux, tels que la manipulation directe de Shneiderman (1983), les interfaces orientée objet (Collins, 1995), et le principe de généricité et de polymorphisme des actions de Beaudouin-Lafon & Mackay (2000). Le modèle de l'interaction est centrée sur les notions d'instrument de Beaudouin-Lafon (2000) et d'interacteur de Myers (1990). L'objet d'intérêt essentiel n'est plus l'application mais le document ; le standard XML (W3C, 2000) a alors inspiré notre modèle de document. Les trois éléments essentiels que sont les documents, leurs présentations et les instruments sont considérés comme des "composants logiciels" (Szyperski, 2002) : ils s'appuient sur un modèle de composant générique qui permet leur interopérabilité.

Les aspects théoriques qui régissent le modèle DPI partent de la boucle action-perception (Norman & Draper, 1986) sur laquelle les trois composants D, P et I ont été mis en jeu. L'interaction est spécifiée en décrivant comment les instruments communiquent avec les documents et dans quelle mesure ils sont compatibles. Le couplage document / présentation est précisé par une relation d'observabilité entre le document et ses présentations. Ces différents aspects du modèle ont été mis en œuvre dans la boîte à outils OpenDPI qui vise, à terme, à mesurer la pertinence de notre modèle.

Un affinement du modèle a été proposé afin d'intégrer l'aspect collaboratif des espaces de travail. La notion de cohérence de l'interaction a été introduite pour mettre en évidence les conséquences possibles de l'application simultanée d'actions sur un même composant. Nous avons proposé, dans le cadre de la collaboration locale et de l'interaction bimanuelle, la technique de verrouillage des propriétés des composants afin d'autoriser ou d'interdire, selon les situations, les actions simultanées.

La collaboration à distance a été imaginée au niveau du concepteur autour de la métaphore de la *place*. Au niveau du programmeur, elle a été abordée en proposant une architecture répliquée basée sur la notion de point de partage. Le concept de point de partage, partiellement implanté dans OpenDPI, permet de prendre en compte d'une manière unifiée les différents points soulevés par le partage de composants, tels que la réplication de l'état et des actions faites sur les composants, le réglage des degrés de couplage spatial et temporel, la cohérence des répliques et la conscience mutuelle.

Perspectives

Les perspectives de nos travaux concernent deux aspects : l'évaluation du modèle DPI d'une part et la proposition d'un modèle basé sur des standards d'autre part.

Lors de l'élaboration du modèle, nous avons choisi certains fondements jugés pertinents. Cependant, si de tels choix ont été justifiés dans cette thèse, il n'en demeure pas moins que, d'une part, le modèle résultant n'a pas nécessairement bien "capturé" ces fondements et que, d'autre part, l'utilisation du modèle pour la conception d'applications et d'espaces de travail ne s'avère pas aussi pertinente "à l'usage". Ainsi, la prochaine étape consisterait à valider le modèle DPI en construisant une suite logicielle définissant un espace de travail interactif et collaboratif dédié à des tâches spécifiques. Disposant d'un tel outil, il deviendrait possible d'évaluer la pertinence du modèle selon les points de vue utilisateur, concepteur et programmeur. En particulier, cet outil devrait permettre de mesurer le degré de pertinence d'une approche "sans" application ainsi que les problèmes (nouveaux) qui en découleraient.

En parallèle à ce premier aspect, il serait intéressant de faire le bilan des insuffisances du modèle ainsi que de proposer un modèle affiné qui serait basé sur des standards reconnus. Nous avons en particulier identifié plusieurs pistes d'affinement et de généralisation de DPI :

Modèle du document Bien que notre modèle de document s'inspire de XML, il reste indépendant du modèle DOM (W3C, 2002). En particulier, les mécanismes d'observation et d'envoi d'événements de DOM complètent le formalisme XML et pourraient avantageusement être intégrés dans notre modèle.
De plus, la liaison entre un document (données du domaine) et ses présentations est basée sur le mécanisme d'observation assez similaire à celui de MVC. Ce mécanisme est cependant relativement lourd du point de vue du programmeur et autorise difficilement des transformations complexes entre le document et ses présentations. Des techniques de transformation basées sur des langages tel que XSL (Drix, 2002) pourraient être envisagées, à condition de les adapter aux systèmes interactifs.

Modèle du rendu graphique L'utilisation de la boîte à outils Piccolo dans OpenDPI nous a permis d'utiliser un graphe de scène simple adapté à nos besoins. Cependant, le standard SVG nous semble plus précis quant à la spécification d'un graphe de scène 2D. De plus, son utilisation permet d'envisager un unique modèle de document basé sur DOM pour les documents comme pour les présentations.

Modèle des actions La notion de document suggère souvent davantage l'idée de données que celle d'action. En particulier, la communauté XML s'intéresse à la forme des données mais pas significativement à leur comportements (*i.e.* aux actions qu'elles peuvent consommer). Il serait intéressant, par exemple, d'imaginer comment le modèle DOM pourrait être agrémenté d'une spécification modélisant le processus de consommation des actions de notre modèle.
Dans le modèle DPI, les actions sont définies au même niveau que les données, *i.e.* à l'intérieur d'une classe. Ceci peut alors poser des problèmes dans le cas des actions génériques. Il serait pertinent de séparer les classes décrivant la forme des données de celles décrivant le traitement des actions sur ces données.

Modèle pour la collaboration La gestion de la concurrence d'accès en collaboration locale et interaction bimanuelle, bien que permettant d'assurer la cohérence de l'interaction, s'avère assez lourde du point de vue de programmeur dès qu'il y a induction. Il faudrait explorer d'autres possibilités moins contraignantes, telle que la substitution d'une action non faisable par une autre action faisable, et qui permettraient de garantir la cohérence de l'interaction.
Enfin, nous n'avons pas montré que les points de partage permettent d'élaborer des

schémas de collaboration variés. Il serait donc nécessaire d'implanter le concept de place en utilisant ces points de partage. De plus, la notion de point de partage ne s'appuie pas exclusivement sur l'ensemble des principes de DPI et pourrait être envisagée d'une manière plus générale. Par exemple, si nous avons opté pour une architecture répliquée, les points de partage devraient pouvoir être envisagés dans des architectures centralisées ou hybrides.

La réalisation de l'ensemble de ces points nécessite encore beaucoup d'effort et de réflexion. Nous sommes cependant convaincus que l'approche sans application permet de mettre en œuvre des environnements plus simples et plus puissants, au prix d'une rupture assez radicales avec les environnements actuels.

Table des figures

Liste des tableaux

Bibliographie

Alashqur, A. M., Su, S. Y. W. & Lam, H. (1989), OQL : a query language for manipulating object-oriented databases, *in Proceedings of the fifteenth international conference on Very large data bases*, Morgan Kaufmann Publishers Inc., pp.433–442.

Apple (1994), OpenDoc Technical Summary, Technical documentation, Apple Computer Inc.

Backlund, B. E. (1997), "OOE : A Compound Document Framework", *ACM SIGCHI Bulletin* **29**(1), 68–75.

Baecker, R. M., Grudin, J., Buxton, W. A. S. & Greenberg, S. (1995), *Readings in Human-Computer Interaction : Towards the Year 2000*, 2nd edition, Morgan Kaufmann.

Beaudouin-Lafon, M. (1997), Interaction Instrumentale : de la manipulation directe à la réalité augmentée, *in Actes de la Conférence Francophone sur l'Interaction Homme-Machine (IHM'97)*, Cépaduès Éditions.

Beaudouin-Lafon, M. (2000), Instrumental interaction : An interaction model for designing post-wimp interfaces, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'00)*, *CHI Letters* **2**(1), ACM Press, pp.446–453.

Beaudouin-Lafon, M. (2001), Novel interaction techniques for overlapping windows, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'01)*, *CHI Letters* **3**(2), ACM Press, pp.153–154.

Beaudouin-Lafon, M. & Karsenty, A. (1992), Transparency and Awareness in a Real-Time Groupware System, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'92)*, ACM Press, pp.171–180.

Beaudouin-Lafon, M. & Lassen, H. M. (2000), The Architecture and Implementation of CPN2000, A Post-WIMP Graphical Application, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'00)*, *CHI Letters* **2**(2), ACM Press, pp.181–190.

Beaudouin-Lafon, M. & Mackay, W. (2000), Reification, polymorphism and reuse : Three principles for designing visual interfaces., *in Proceedings of the Conference on Advanced Visual Interfaces (AVI'00)*, ACM Press, pp.102–109.

Beaudouin-Lafon, M. & Mackay, W. (2002), Prototyping Development and Tools, *in* J. Jacko & A. Sears (eds.), *Human Computer Interaction Handbook*, Lawrence Erlbaum Associates, pp.1006–1047.

Beaudouin-Lafon, M., Berteaud, Y. & Chatty, S. (1990), Creating direct manipulation applications with XTV, *in Proceedings of the European X Window System Conference (EX'90)*.

Beaudoux, O. (2002), Un Pivot pour la Collaboration : les Places Publiques, *in Annexes des actes de la Conférence Francophone sur l'Interaction Homme-Machine (IHM'2002)*.

Beaudoux, O. & Beaudouin-Lafon, M. (2001), DPI : A Conceptual Model Based on Documents and Interaction Instruments, *in People and Computer XV - Interaction without*

Frontier (Joint proceedings of HCI 2001 and IHM 2001), Springer Verlag, pp.247–263.

Bederson, B. B. & Hollan, J. D. (1994), Pad++ : a zooming graphical interface for exploring alternate interface physics, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'94)*, ACM Press, pp.17–26.

Bederson, B. B., Meyer, J. & Good, L. (2000), Jazz : An Extensible Zoomable User Interface Graphics Toolkit in Java, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'00)*, *CHI Letters* **2**(2), ACM Press, pp.171–180.

Berlage, T. (1994), "A selective undo mechanism for graphical user interfaces based on command objects", *ACM Transaction on Computer-Human Interaction* **1**(3), 269–294.

Bier, E. & Freeman, S. (1991), MMM : A User Interface Architecture for Shared Editors on a Single Screen, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'91)*, ACM Press, pp.79–86.

Bier, E. A., Stone, M. C., Fishkin, K., Buxton, W. & Baudel, T. (1994), A taxonomy of see-through tools, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'94)*, ACM Press, pp.358–364.

Bier, E. A., Stone, M. C., Pier, K., Buxton, W. & DeRose, T. D. (1993), Toolglass and magic lenses : the see-through interface, *in Proceedings of the Conference on Computer Graphics and Interactive Techniques*, ACM Press, pp.73–80.

BNF (2003), "L'aventure des écritures - Matières et formes", Dossiers pédagogiques de la Bibliothèque Nationale de France (http ://classes.bnf.fr/dossisup/).

Brockschmidt, K. (1995), *Inside OLE, Second Edition*, Microsoft Press.

Buchner, J. (2000), "HotDoc : a framework for compound documents", *ACM Computing Surveys* **32**(1), 33–38.

Cadoz, C. (1994), "Le geste canal de communication homme-machine - la communication instrumentale", *Technique et Science Informatique* **13**(1), 31–61.

Callahan, J., Hopkins, D., Weiser, M. & Shneiderman, B. (1998), An empirical comparison of pie vs. linear menus, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'98)*, ACM Press, pp.95–100.

Carlier, P., Saugis, G., Dumas, C., Chaillou, C., Derycke, A. & J.-C.Tarby (1997), SPACE, une nouvelle génération d'interface Homme-Machine pour le TCAO, *in Actes de la Conférence Francophone sur l'Interaction Homme-Machine (IHM'97)*.

Celentano, A., Pozzi, S. & Toppeta, D. (1992), A multiple presentation document management system, *in Proceedings of the International Conference on Systems documentation*, ACM Press, pp.63–71.

Collectif (2001), Device Class Definition for Human Interface Devices (HID), Firmware Specification, Multiple Organizations.

Collins, D. (1995), *Designing Object-Oriented User Interfaces*, Benjamin/Cumming.

