Titre: Title:	Un environnement collaboratif de design en immersion
Auteur: Author:	Moncef Naji
Date:	2006
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Naji, M. (2006). Un environnement collaboratif de design en immersion [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/7732/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/7732/
Directeurs de recherche: Advisors:	Benoît Ozell
Programme: Program:	Non spécifié

UNIVERSITÉ DE MONTRÉAL

UN ENVIRONNEMENT COLLABORATIF DE DESIGN EN IMMERSION

MONCEF NAJI
DÉPARTEMENT DE GÉNIE INFORMATIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
AVRIL 2006

Your file Votre référence
ISBN: 978-0-494-17961-1
Our file Notre référence
ISBN: 978-0-494-17961-1

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

UN ENVIRONNEMENT COLLABORATIF DE DESIGN EN IMMERSION

présenté par: NAJI Moncef

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. GRANGER Louis, M.Sc., président

M. OZELL Benoît, Ph.D., membre et directeur de recherche

M. TRÉPANIER Jean-Yves, Ph.D., membre

À mes grand-mères,
des femmes lucides, drôles,
sages et curieuses.

REMERCIEMENTS

Mes premiers remerciements vont, sans contredit, à mon directeur de recherche, Benoît Ozell, qui m'a fait maintes fois confiance, depuis mon PFE jusqu'à ce jour et qui a su manier les différents leviers de ma motivation comme nul autre n'aurait pu. Je lui en suis infiniment reconnaissant et lui dois, à bien des égards, beaucoup plus que ce travail de maîtrise.

Je remercie chaleureusement Jean-Yves Trépanier, d'abord pour l'opportunité qu'il m'a offerte dans MOSAIC mais aussi, bien sûr, pour l'inestimable privilège de son aide financière. Je remercie également Christophe Tribes pour sa disponibilité avenante ainsi que les étudiants de MOSAIC qui m'ont fait l'honneur de participer aux tests usagers.

Enfin, il est une personne que je remercie tout particulièrement, ma conjointe Marie-Claude Pélissier, qui m'a véritablement accompagné, inspiré et soutenu, tout au long de nos deux ans, par son inépuisable générosité et par sa sagesse dévouée.

RÉSUMÉ

Les Environnements Virtuels de Collaboration (EVC) sont appelés à occuper une place grandissante autant dans l'industrie que dans les cercles académiques. La révolution technologique Internet nous apparaîtra bien minime face à la vraie révolution culturelle issue d'une utilisation ubiquitaire du cyberespace. Le défi, énorme, est de bâtir non seulement l'infrastructure mais surtout le *modus operandi* de ces environnements virtuels où nous pouvons créer, manipuler, échanger les artéfacts—maintenant numériques—de nos activités de production.

Nous nous intéressons, dans ce travail de recherche, aux EVC et, plus particulièrement, à leur application à des activités de design dans le domaine de l'aéronautique. Ce travail s'inscrit également dans le cadre du projet MOSAIC en participant à la recherche de solutions intégratrices multidisciplinaires en ingénierie. Nos objectifs de recherche sont alors de formuler une architecture d'Environnement Collaboratif de Design en Immersion (ECDI) qui permette la visualisation scientifique de résultats provenant des autres disciplines de MOSAIC, l'interaction intuitive en immersion avec l'environnement, le contrôle distant de processus informatiques et la participation synchrone de plusieurs intervenants à une session de design collaboratif.

Nous effectuons ainsi une revue de littérature sur la visualisation scientifique et les EVC pour dégager les caractéristiques principales de ces derniers et nous aider à guider les choix architecturaux. Ceux-ci sont atteints après formulation du problème, des contraintes et des besoins fonctionnels exprimés sous forme de scénarios de collaboration.

Nous proposons ainsi une architecture d'environnement immersif présentant une topologie réseau distribuée en pair-à-pair, la visualisation de surface NURBS, pas de gestion de l'attention, une gestion minime de la cohérence, une interactivité simple et intuitive basée sur un langage gestuel principalement fait de glisser-déposer, une persistance du monde virtuel limitée et un support à l'audio-conférence par téléphonie

IP.