Croft, W. B. & Lefkowitz, L. S. (1988), Using a planner to support office work, *in Proceedings of the Conference on Office Information Systems (OIS'98)*, ACM Press, pp.55–62.

Demers, A., Greene, D., Hauser, C., Irish, W. & Larson, J. (1987), Epidemic algorithms for replicated database maintenance, *in Proceedings of the ACM Symposium on Principles of Distributed Computing*, ACM Press, pp.1–12.

Dewan, P. (1999), Architectures for Collaborative Applications, *in* M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, Vol. 7 of *Trends in Software Series*, John Wiley & Sons, pp.167–193.

Dewan, P. & Choudhary, R. (1995), "Coupling the user interfaces of a multiuser program", *ACM Transaction on Computer-Human Interaction* **2**(1), 1–39.

Dix, A., Finaly, J., Abowd, G. & Beale, R. (1998), *Human-Computer Interaction*, second edition edition, Prentice Hall.

Dourish, P. & Bly, S. (1992), Portholes : Supporting Awareness in a Distributed Work Group, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'92)*, ACM Press, pp.541–547.

Dragicevic, P. & Fekete, J. (2001), Input Device Selection and Interaction Configuration with ICON, *in People and Computer XV - Interaction without Frontier (Joint proceedings of HCI 2001 and IHM 2001)*, Springer Verlag, pp.543–448.

Drix, P. (2002), *XSLT fondamental*, Eyrolles.

Druin, A., Stewart, J., Proft, D., Bederson, B. & Hollan, J. (1997), Kidpad, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'97)*, ACM Press, New York, pp.463–470.

Dumas, C., Saugis, G., Chaillou, C. & Viaud, M.-L. (1998), A 3-D Interface for Cooperative Work, *in Proceedings of the Conference on Collaborative Virtual Environments (CVE'98)*.

Eckel, B. (2002), *Thinking in Java*, 3rd edition edition, Prentice-Hall.

Eckstein, R., Loy, M. & Wood, D. (1998), *Java Swing*, O'Reilly.

Edwards, W. K., Hudson, S. E., Marinacci, J., Rodenstein, R., Rodriguez, T. & Smith, I. (1997a), Systematic output modification in a 2D user interface toolkit, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'97)*, ACM Press, pp.151–158.

Edwards, W. K., Mynatt, E. D., Petersen, K., Spreitzer, M. J., Terry, D. B. & Theimer, M. M. (1997b), Designing and implementing asynchronous collaborative applications with Bayou, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'97)*, ACM Press, pp.119–128.

Ellis, C. A. (1999), Workflow Technology, *in* M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, Vol. 7 of *Trends in Software Series*, John Wiley & Sons, pp.29–54.

Ellis, C. A., Gibbs, S. J. & Rein, G. (1991), "Groupware : some issues and experiences", *Communication of the ACM* **34**(1), 39–58.

Engelbart, D. & English, W. (1968), "A Research Center for Augmenting Human Intellect", Reprinted in ACM SIGGRAPH (1994).

English, W., Engelbart, D. & Berman, M. (1967), "Display selection techniques for text manipulation", *IEEE Transactions on Human Factors in Electronics* **8**(1), 5–15.

Fidge, J. (1988), Timestamps in message-passing systems that preserve the partial ordering, *in Proceedings of the Australian Conference on Computer Science*, pp.56–66.

Fish, R. S., Kraut, R. E. & Leland, M. D. P. (1988), Quilt : a collaborative tool for cooperative writing, *in Proceedings of the Conference on Office Information Systems (OIS'88)*, ACM Press, pp.30–37.

Flores, F., Graves, M., Hartfield, B. & Winograd, T. (1988), "Computer systems and the design of organizational interaction", *ACM Transaction on Information Systems* **6**(2), 153–172.

Furnas, G. W. (1986), Generalized fisheye views, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'86)*, ACM Press, pp.16–23.

Furnas, G. W. & Bederson, B. B. (1995), Space-scale diagrams : understanding multiscale interfaces, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'95)*, ACM Press, pp.234–241.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994), *Design Patterns : Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Gaver, W., Moran, T., MacLean, A., Lövstrand, L., Dowrish, P., Carter, K. & Buxton, W. (1992), Realizing a Video Environment : EuroPARC's RAVE System, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'92)*, ACM Press, pp.27–35.

Geib, J.-M., Gransart, C. & Merle, P. (1999), *Corba : des concepts à la pratique*, 2nd edition, Dunod.

Genau, A. & Kramer, A. (1995), Translucent History, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'95)*, ACM Press, pp.250–251.

Gibson, J. J. (1977), *The Theory of Affordances*, Perceiving, Acting and Knowing, Lawrence Erlbaum Associates, R. Shaw & J. Bransford (eds.).

Gibson, J. J. (1979), *The Ecological Approach to Visual Perception*, Houghton Mifflin.

Gray, D. (1997), *The PhotoShop Plug-INS Book : Category Listings, Instructions and Examples*, Ventana Communications Group Inc.

Greenberg, S. & Marwood, D. (1994), Real Time Groupware as a Distributed System : Concurrency Control and its Effect on the Interface, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94)*, ACM Press, pp.207–217.

Greenberg, S. & Roseman, M. (2002), Using a Room Metaphor to Ease Transitions in Groupware, *in* V. W. M. Ackerman, V. Pipek (ed.), *Beyond Knowledge Management : Sharing Expertise*, MIT Press.

Greenberg, S., Roseman, M., Webster, D. & Bohnet, R. (1992), Issues and experiences designing and implementing two group drawing tools, *in Proceedings of the Hawaii International Conference on System Sciences (HICSS'92)*, IEEE Computer Society, pp.139–150.

Guiard, Y. (1987), "Asymmetric Division of Labor in Human Skilled Bimanual Action : The Kinematic Chain as a Model", *Journal of Motor Behavior* **4**, 486–517.

Guiard, Y., Bourgeois, F., Mottet, D. & Beaudouin-Lafon, M. (2001), Beyond the 10-bits Barrier : Fitts' Law in Multi-Scale Electronic Worlds, *in People and Computer XV - Interaction without Frontier (Joint proceedings of HCI 2001 and IHM 2001)*, Springer Verlag, pp.573–587.

Gutwin, C., Greenberg, S. & Roseman, M. (1996), Workspace Awareness Support With Radar Views, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'96)*, ACM Press, pp.210–211.

Habermann, A. N. (1969), "Prevention of system deadlocks", *Communication of the ACM* **12**(7), 373–377.

Harrison, B. L., Ishii, H., Vicente, K. J. & Buxton, W. A. S. (1995), Transparent Layered User Interfaces : An Evaluation of a Display Design to Enhance Focused and Divided Attention, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'95)*, ACM Press, pp.317–324.

Harrison, S. & Dourish, P. (1996), Re-Place-ing Space : The Roles of Place and Space in Collaborative Systems, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, ACM Press, pp.67–76.

Henderson, D. A. & Card, S. (1986), "Rooms : the use of multiple virtual workspaces to reduce space contention in a window-based graphical user interface", *ACM Transactions on Graphics* **5**(3), 211–243.

Hinckley, K. & Sinclair, M. (1999), Touch-sensing input devices, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'99)*, ACM Press, pp.223–230.

Hollan, J. & Stornetta, S. (1992), Beyond Being There, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'92)*, ACM Press, pp.119–125.

Hollingsworth, D. (1995), The Workflow Reference Model, Technical Report, Workflow Management Coalition, http ://www.wfmc.org.

Ishii, H., Kobayashi, M. & Grudin, J. (1992), Integration of Inter-personnal Space and Shared Workspace : ClearBoard Design and Experiments, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'92)*, ACM Press, pp.33–42.

Johansen, R. (1988), *Groupware : Computer Support for Business Teams*, The Free Press.

Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C., Beard, M. & Mackey, K. (1989), "The Xerox Star : A retrospective", *IEEE Computer* **22**(9), 11–29.

Kabbash, P., Buxton, W. & Sellen, A. (1994), Two-handed input in a compound task, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'94)*, ACM Press, pp.417–423.

Kandogan, E. & Shneiderman, B. (1996), Elastic Windows : improved Spatial Layout and Rapid Multiple Window Operations, *in Proceedings of the Conference on Advanced Visual Interfaces (AVI'96)*, ACM Press, pp.29–38.

Karsenty, A. & Beaudouin-Lafon, M. (1993), An Algorithm for Distributed Groupware Applications, *in Proceedings of the International Conference on Distributed Computing Systems (ICDCS'93)*, pp.195–202.

Kistler, J. J. & Satyanarayanan, M. (1992), "Disconnected operation in the Coda File System", *ACM Transactions on Computer Systems* **10**(1), 3–25.

Knudsen, J. (1999), *Java 2D Graphics*, O'Reilly.

Kramer, A. (1994), Translucent patches dissolving windows, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'94)*, ACM Press, pp.121–130.

Krasner, G. E. & Pope, S. T. (1988), "A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80", *Journal of Objet Oriented Programming* pp.26–49.

Krause, R. (2001), "CVS : an introduction", *Linux Journal* **2001**(87), 3.

Kurtenbach, G. & Buxton, W. (1993), The limits of expert performance using hierarchic marking menus, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'93)*, ACM Press, pp.482–487.

Kurtenbach, G., Fitzmaurice, G., Baudel, T. & Buxton, B. (1997), The design of a GUI paradigm based on tablets, two-hands, and transparency, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'97)*, ACM Press, pp.35–42.

Lamport, L. (1978), "Time, clocks, and the ordering of events in a distributed system", *Communication of the ACM* **21**(7), 558–565.

Mackay, W. E. (1999), Media spaces : Environments for Informal Multimedia Interaction, *in* M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, Vol. 7 of *Trends in Software Series*, John Wiley & Sons, pp.55–82.

Mattern, F. (1989), Virtual time and global states of distributed system, *in Proceedings of the Workshop on Parallel and Distributed Algorithm*, pp.215– 226.

Maulsby, D. L., Witten, I. H. & Kittlitz, K. A. (1989), Metamouse : specifying graphical procedures by example, *in Proceedings of the ACM Conference on Computer Graphics (CG'89)*, ACM Press, pp.127–136.

Microsoft (1995), "The Component Object Model Specification", Specification Document.

Miller, T. & Zeleznik, R. (1998), An insidious Haptic invasion : adding force feedback to the X desktop, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'98)*, ACM Press, pp.59–64.

Moran, T. P. & Anderson, R. J. (1990), The workaday world as a paradigm for CSCW design, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'90)*, ACM Press, pp.381–393.

Myers, B. A. (1990), "A new model for handling input", *ACM Transaction on Information Systems* **8**(3), 289–320.

Myers, B. A. & Kosbie, D. S. (1996), Reusable hierarchical command objects, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'96)*, ACM Press, pp.260–267.

Myers, B., Giuse, D., Dannenberg, R. B., Zanden, B. V., Kosbie, D., Marchal, P. & Pervin, E. (1990), "Comprehensive support for graphical, highly-interactive user interfaces : The garnet user interface development environment", *IEEE Computer* **23**(11), 71–85.

Nardi, B. A., Whittaker, S. & Bradner, E. (2000), Interaction and Outeraction : Instant Messaging in Action, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, *CHI Letters* **2**(3), ACM Press, pp.79–88.

Neuwirth, C. M., Kaufer, D. S., Chandhok, R. & Morris, J. H. (1990), Issues in the design of computer support for co-authoring and commenting, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'90)*, ACM Press, pp.183–195.

Newman, R. & Pelimuhandiram, H. (1991), MACE : A Fine-grained Concurrent Editor, *in Proceedings of the ACM/IEEE Conference on Organizational Computing Systems (COCS'91)*, ACM Press, pp.240–254.

Norman, R. & Draper, S. (1986), *User Sentered System Design : new Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates.

OMG (2000), OMG Unified Modeling Language Specification, Technical specification, OMG.

Orfali, R., Harkley, D. & Edwards, J. (1996), *Objets répartis, guide de survie*, International Thomson Publishing.

Pekkola, S. (2002), Critical Approach to 3D Virtual Realities for Group Work, *in Proceedings of the NordCHI'2002 Conference*, ACM Press, pp.129–138.

Perlin, K. & Fox, D. (1993), Pad : an alternative approach to the computer interface, *in Proceedings of the ACM Conference on Computer Graphics (CG'93)*, ACM Press, pp.57–64.

Prakash, A. (1999), Group Editors, *in* M. Beaudouin-Lafon (ed.), *Computer Supported Cooperative Work*, Vol. 7 of *Trends in Software Series*, John Wiley & Sons, pp.103–133.

Prakash, R. & Singhal, M. (1997), "Dependency sequences and hierarchical clocks : efficient alternatives to vector clocks for mobile computing systems", *Wireless Networks* **3**(5), 349–360.

Preguiça, N., Martins, J. L., Domingos, H. & Duarte, S. (2000), Data management support for asynchronous groupware, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'00)*, *CHI Letters* **2**(3), ACM Press, pp.69–78.

Prince, S., Cheok, A. D., Farbiz, F., Williamson, T., Johnson, N., Billinghurst, M. & Kato, H. (2002), 3-D live : real time interaction for mixed reality, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'02)*, *CHI Letters* **4**(3), ACM Press, pp.364–371.

Rekimoto, J. (1997), Pick-and-Drop : A Direct Manipulation Technique for Multiple Computer Environments, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'97)*, ACM Press, pp.31–39.

Riveill, M. (1995), "Synchronising Shared Objects", *Distributed Systems Engeneering Journal* **2**(2), 112–125.

Roberts, D. (2000), RealPlaces, 3D Interfaces for Office Applications, *in Proceedings of the Conference on Tools for Working with Guidelines (TFWWG'00)*, Springer Verlag.

Root, R. W. (1988), Design of multi-media vehicle for social browsing, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'88)*, ACM Press, New York, pp.25–38.

Roseman, M. & Greenberg, S. (1996), TeamRooms : network places for collaboration, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'96)*, ACM Press, pp.325–333.

Roussel, N. (1999), "Mediascape : a Web-based Mediaspace", *IEEE Multimedia* **6**(2), 64–74.

Rubine, D. (1991), "Specifying gestures by example", *Computer Graphics* **25**(4), 329–337.

Rubine, D. (1992), Combining gestures and direct manipulation, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'92)*, ACM Press, pp.659–660.

Saito, Y. & Shapiro, M. (2002), Replication : Optimistic Approaches, Technical Report, HP Internet Systems and Storage Laboratory.

Sarkar, M. & Brown, M. H. (1992), Graphical fisheye views of graphs, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'92)*, ACM Press, pp.83–91.

Searle, J. (1969), *Speech Act Theory*, Cambridge University Press.

Shneiderman, B. (1983), "Direct manipulation : a step beyond programming languages", *IEEE Computer* **16**(8), 57–69.

Shneiderman, B. (1998), *Designing the User Interface, Strategies for Effective Human-Computer Interaction*, 3rd edition, Addison-Wesley.

Singhall, S. & Zyda, M. (1999), *Networked virtual environments : design and implementation*, Addison-Wesley.

Smith, R. B. & Taivalsaari, A. (1999), Generalized and stationary scrolling, *in Proceedings of the ACM Symposium on User Interface Software and Technology (UIST'99)*, *CHI Letters* **1**(1), ACM Press, pp.1–9.

Sohlenkamp, M. & Chwelos, G. (1994), Integrating Communication, Cooperation, and Awareness : The DIVA Virtual Office Environment, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94)*, ACM Press, pp.331–343.

Stefik, M., Bobrow, D. G., Foster, G., Lanning, S. & Tatar, D. (1987), "WYSIWIS revised : early experiences with multiuser interfaces", *ACM Transaction on Information Systems* **5**(2), 147–167.

Stewart, J., Bederson, B. B. & Druin, A. (1999), Single display groupware : a model for co-present collaboration, *in Proceedings of the ACM Conference on Human factors in Computing Systems (CHI'99)*, ACM Press, pp.286–293.

Streitz, N. A., Geißler, J., Haake, J. M. & Hol, J. (1994), DOLPHIN : integrated meeting support across local and remote desktop environments and LiveBoards, *in Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'94)*, ACM Press, pp.345–358.

Stults, R. (1986), Media Space, Technical report, Xerox PARC.

Sun (1997), "JavaBeans API specification", Specification document.

Sun (1999), Java Media Framework, API Guide, Sun Microsystem.

Sun, C., Jia, X., Zhang, Y., Yang, Y. & Chen, D. (1998), "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems", *ACM Transaction on Computer-Human Interaction* **5**(1), 63–108.

Szyperski, C. (2002), *Component Software : Beyond Object-Oriented Programming*, 2nd edition, Addison-Wesley.

W3C (2000), Extensible Markup Language (XML) 1.0 (second edition), W3C Recommendation, Consortium W3C.

W3C (2002), Document Object Model (DOM) Level 2 HTML Specification, W3C Recommendation, Consortium W3C.

W3C (2003), Scalable Vector Graphics (SVG) 1.1 Specification, W3C Recommendation, Consortium W3C.

Wellner, P. (1993), "Interacting with paper on the DigitalDesk", *Communication of the ACM* **36**(7), 87–96.

Wellner, P., Mackay, W. & Gold, R. (1993), "Back to the real world", *Communication of the ACM* **36**(7), 24–27.

Wernecke, J. (1994), *The Inventor Mentor : Programming Object-Oriented 3D Graphics with Open Inventor, release 2*, Addison-Wesley.

Wittmann, R., Zitterbart, M. & Wittman, R. (2000), *Multicast Communication : Protocols, Programming, and Applications*, Morgan Kaufmann.

Annexe A

Exemples de mise en œuvre du modèle DPI

Dans cette annexe, nous décrivons un ensemble de composants DPI, principalement des instruments, afin d'illustrer d'une manière plus approfondie les principes du modèle. Nous serons ainsi amenés à affiner le modèle initial par quelques nouvelles considérations.

Pour réaliser ces différents composants, nous avons effectué la simulation de quelques idées sous la forme de *prototypes* animés (Beaudouin-Lafon & Mackay, 2002) réalisés avec l'application Flash de Macromedia. L'ensemble des "mini-scénarios" proposés à travers ce chapitre a été implanté en utilisant OpenDPI.

A.1 Composition graphique et généricité des actions

A.1.1 Problème

Si elle accroît les possibilités interactives du point de vue de l'utilisateur, la généricité des actions induit un problème lorsque nous l'appliquons sur les graphes de scène. En effet, une action générique, définie comme consommable pour le type T, est susceptible d'être effectivement consommée par toutes les instances n de T quel que soit le contexte de n, *i.e.* sa position dans le graphe de scène. Considérons par exemple le graphe d'une fenêtre :

- Une fenêtre est une instance de Rectangle.
- Elle contient une barre de titre instance de Rectangle qui, à sont tour, contient une instance de Text.
- Elle inclut également un rectangle définissant le contenu de la fenêtre et instance de ClipRectangle.

Les trois classes Rectangle, ClipRectangle et Text définissent la consommation de l'action Translate puisqu'elles dérivent de Graphic. L'utilisateur peut alors déplacer soit la fenêtre dans son ensemble, soit l'un de ses trois éléments (la barre de titre, le titre ou le contenu). Dans le deuxième cas, la géométrie de la fenêtre se trouvera probablement dénaturée. Il est bien sûr possible, pour une telle fenêtre, de définir un ensemble de contraintes comme cela a été fait pour le rectangle composite (section 4.1.5.3), mais cette solution s'avère lourde. Une solution plus légère consiste à *verrouiller* les propriétés adéquates de certains nœuds fils, et d'interdire la production d'une action lorsque les propriétés qu'elle modifie sont verrouillés. Dans l'exemple de la fenêtre, les propriétés dites géométriques, *i.e.* x, y, width,

height, rotation et scale, sont verrouillées lors de sa construction pour ses trois éléments constitutifs.

L'adoption de cette deuxième solution nous conduit à affiner, d'une part, le modèle des propriétés et, d'autre part, le modèle des actions. Notons que le verrouillage de propriété n'a pas été initialement défini afin de résoudre le problème posé ici ; il a été considéré afin de garantir la cohérence de l'interaction, notion qui sera abordée dans le chapitre 5.

A.1.2 Affinement du modèle des propriétés

Toute propriété d'un composant peut être verrouillée et déverrouillée. La classe Component définit pour ce faire une table de verrous qui associe à chaque propriété un verrou. La syntaxe définissant les propriétés par les accesseurs get et set est alors enrichie de deux méthodes permettant respectivement de verrouiller et déverrouiller une propriété et de déterminer si la propriété est éditable[1]. Par exemple, la propriété x de la classe Graphic définit les deux méthodes suivantes :

```
public void  setXLocked(boolean locked) {
   setPropertyLocked("x", locked) ;
}
public boolean  isXEditable() {
   return isPropertyEditable("x") ;
}
```

Dans l'exemple de la fenêtre et de l'action générique Translate, la fenêtre verrouille les propriétés géométriques (méthode setGeometryLocked) de ces éléments fils lors de sa création.

Ce modèle de verrouillage des propriétés sera une nouvelle fois affiné lorsque nous aborderons le problème des actions locales et simultanées qui apparaît en interaction bimanuelle et en collaboration locale sur un écran partagé (chapitre 5).

A.1.3 Affinement du modèle des actions

Le modèle des actions doit à son tour être affiné de manière à prendre en considérant l'existence des verrous. En effet, l'action qui modifie des propriétés préalablement verrouillées *ne doit pas* pouvoir être produite. Nous modifions en conséquence le cycle de production de telle sorte qu'une action ne puisse être produite que si elle peut être consommée, ce qui est le cas lorsque les propriétés concernées ne sont pas verrouillées (figure A.1). La réponse à la question "can the action be consumed ?" s'effectue en considérant une quatrième méthode, dite *méthode de faisabilité*, définie dans l'interface de consommation de l'action ; elle permet au producteur d'interroger le consommateur quant à la faisabilité de la production de l'action. La méthode de faisabilité est préfixée par le mot clé "can" suivi du nom de l'action. Par exemple, l'interface de consommation de l'action Translate est complétée comme suit :

```
public interface TranslateConsumer extends Consumer {
    public boolean  canTranslate() ;
   public void beginTranslate(TranslateProducer producer) ;
   public void endTranslate(TranslateProducer producer) ;
   public void doTranslate(double x, double y) ;
}
```

[1] A cette étape du modèle, une propriété est éditable si et seulement si elle n'est pas verrouillée.

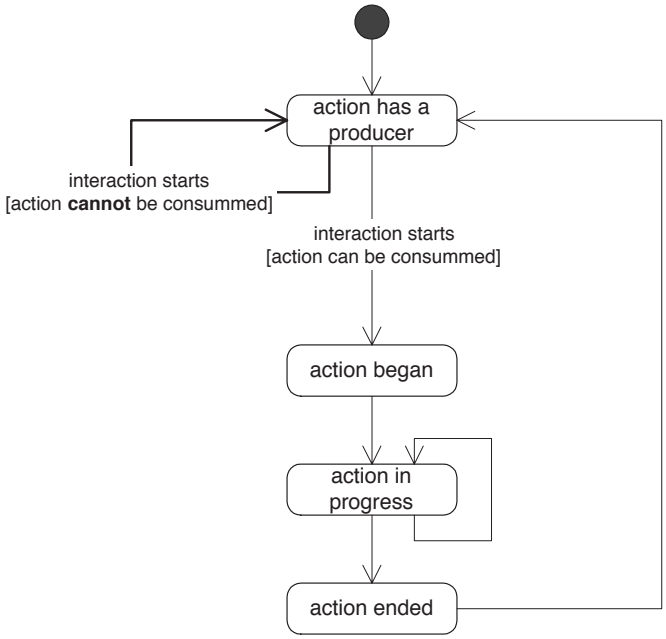


FIG. A.1 – Cycle de production d'une action

La méthode de faisabilité, implantée par le consommateur de l'action, effectue typiquement un test sur le verrouillage des propriétés impliquées par l'action. Par exemple, l'action Translate de la classe Graphic est définie comme suit :

```
public boolean canTranslate(){
   return isXEditable() && isYEditable() ;
}
```

A.2 Défaire et refaire

A.2.1 Problème

Dans un environnement axé sur les instruments d'interaction et, par conséquent, sur la manipulation directe, toutes les actions utilisateurs doivent pouvoir être annulées et rejouées (Shneiderman, 1983). Le mécanisme de production des actions doit donc prendre en considération la capacité d'*annuler* et de *ré-exécuter* chaque action.

Nous avons vu dans la section 4.2.1.2 l'intérêt qu'il y a à ce que les actions soient polymorphes. Le caractère polymorphe a cependant, du point du vue du programmeur, un coût non négligeable. En effet, l'annulation d'une action consistant à appliquer une *opération inverse* annulant son effet, la fourniture de l'opération inverse ne peut être faite *que* par le consommateur de l'action : ce dernier fixant lui-même la sémantique de l'action, il est le seul habilité à fournir une opération inverse conforme à cette sémantique.

Notons que nous avons simplifié le plus possible l'implantation de l'annulation dans OpenDPI : l'historique est géré à un niveau global (*i.e.* l'historique est unique et est lié à l'espace de travail dans son ensemble), l'annulation n'est ni sélective (Berlage, 1994), ni hiérarchique (Myers & Kosbie, 1996).

A.2.2 Affinement du modèle des actions

Nous devons, afin d'agrémenter le modèle DPI de l'annulation / ré-exécution des actions, enrichir le modèle des interfaces de consommation. Nous ajoutons à cette interface deux nouvelles méthodes, appelées *méthode d'annulation* et *méthode de ré-exécution*, dont la syntaxe est illustrée par l'exemple suivant :

```
public interface TranslateConsumer extends Consumer {
  public boolean canTranslate() ;
  public void beginTranslate(TranslateProducer producer) ;
  public void endTranslate(TranslateProducer producer) ;
  public void doTranslate(double x, double y) ;
   public void  undoTranslate(ArgumentStack arguments) ;
   public void  redoTranslate(ArgumentStack arguments) ;
}
```

De par le côté polymorphe des actions, les arguments passés aux méthodes d'annulation et de ré-exécution ne peuvent être précisés par la classe d'action elle-même. En conséquence, les arguments sont renseignés par le consommateur de l'action d'une manière *générique* au travers de la classe ArgumentStack.

A.2.3 Action de translation

Les arguments passés aux méthodes d'annulation et de ré-exécution sont construits généralement et respectivement au début et à la fin de l'action. Par exemple, la classe Graphic gère l'annulation et la ré-exécution de l'action Translate comme suit :

```
public void  beginTranslate(TranslateProducer producer) {
  //...
  HistoryEntry entry = History.self().addEntry(Translate.class, producer, this) ;
  entry.getUndoArguments().addDouble(getX()) ;
  entry.getUndoArguments().addDouble(getY()) ;
}
public void  endTranslate(TranslateProducer producer) {
  //...
  HistoryEntry entry = History.self().getEntry(Translate.class, this) ;
  entry.getRedoArguments().addDouble(getX()) ;
  entry.getRedoArguments().addDouble(getY()) ;
}
public void doTranslate(double dx, double dy) {
  setLocation(getX() + dx, getY() + dy) ;
}
public void  undoTranslate(ArgumentStack arguments) {
  setLocation(arguments.getDouble(0), arguments.getDouble(1)) ;
}
public void  redoTranslate(ArgumentStack arguments) {
  setLocation(arguments.getDouble(0), arguments.getDouble(1)) ;
}
```

Lorsque l'action translate débute, les propriétés x et y du Graphic à translater sont sauvegardées dans la section "arguments d'annulation" (méthode getUndoArguments) de l'entrée (entry) de l'historique global (singleton Historic.self()). De même, les arguments de la ré-exécution sont mémorisés à la fin de la translation. Lorsque l'utilisateur effectue l'annulation de la dernière action, l'historique global invoque la méthode d'annulation sur la cible sauvegardée dans l'entrée associée à cette action, en lui passant les arguments sauvegardés dans cette même entrée. Un principe analogue est utilisé pour invoquer la méthode de ré-exécution.

A.2.4 Action de peindre

Si l'action Translation n'est *a priori* que peu polymorphe et reste inchangée dans les classes dérivées de Graphic, le côté polymorphe de l'action Paint peut s'illustrer en considérant respectivement les consommateurs Graphic et Image. Peindre une instance de Graphic consiste à lui appliquer une couleur principale, laquelle correspond à la couleur de fond pour une forme géométrique vectorielle (classe Shape) ou la couleur du trait pour un texte (classe Text). Par contre, l'idée de couleur principale n'ayant pas de définition directe pour une image formée de pixels colorés (classe Image), l'action de peindre une image peut consister, par exemple, à ajouter la couleur fournie par l'action à chaque pixel, l'image finale affichant alors une teinte dominante égale à cette couleur[2].

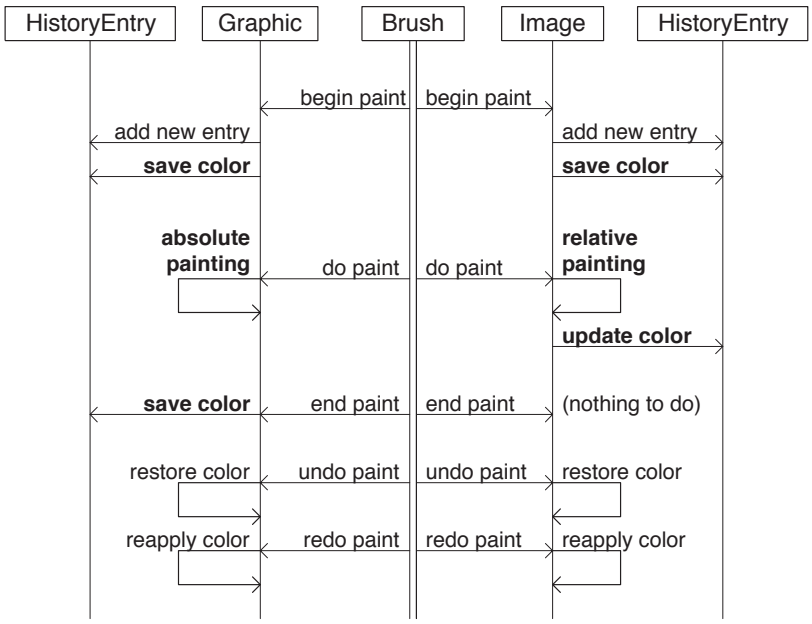


FIG. A.2 – Action "peindre" pour une forme graphique (à gauche) ou une image (à droite)

La figure A.2 donne le diagramme de séquence de la production de l'action "peindre" pour les classes Graphic et Image. Nous voyons qu'il subsiste quelques différences de séquencement entre les deux classes : dans le cas du graphique, sa couleur d'origine est sauvegardée au début de l'action (méthode beginPaint) alors que, pour l'image, la couleur "ajoutée" est mise à jour à chaque étape d'application de la couleur (méthode d'exécution doPaint).

Cet exemple illustre que le polymorphisme des actions peut induire des variations sur la séquence de sauvegarde dans l'historique. Le principe général de la propagation de l'annulation et de la ré-exécution reste cependant relativement homogène.

A.3 Instruments semi-directs

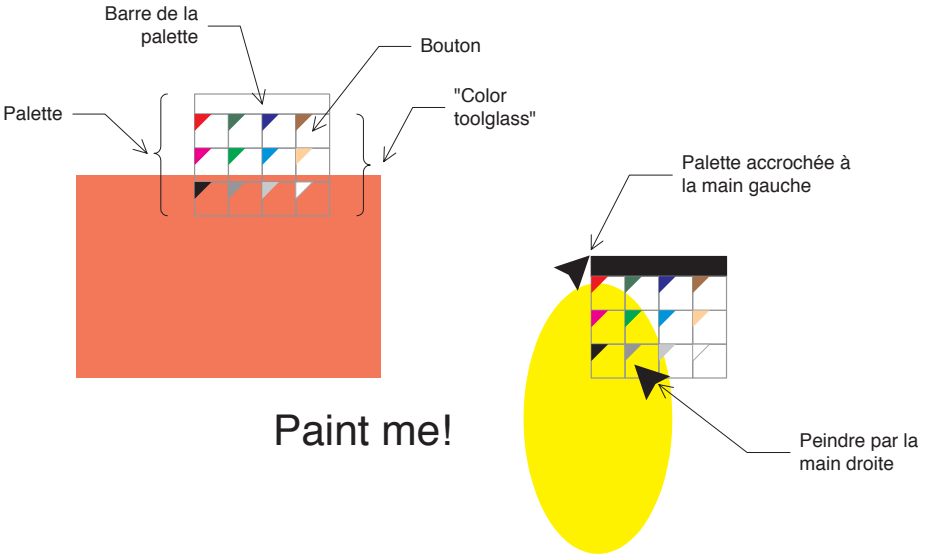
A.3.1 Définitions

Les instruments semi-directs sont les instruments qui induisent une distance logique positive et une distance géométrique nulle dans la chaîne de production des actions. La distance

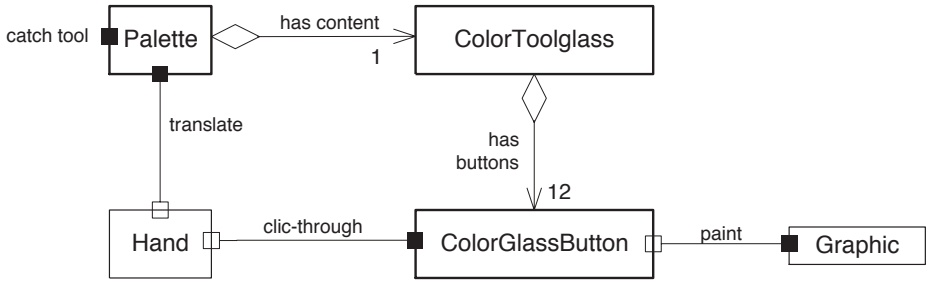
[2]L'ajout de la couleur à chaque pixel n'est pas nécessairement une addition des trois composantes RVB.

logique positive est caractérisée par la présence d'une action intermédiaire dans la chaîne de production de l'action intentionnelle. Nous allons illustrer, dans les trois exemples suivants, l'impact de cette action intermédiaire sur le mécanisme de production de l'action et, en particulier, sur l'annulation et la ré-exécution.

A.3.2 Toolglass



(a) Instrument en action



(b) Notation composant

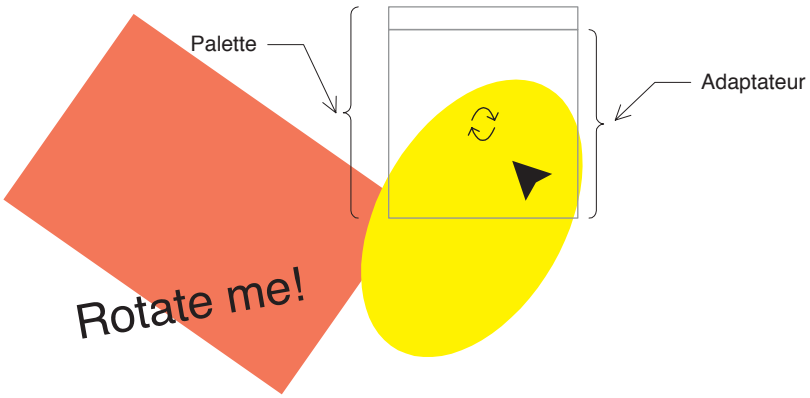
FIG. A.3 – Toolglass pour peindre

Le toolglass d'application d'une couleur (classe ColorToolglass) permet la transformation de l'action de "clic-au-travers" en une action "peindre" (figure A.3). Il est composé de bouton transparents (classe ColorGlassButton) susceptibles de produire l'action peindre sur l'objet ciblé par le "clic-au-travers". Le toolglass est disposé sur une palette (classe Palette) qui permet, *via* sa barre de manipulation, de déplacer la toolglass sur un graphique sans pour autant générer l'action clic-au-travers. Par ailleurs, la palette peut être "accrochée" à un outil (classe CatchTool) de manière à devenir solidaire de cet outil et à permettre une interaction bimanuelle plus efficace.

L'action intermédiaire est, dans cet exemple, le clic-au-travers (classe ClickThrough). Cette action suggérant la notion de geste, elle est considérée comme non annulable (l'interface

de consommation ne définit donc pas les méthodes d'annulation et de ré-exécution). Ceci résout du même coup le problème de l'annulation de l'action intermédiaire : seule l'action générée par la clic-au-travers ("peindre" dans le cas du ColorToolglass) est enregistrée dans l'historique.

A.3.3 Adaptateur



(a) Instrument en action

(b) Notation composant

FIG. A.4 – Adaptateur *translation* $\rightsquigarrow$ *rotation*

La figure A.4 présente l'exemple d'un adaptateur transformant une action de translation en une action de rotation. Le principe de fonctionnement de cet adaptateur reste assez analogue à celui des toolglasses : il s'agit d'un instrument transparent qui consomme l'action de translation et la transmet à l'objet visé par l'adaptateur. Ce chaînage des actions de translation et de rotation est ainsi une substitution : l'action de rotation se substitue à l'action de translation, l'adaptateur fournissant la relation de compatibilité translation $\rightsquigarrow$ rotation (section 3.3.1.2).

Le principe de l'adaptateur translation $\rightsquigarrow$ rotation peut être aisément généralisé à tout adaptateur $A \rightsquigarrow B$, comme le suggère le diagramme de séquence de la figure A.5 :

1. La faisabilité de l'action A est demandée à l'adaptateur. Ce dernier questionne alors le consommateur de l'action B sur la faisabilité de B : si B est faisable, cela signifie que l'action A peut être produite sur l'adaptateur.
2. Lorsque l'action A est consommée par l'adaptateur, ce dernier produit l'action B sur le consommateur final. Lorsque la méthode d'exécution est invoquée, il y a transformation de l'action A en action B par une *adaptation des paramètres* de A vers les paramètres de B .

Ainsi, l'adaptateur $A \rightsquigarrow B$ délègue chaque invocation des méthodes de l'interface de consommation de A au consommateur de B . Cette délégation n'est par contre plus de

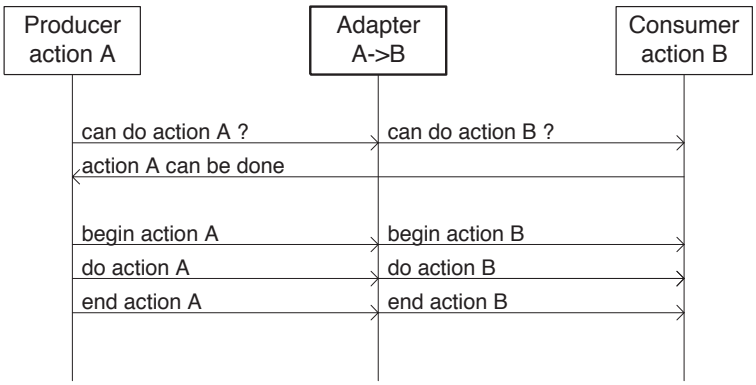


FIG. A.5 – Fonctionnement d'un adaptateur

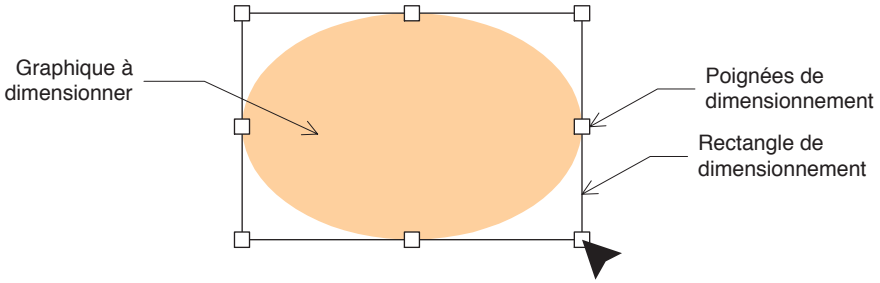
mise pour l'annulation et la ré-exécution : seule l'action B est enregistrée dans l'historique par son consommateur, l'adaptateur n'effectuant aucune entrée dans l'historique. Ainsi, l'annulation de l'action intentionnelle B peut avoir lieu sans se préoccuper de l'action intermédiaire A substituée.

Notons qu'un adaptateur est relativement simple à construire : le code peut être généré par un compilateur rudimentaire une fois l'adaptation des paramètres des deux actions précisées. Ainsi, tout comme l'utilisateur a la possibilité de définir lui-même l'adaptation en entrée de ses instrument (*via* le profil d'interaction), il devrait pouvoir sans trop de peine construire des adaptateurs $A \rightsquigarrow B$ et donc définir l'adaptation en sortie de ses instruments.

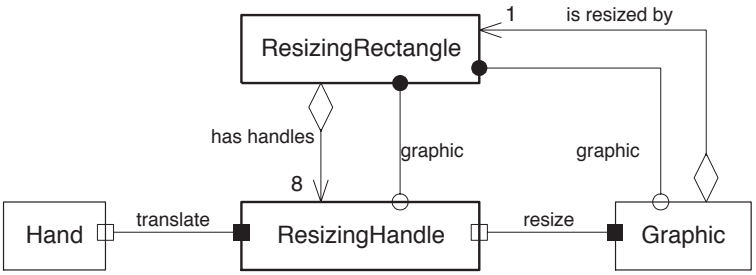
A.3.4 Poignées de dimensionnement

Les poignées de dimensionnement (classe ResizingHandle) sont des instruments semi-directs transformant une action de translation en une action de dimensionnement sur un graphique ciblé (figure A.6). Les poignées sont reparties autour d'un rectangle de dimensionnement (classe ResizingRectangle) offrant huit poignées. Le rectangle observe les translations de ses poignées (*via* l'observation de leurs positions) afin de contraindre leurs positions respectives tout autour du rectangle. Le rectangle de dimensionnement est un instrument placé à l'intérieur même du graphique, toute transformation géométrique appliquée à ce graphique se répercutant sur l'instrument.

De la même manière qu'un adaptateur $A \rightsquigarrow B$ n'enregistre pas l'action A dans l'historique, la poignée de dimensionnement ne sauvegarde pas l'action de translation qu'elle sait consommer. L'annulation et la ré-exécution de l'action intentionnelle ne pose donc, là encore, pas de problème puisqu'elles sont directement invoquées sur le graphique ciblé. Cependant, lorsqu'une action est annulée ou ré-exécutée, seul le graphique est mis à jour, ce dernier ne connaissant pas le rectangle de dimensionnement qui lui est associé (principe d'indépendance). Une observation des propriétés width et height du graphique permet de remédier à ce problème (lien d'observation du graphique par le rectangle de dimensionnement dans la figure A.6).



(a) Instrument en action



(b) Notation composant

FIG. A.6 – Poignées de dimensionnement

A.4 Actions composées

A.4.1 Définition

Certaines interactions nécessitent plusieurs étapes pour leur réalisation, la transition d'une étape à la suivante étant assurée par une action élémentaire. Par exemple, l'action composée de "glisser-déposer" un objet ("*drag and drop*") se décompose en trois actions élémentaires et successives : la prise de l'objet, sa translation (le "glisser") et sa dépose.

Nous affinons le modèle des actions en définissant les actions composées comme suit :

> *Une* action composée *est un ensemble d'actions élémentaires. La production d'une action composée consiste à produire indépendamment chaque action élémentaire dans un ordre déterminé par l'action composée.*
>
> *Corollairement, une* action élémentaire *est une action qui n'est* pas *composée.*

Les classes Action et CompositeAction précisent les notions respectives d'action élémentaire et d'action composée ; elles dérivent de la classe de base Consumable qui définit les entités consommables (et productibles). La notation des composants est en conséquence étendue comme l'indique la figure A.7 :

- L'action composée ac est productible pour le composant A et consommable pour le composant B. Cette action se compose de deux actions élémentaires ae1 et ae2.
- Le composant A sait produire ae1 et ae2 indépendamment de ac : les deux broches ae1 et ae2 sont indiquées pour A.
- La relation de dépendance entre ac et (ae1, ae2) est indiquée par un lien curviligne.

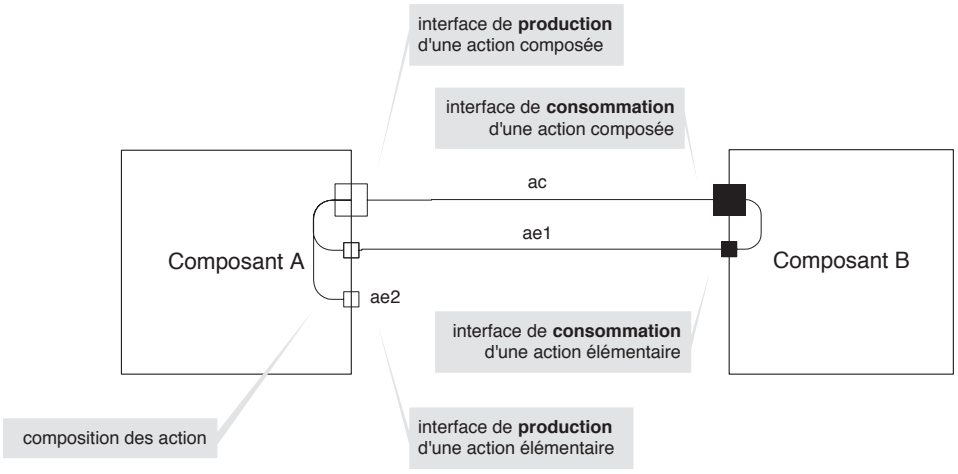


FIG. A.7 – Notation des actions composites

– Le composant B sait consommer l'action ae1 indépendamment de ac. Par contre, il ne sait pas consommer l'action ae2 autrement que dans le contexte de la consommation de l'action ac : la broche ae2 n'est ainsi pas représentée pour B mais figure implicitement "à l'intérieur" de la broche ac.

La section suivante illustre l'affinement du modèle des actions en considérant le "glisser-déposer". Elle précise en particulier les interfaces de production et de consommation des actions composites.

A.4.2 Glisser-déposer

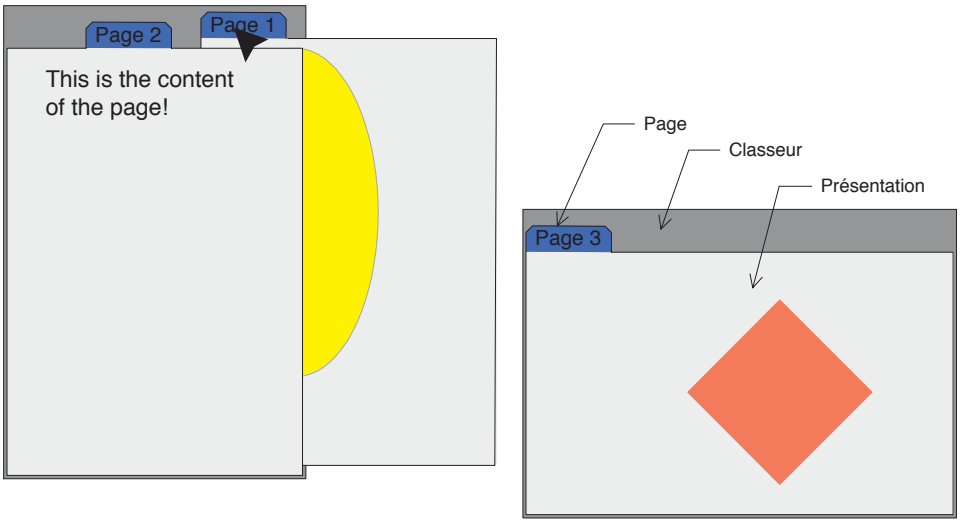
L'action de "glisser-déposer" concerne par exemple les pages qui peuvent être déplacées d'un classeur à l'autre (figure A.8) dont le principe se résume ainsi :

1. Une page (classe Page) sait consommer les deux actions prendre et glisser-déposer (classes Take et DragAndDrop). Par contre, elle ne gère pas l'action de translation en dehors d'un contexte de glisser-déposer (classe Translate).
2. Lorsque la page consomme l'action prendre, elle peut alors consommer l'action de translation et se déplacer sur l'espace de travail.
3. Lors de ce déplacement, la page réalise un piquer afin de déterminer quel objet se situant dessous est susceptible de l'accueillir.
4. Lorsqu'un tel objet a été trouvé et lorsque l'action prendre se termine (équivalent du "lâcher"), la page produit l'action "lâcher page" (classe DropPage) vers cet objet. Dans l'implantation actuelle de OpenDPI, seul le classeur (classe Binder) sait consommer l'action DropPage.

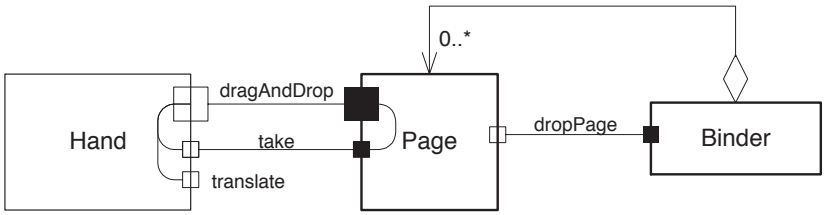
Les interfaces de consommation et de production de l'action DragAndDrop sont définies ainsi :

```
public interface DragAndDropProducer
   extends TakeProducer, TranslateProducer {
}

public interface DragAndDropConsumer
```



(a) Instrument en action



(b) Notation composant

FIG. A.8 – Glisser-déposer d'une page vers un classeur

```
    extends TakeConsumer, TranslateConsumer {
}
```

Ces deux interfaces sont simplement utiles pour marquer que la main sait produire l'action DragAndDrop et que la page sait la consommer. La classe DragAndDrop définit les deux actions élémentaires dont elle est composée :

```
public class DragAndDrop extends CompositeAction {
  public DragAndDrop(DragAndDropProducer producer,
                     Take take, Translate translate {
    super(producer, new PrimitiveAction[]{take,translate}) ;
  }
}
```

Notons que la définition d'une classe d'action composée, bien qu'elle soit triviale, est indispensable en vertu du principe d'indépendance des composants DPI. En effet, imaginons qu'un composant puisse consommer les actions Take et Translate indépendamment [3] au travers des deux interfaces de consommation TakeConsumer et TranslateConsumer. Si la main suppose que le simple fait de définir ces deux interfaces signifie que le composant supporte le glisser-déposer, alors elle sera susceptible de produire les deux actions Take et

[3] Par exemple, l'action Take de la page consiste à l'amener au premier plan de son classeur.

Translate *de manière dépendante*, et leur consommation *indépendante* risque de ne pas réaliser un glisser-déplacer correct. L'interface de consommation d'une action composée est donc indispensable pour préciser que le consommateur sait prendre en compte le *contexte* de consommation de l'action composée.

Par ailleurs, une action composée ne saurait être produite à la manière des actions élémentaires, c'est à dire au début de l'interaction qui est à son origine. Par exemple, au moment où elle produit l'action Take sur une page, la main ne sait pas s'il s'agit d'une action qui sera consommée pour une mise au premier plan ou pour un un glisser-déposer.

La gestion de l'annulation et de la ré-exécution des actions composées ne pose pas de problème particulier. L'action intentionnelle DropPage étant la dernière à être enregistrée dans l'historique, l'annulation de glisser-déposer commence par une invocation de undoDropPage. Cette invocation effectue successivement l'annulation des deux dernières actions associées qui sont nécessairement Take et Translate. Les méthodes undoTake et undoTranslate prennent alors en charge le re-positionnement correct de la page dans son classeur initial. La ré-exécution s'effectue selon un principe analogue : la méthode redoTake invoque les méthodes de ré-exécution de Translate puis DropPage.

A.5 Instruments autonomes

A.5.1 Définition

Dans notre définition de l'instrument (section 3.2.1 on page 80), nous avons souligné que l'acteur est l'entité qui souhaite agir sur l'objet d'intérêt *via* un instrument. L'acteur n'est pas nécessairement l'utilisateur : il peut s'agir du système. Le guide magnétique est un exemple caractéristique de ce type d'instrument que nous appelons *instruments autonomes*. Le guide magnétique, une fois créé par l'utilisateur, devient un instrument autonome puisqu'il est susceptible d'agir indépendamment des actions utilisateur.

A.5.2 Guide magnétique

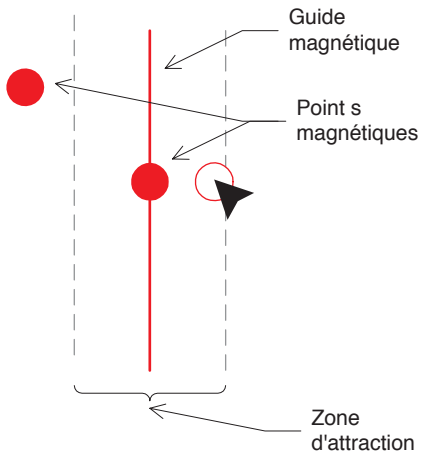


FIG. A.9 – Guide magnétique

Le guide magnétique vertical (classe VerticalMagneticGuide) est un instrument qui émet régulièrement l'action "attire" (classe AttractHorizontally) à l'intérieur de la zone d'attraction qui le caractérise (figure A.9). Il est alors possible de définir un point magnétique (classe MagneticPoint) qui consomme une telle action : lorsqu'il pénètre dans la zone d'attraction, le point se place sur l'axe et y reste collé tant que l'instrument qui le déplace n'a pas franchi cette zone.

Le guide magnétique est donc bien un instrument autonome car il ne produit pas l'action d'attraction à l'initiative de l'utilisateur, mais à fréquence régulière. Lorsqu'il réalise son piquer, le guide magnétique effectue un piquer sur la zone d'attraction ; il peut alors initier l'action d'attraction sur les objets entrant dans son champ d'attraction et la finaliser sur les objets qui en sortent.

La seule différence entre un instrument autonome et les autres instruments est qu'il vit à l'intérieur d'un processus léger qui lui est propre.

A.6 Protocoles des producteurs

A.6.1 Problème

Dans les exemples d'instruments donnés jusqu'ici, les interfaces de production des actions étaient vides. Lors rôle était de typer fortement les producteurs et les consommateurs d'une action donnée : les producteurs et les consommateurs d'une action Aaaa sont *nécessairement* des instances des classes respectives AaaaProducer et AaaaConsumer.

Les interfaces de production ne sont cependant pas toujours vides. En particulier, le producteur d'une action peut exiger que le consommateur lui fournisse une *réponse* en retour de la production de l'action. Dans ce cas, le consommateur effectue ce retour en invoquant sur le producteur l'une des méthodes définies dans l'interface de production.

Notons que le polymorphisme des actions implique que le consommateur sur lequel est produite l'action peut, en *théorie*, effectuer un traitement sans fournir un retour au producteur. Cependant, dans la *pratique*, le consommateur doit respecter le protocole imposé par l'action, et plus particulièrement par son interface de production. Ceci induit un couplage plus fort entre producteur et consommateur, conséquence d'une sémantique de l'action fortement suggérée par l'action elle-même (nous pourrions dire qu'une action à interface de production non vide n'a pas un caractère fortement polymorphe).

Par exemple, l'action de clonage consiste à demander au consommateur qu'il fournisse au producteur un clone, le producteur prenant alors en charge l'ajout du clone dans l'espace de travail. La valeur de retour correspond ici au clone de l'objet. Un autre exemple concerne la sélection d'objet : l'action de sélection demande aux consommateurs de s'enregistrer auprès du producteur pour qu'ils fassent partie de la sélection, le producteur définissant lui-même la manière dont les consommateurs sélectionnés sont regroupés.

Nous présentons dans les trois sections suivantes des instruments qui produisent des actions nécessitant un protocole de production particulier.

A.6.2 Cloneurs

L'action de clonage ne définit pas d'autre paramètre que le producteur de l'action. Celui-ci implante l'interface de production suivante :

```
public interface CloneProducer extends Producer {
    public void pushClone(Graphic clonedGraphic) ;
    public void popClone() ;
}
```

Lorsque le consommateur reçoit l'action de clonage *via* la méthode doClone, il se doit de fournir la réponse par une invocation de la méthode pushClone en passant son clone en argument.

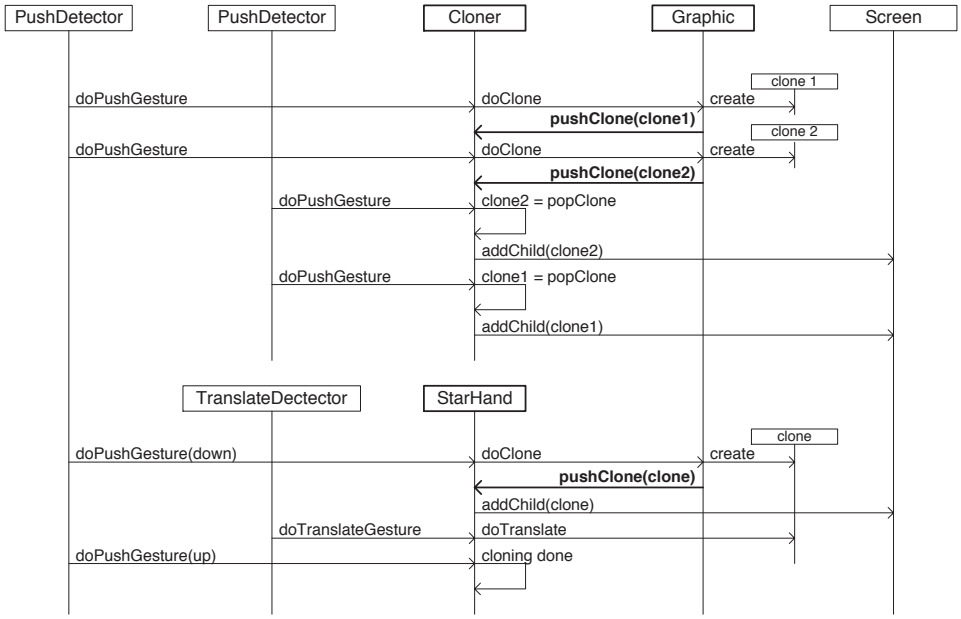


FIG. A.10 – Fonctionnement de deux cloneurs

La figure A.10 illustre le principe de l'action clone lorsqu'elle est produite par deux instruments différents, le cloneur (classe Cloner) et la main "façon Xerox Star" (classe StarHand) :

– L'instrument cloneur :

1. Le cloneur consomme l'appui sur le bouton gauche de la souris et, lorsqu'un consommateur de l'action clone est piqué, la méthode doClone est invoquée sur ce consommateur.
2. Le graphique consommateur crée alors le clone nommé clone 1, puis l'envoie au producteur par une invocation de pushClone.
3. Lorsqu'un deuxième appui sur le bouton gauche de la souris survient, la même séquence a lieu, ce qui conduit à l'empilement du clone 2 côté producteur.
4. Le cloneur consomme maintenant l'appui sur le bouton droit de la souris, ce qui a pour effet de dépiler le clone 2 puis de le positionner sur l'espace de travail (classe Screen). Au deuxième appui, c'est le clone 1 qui est alors placé.

– L'instrument main "façon Xerox Star" :

1. La main, lorsque le bouton de la souris est enfoncé, demande au graphique piqué de fournir un clone qui est alors ajouté à l'écran.
2. Lorsque la souris se déplace, le clone est translaté en conséquence (méthode doTranslate).

3. Quand le bouton de la souris est relâché, la translation s'interrompt et l'action de clonage est terminée.

Cet exemple illustre comment une même action peut être réalisée par deux interactions très différentes. Dans le cas du cloneur, il y a accumulation de plusieurs clones (dans une pile) puis insertion dans l'espace de travail : il est alors possible de cloner successivement plusieurs objets puis de les placer ultérieurement à des endroits spécifiques. Dans le cas de la main "Star", le clonage consiste à préciser l'objet à cloner et la position d'insertion du clone en un seul geste de type "glisser-déposer".

Nous avons justifié, du fait du polymorphisme des actions, la nécessité de définir l'annulation et la ré-exécution d'une action au niveau du consommateur, par opposition au niveau du producteur. L'interface de production va permettre au consommateur de réaliser l'annulation en invoquant une méthode de cette interface pour que le producteur participe à l'annulation ou à la ré-exécution. Par exemple, l'annulation de l'action clone consiste, pour le consommateur, à invoquer la méthode popClone sur le producteur. Pour la main Star, popClone retire le clone de l'endroit où il a été inséré, tandis que, pour le cloneur, cette méthode retire le clone de la pile.

A.6.3 Sélecteur et sélection

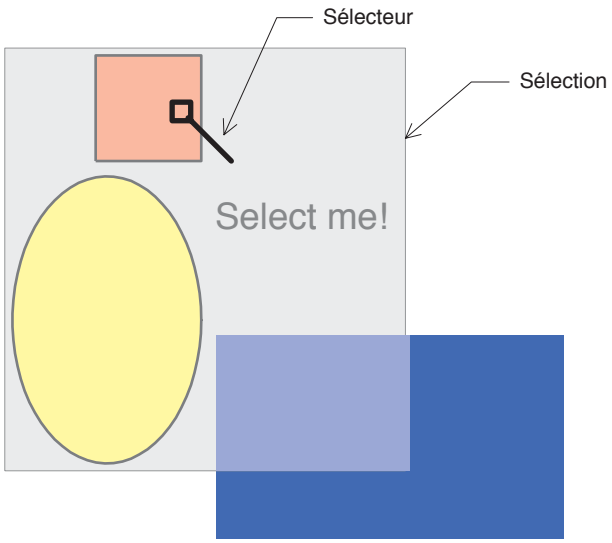


FIG. A.11 – Le sélecteur et sa sélection

La figure A.11 illustre le principe de l'instrument "sélecteur", lequel est constitué du curseur représentant le sélecteur à proprement parler et de la sélection rectangulaire qu'il créé :

1. Lorsqu'un graphique consomme l'action de sélection (classe Select), il demande au producteur de l'ajouter dans la sélection s'il n'y figure pas déjà, ou de le retirer sinon. Il utilise à cet effet les méthodes suivantes de l'interface de production :

```
public interface SelectProducer extends Producer {
   void addGraphic(Graphic graphic) ;
   void removeGraphic(Graphic graphic) ;
   boolean containsGraphic(Graphic graphic) ;
}
```

2. Le consommateur fournit un feed-back de son état de sélection : son contour est épaissi s'il est sélectionné ou revient à son état initial s'il ne l'est plus.
3. Le producteur, lorsqu'il reçoit la demande d'ajout ou de retrait, *délègue* le travail de sélection au rectangle de sélection : ce rectangle se met alors à jour en maintenant sa liste interne des objets sélectionnés et en ajustant sa géométrie de manière à englober l'ensemble des objets sélectionnés.
4. Lorsque l'utilisateur effectue une action sur le rectangle de sélection, telle que l'action peindre, cette dernière est appliquée à tous les éléments de la sélection.

L'annulation et la ré-exécution concernent le sélecteur *et* la sélection : le graphique qui reçoit l'ordre d'annuler sa sélection invoque la méthode removeGraphic sur le sélecteur, et l'annulation d'une action envoyée à la sélection se fait en annulant toutes les production de cette action vers les éléments sélectionnés dont une référence a été sauvegardée dans l'historique.

A.6.4 Stylo et curseur texte

L'édition d'un texte est typiquement réalisée par un instrument direct de type stylo. Ce dernier est composé du stylo lui-même, lequel se déplace en suivant un périphérique de pointage, ainsi que d'une plume, laquelle peut être immergée dans le texte lui-même afin de pouvoir s'y déplacer en fonction des actions faites au travers du clavier.

De manière à satisfaire le principe d'indépendance, nous avons défini une action d'édition de texte assez générale sous la forme d'une action composée (classe EditText) de trois actions élémentaires : l'insertion d'un caractère (classe InsertChararter), la suppression d'un caractère (classe DeleteCharacter) et le test de piquer d'un caractère (classe HitCharacter). La dernière action est purement informative et définit une interface de production non vide qui permet de connaître les caractéristiques géométrique du caractère situé à la position (x,y) donnée.

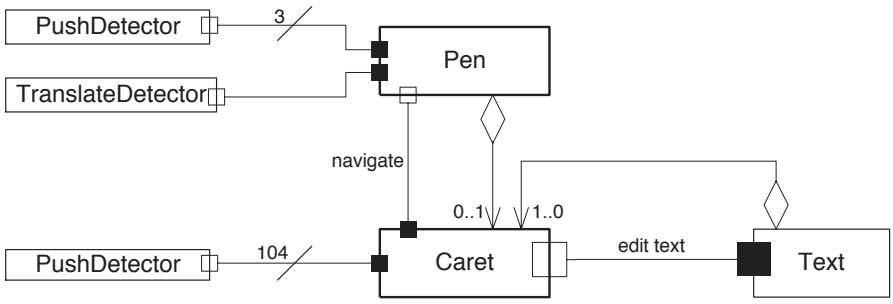


FIG. A.12 – Le stylo et son curseur texte

La figure A.12 montre la relation entre les deux parties constitutives du stylo, le corps du stylo (class Pen) et son curseur texte (la plume, classe Caret). Le stylo prend en charge tout déplacement du curseur : insertion et retrait du texte, déplacement à droite ou à gauche. Il consomme pour ce faire les gestes de translation, d'appui sur le bouton de la souris et d'appui sur les touches "flèche droite" et "flèche gauche" du clavier. Le curseur gère quant à lui l'édition du texte, *i.e.* l'ajout et le retrait de caractères, ainsi que la demande de piquer du texte.

L'édition de texte définissant des interactions assez complexes, nous ne détaillons pas ici les différents scénarios de fonctionnement. La sélection de caractères dans le texte n'est

pas implantée à l'heure actuelle et nécessite une approche similaire à la sélection (section précédente).

Il est important de remarquer que ni le stylo ni le curseur ne sont liés explicitement au type Text. La liaison stylo et curseur vers le texte se fait en respectant le principe d'indépendance, ce qui est rendu possible par l'adjonction de l'action informative HitCharacter. Ceci rend alors possible l'utilisation du stylo et de son curseur dans d'autres contextes. Par exemple, il est possible de définir l'édition d'un texte "invisible" (encre magique), ce qui peut être utile pour la saisie d'un mot de passe par exemple. Par ailleurs, les actions d'insertion et de suppression de caractères peuvent être produites *via* par des périphériques autres que le clavier, comme dans le cas de la dictée vocale.

A.7 Instruments de perception

A.7.1 Principe

L'instrument de perception, ou lentille, fournit une présentation alternative de celle proposée par le graphe de scène. Cette présentation est fournie par une caméra qui se déplace dans le graphe de scène. L'image fournie est alors affichée par la lentille, cette dernière étant elle-même présente dans le graphe de scène. Notons qu'il n'est pas possible que la lentille soit dans l'espace filmé par sa propre caméra puisqu'il pourrait en résulter une récursion infinie[4].

La lentille offre la capacité d'effectuer des zooms arrière et des zooms avant (loupe), ce qui peut-être utile pour des opérations nécessitant une grande précision (zoom avant) ou une vue d'ensemble (vue arrière). Les lentilles permettent également de fournir un *rendu* altéré des objets filmés. Par exemple, une lentille peut n'afficher que le contour des formes vectorielles, permettant ainsi de visualiser certains détails notamment lorsque des graphiques sont masqués par d'autres graphiques.

A.7.2 Loupe

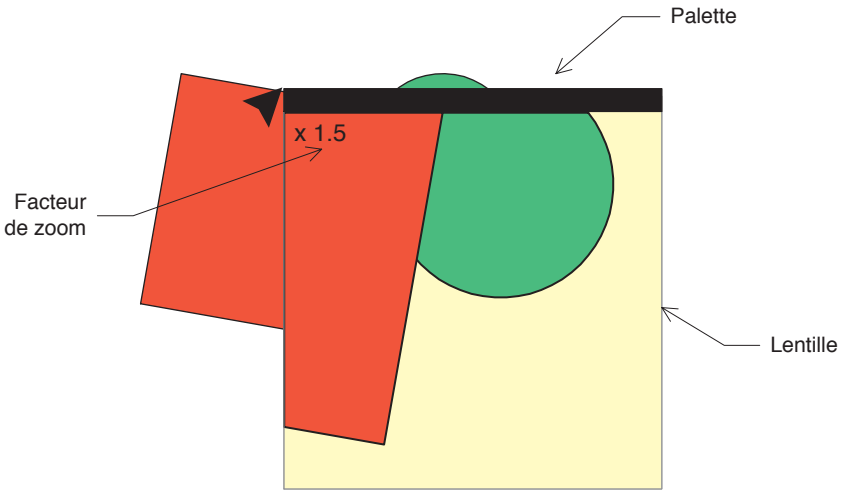
La figure A.13 représente le modèle simplifié d'une loupe (classe Lens). La loupe est un graphique associé à une caméra Piccolo[5] (non représentée) qui filme une ou plusieurs couches (classes Layer), chacune des couches contenant une partie du graphe de scène[6]. L'action translate est capturée de manière à permettre le déplacement conjoint de la loupe et de sa caméra, tandis que l'action de mise à l'échelle est redéfinie afin de modifier le facteur de zoom et non le facteur d'échelle du rectangle formant la loupe.

Une telle loupe offre l'intérêt de permettre "l'action-au-travers" : les actions ne sont pas appliquées en priorité à la loupe mais aux objets qu'elle affiche. Par exemple, il est possible d'effectuer la translation précise d'un graphique dont l'image est fortement grossie par la loupe en utilisant l'outil "main" *au dessus* de la loupe (le grossissement ne s'applique ainsi pas à l'outil). Cependant, dans une telle situation, la translation de l'outil engendrant un déplacement égal de l'objet ciblé, la perception grossie de ce déplacement conduit à une perception erronée de l'interaction. Cette perception erronée n'aurait pas lieu si l'outil se trouvait sous la loupe. Nous parlons dans pareil cas d'*interaction inconsistante*, terme qui sera défini dans le chapitre 5. Nous proposerons une approche qui rend l'interaction consistante au travers des loupes à la fin du chapitre 6.

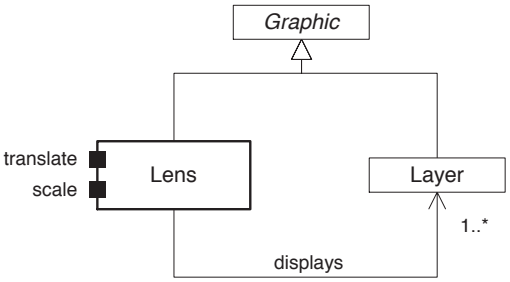
[4] Cette récursion infinie, difficilement possible à reproduire dans l'espace logique, reste naturelle dans l'espace physique comme dans le cas de deux miroirs en vis-à-vis.

[5] Rappelons que Piccolo est la boîte à outils utilisée par OpenDPI et qu'elle gère l'ensemble du graphe de scène.

[6] Les couches sont créées par le singleton représentant l'écran (Screen.self()) et en sont les nœuds fils.



(a) Instrument en action



(b) Diagramme de classes

FIG. A.13 – La loupe

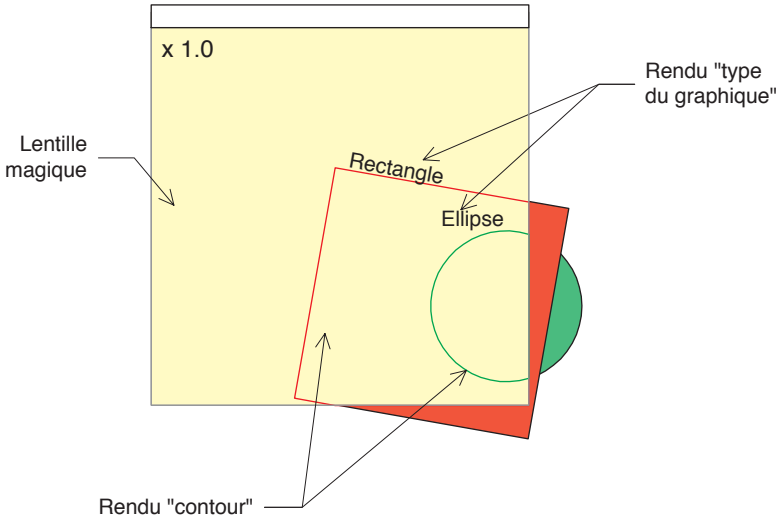
A.7.3 Lentille magique

La lentille magique (classe MagicLens) réalise un rendu en fonction des différents algorithmes de rendu (instances de Renderer) qu'il est possible de lui adjoindre. La figure A.14-a donne l'exemple d'une lentille dont le rendu s'effectue selon deux algorithmes : le premier réalise l'affichage des formes géométriques vectorielles (instances dérivées de Shape) en ne montrant que leur contour, le second affiche le nom du type du graphique.

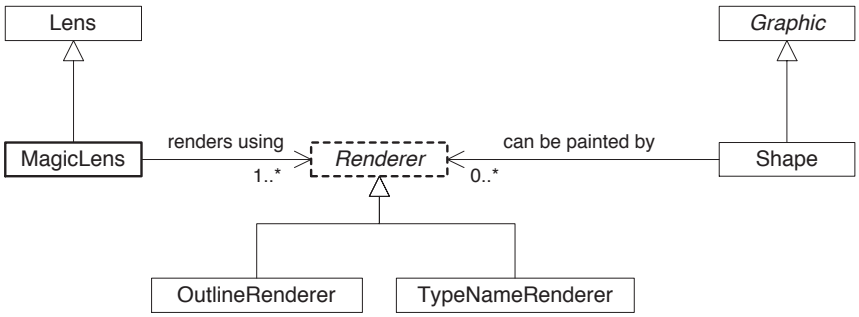
Le diagramme de classes de la figure A.14-b synthétise le principe du rendu effectué par une lentille magique :

1. Une lentille magique utilise au moins un algorithme de rendu. Chaque algorithme est appelé afin de réaliser le rendu des objets graphiques visés par la caméra associée à la lentille.
2. Toutes les formes géométriques peuvent être rendues d'une manière spécifique (*i.e.* de manière différente de celle prévue par défaut, cette dernière étant fournie par Piccolo) par ces algorithmes. Les classes Graphic et Shape ont été modifiées pour prendre en compte cette possibilité.

Un algorithme de rendu est relativement simple à écrire. Par exemple, le rendu affichant le contour des formes vectorielles peut être réalisé comme suit :



(a) Instrument en action



(b) Diagramme de classes

FIG. A.14 – La lentille magique

```
public class OutlineRenderer implements  Renderer  {
  public void  render(Graphics2D g2, Graphic graphic) {
    g2.setColor(graphic.getColor()) ;
    g2.setStroke(new BasicStroke(1)) ;
    if (graphic instanceof Shape)
      g2.draw(((Shape) graphic).getAwtShape()) ;
    else
      g2.drawRect(0, 0,
       (int) graphic.getWidth(),
       (int) graphic.getHeight()) ;
  }
}
```

La fonction render est invoquée à chaque fois que la représentation de l'instance graphic doit être mise à jour. Pour ce faire, l'argument g2 (le "contexte graphique") est passé en paramètre et permet de définir la manière dont va être "peint" graphic. Le code donné en exemple est ainsi très simple :

– Le contexte graphique précise d'utiliser la couleur de fond du graphique dans les opérations de rendu qui suivent, avec un "pinceau" de un pixel.

– Si le graphique à rendre est une forme géométrique vectorielle, le contour est peint (méthode draw) avec la couleur précisée précédemment.
– Sinon (cas du texte ou des images par exemple), un rectangle est dessiné à la place de l'instance graphic.

Remarquons que les interactions réalisées au-dessus des lentilles magiques ne non pas nécessairement consistantes. Dans l'exemple de la figure A.14-a, si l'utilisateur clique au travers de la loupe sur le cercle vert puis tente de le déplacer, le carré rouge se déplacera. En effet, le rendu du cercle a changé mais *pas* ses propriétés ; ainsi, la couleur de fond du cercle étant définie, le cercle est "piquable" *via* ce fond. Pour que l'interaction devienne cohérence, il faudrait changer la propriété "couleur" du graphique, ce qui n'est pas directement possible. Comme pour la loupe, nous fournirons une solution à ce problème à la fin du chapitre 6.

A.8 Présentations multiples

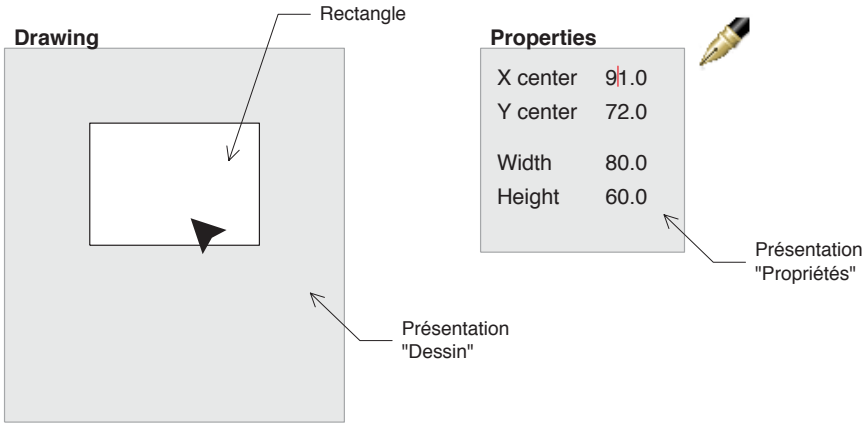


FIG. A.15 – Deux présentations d'un rectangle (composite)

A.8.1 Problème

La plupart des exemples donnés dans ce chapitre sont relatifs aux instruments. Nous donnons ici un aperçu de la facette document de notre modèle en précisant, en particulier, la liaison entre les différentes présentations d'un document et ses données du domaine. Nous reprenons pour ce faire l'exemple déjà mentionné du rectangle avec ses deux présentations, l'une de type "Dessin", l'autre de type "Propriétés" (figure 4.10 on page 111) .

A.8.2 Rectangle composite

La figure A.15 montre la réalisation de l'exemple du rectangle et de ses deux présentations "Dessin" et "Propriétés". Cet exemple illustre la relation entre le document et ses présentations et précise comment le synchronisme entre les présentations est assuré[7].

[7]La gestion des actions concurrentes sur les deux présentations sera abordée dans le chapitre 5.

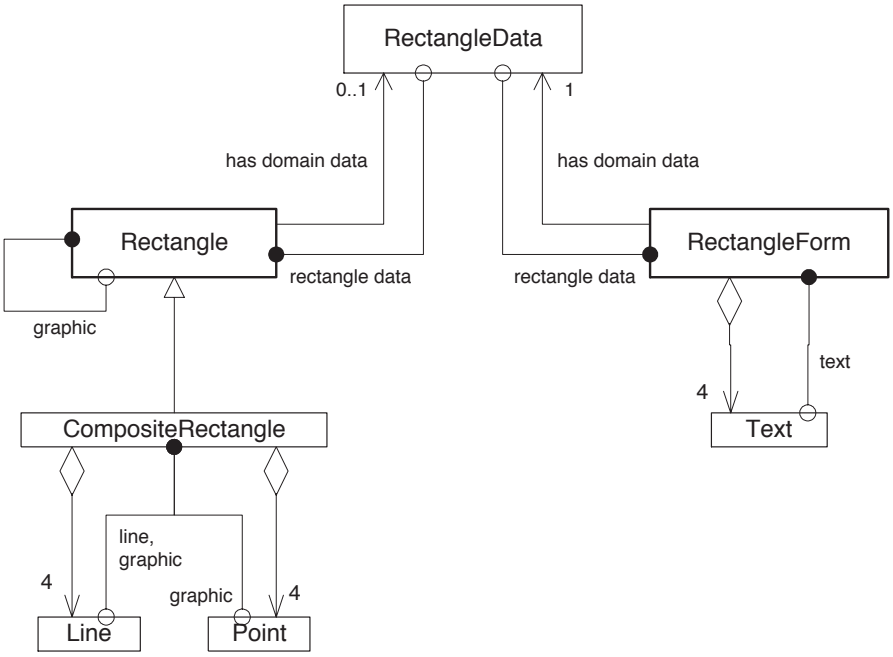


FIG. A.16 – Composants relatifs aux deux présentations d'un rectangle

Les données du domaine du rectangle sont renseignées *via* la classe RectangleData (figure A.16), les deux présentations étant respectivement des instances de Rectangle et de RectangleForm (non représentées). Ces données du domaine jouent le rôle de "pivot" pour la synchronisation des deux présentations au travers des liens d'observation entre Rectangle et RectangleData d'une part, et entre RectangleForm et RectangleData d'autre part. La réalisation de cette exemple suit exactement les spécifications données dans la section 4.1.5.3 on page 111.

Les instances RectangleData, Rectangle et RectangleForm sont ajoutées respectivement à un document, une présentation titrée "Drawing" et une présentation titrée "Properties" (non représentés sur la figure A.16). Le code suivant illustre ce point :

```
Document document = new Document("A nice rectangle") ;
RectangleData data = new RectangleData() ;
document.addChild(data) ;

Presentation presentation1 = new Presentation(document, "Drawing") ;
CompositeRectangle rectangle = new CompositeRectangle(data) ;
presentation1.addChild(rectangle) ;

Presentation presentation2 = new Presentation(document, "Properties") ;
RectangleForm form = new RectangleForm(data) ;
presentation2.addChild(form) ;
```

La création d'un document et de ses présentations est ainsi générique. Le lien entre la présentation et le document se fait à la construction de la présentation. Les données du domaine sont ajoutées en tant que nœuds fils du document, et les représentations (instances de Graphic) sont ajoutées en tant que nœuds fils des présentations. Pour un nouveau format de document, il est nécessaire de définir de nouvelles données du domaine ainsi que des

représentations associées ; par contre, aucune nouvelles classes dérivées de Document et de Presentation ne sont à créer.

La simplicité du code précédent provient du modèle arborescent identique pour les présentations *et* pour les documents. Nous reviendrons à plusieurs reprises sur cet exemple de présentations multiples dans le chapitre suivant.

A.9 Conclusion

Dans cette annexe, nous avons mis en œuvre le modèle DPI dans des contextes variés. Ceci nous a permis d'affiner notre modèle, initialement mis en pratique dans le chapitre 4, ainsi que de confronter ses aspects théoriques présentés dans le chapitre 3 à des situations concrètes.

Le modèle qui en résulte permet d'aborder le problème des espaces de travail mono-utilisateur sous l'angle des documents pour les données et des instruments pour l'interaction. Les affinements du modèle autorisent la conjonction de la composition des graphiques et de la généricité des actions, l'annulation des actions intentionnelles de l'utilisateur et la gestion des présentations multiples. Le modèle de l'interaction instrumentale a affiné la taxonomie des instruments en mettant en avant la notion d'actions composées, d'instruments semi-directs, autonomes, ou imposant un protocole de production, et d'instruments de perception.