Nous détaillons également dans ce document la méthodologie à adopter pour bâtir un prototype de validation puis nous présentons l'évaluation de ce prototype à travers des tests usagers. Nous discutons des résultats de ces tests et vérifions alors l'atteinte de nos objectifs puis nous proposons des avenues futures de recherche pour notre ECDI.

ABSTRACT

Collaborative Virtual Environments (CVEs) are increasingly rising both in the industrial field and the academic field. The technological revolution such as Internet will appear but minimal in the face of the real cultural revolution, emerging from an ubiquitous use of cyberspace. The enormous challenge is not only to build the infrastructure but also, the *modus operandi* of these virtual environments where one can create, manipulate and exchange the artefacts—now digital—of our activities of production.

While being interested in CVEs, the following research is specifically concerned with their application to design activities in aeronautics. The study also contributes to the MOSAIC project, in seeking integrative multidisciplinary solutions. Research objectives involve formulating an architecture for an Immersive and Collaborative Design Environment (ICDE) which would allow scientific visualisation of results coming from other disciplines of MOSAIC, intuitive and immersive interaction with the environment, distant control of computer processes as well as synchrone participation of many individuals in a collaborative design session.

A literature review covering both scientific visualisation and CVEs is elaborated in order to identify their main characteristics and help guide architectural choices. These choices are made following the formulation of the problem, the constraints and the functional needs expressed in the form of collaboration scenarios.

Thus an architecture of an immersive environment is proposed featuring a peer-to-peer distributed network topology, the visualisation of NURBS surfaces, no awareness management, minimal floor control management, simple and intuitive interactivity based on drag-and-drop movements, a limited persistence of the virtual world and IP based audio-conference.

A detailed account of the methodology used to build the validation prototype is presented, followed by the evaluation of the prototype, which is completed through

users' tests. Results from these tests are discussed while underlining the fulfilment of initial objectives. At last, future research goals are proposed for the ICDE.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES NOTATIONS ET DES SYMBOLES

CAVE	CAVE Automatic Virtual Environnement
ECDI	Environnement Collaboratif de Design en Immersion
EVC	Environnement Virtuel de Collaboration
MDO	Multidisciplinary Design Optimisation
OAV	Objet d'Abstraction Visuelle
RV	Réalité Virtuelle
TCAO	Travail Collaboratif Assisté par Ordinateur

LISTE DES ANNEXES

INTRODUCTION

Motivation de la recherche

Les environnements virtuels de télé-collaboration sont appelés à occuper une place grandissante autant dans l'industrie que dans les cercles académiques. La révolution technologique Internet nous apparaîtra bien minime face à la vraie révolution culturelle issue d'une utilisation ubiquitaire du cyberespace. Nous allons vers un « village global » dont la topologie se métamorphose et où les distances ne se mesurent plus en mètres ni même en secondes mais en *bande passante.*

Déjà, les ressources matérielles et humaines distantes géographiquement mais voisines cybernétiquement amènent une nouvelle réalité pour les entreprises. Le défi, énorme, est de bâtir non seulement l'infrastructure mais surtout le *modus operandi* de ces environnements virtuels où nous pouvons créer, manipuler, échanger les artéfacts—maintenant numériques—de nos activités de production.

Le présent travail de recherche s'est ainsi intéressé à étudier la télé-collaboration dans un contexte de design multidisciplinaire, domaine très en vogue dans les industries de haute technologie tant elles connaissent la nécessité d'intégrer des informations complexes et dispersées. Ce travail fait aussi partie d'un plus grand effort de recherche, le projet MOSAIC, qui œuvre de façon plus large dans cette direction.

Contexte de la recherche

Le projet MOSAIC[1] est un projet de recherche s'inscrivant dans le cadre du consortium de recherche et d'innovation aérospatiale au Québec (CRIAQ) en partenariat institutionnel[2] et industriel[3] et qui vise le développement d'environnements de conception et de design optimal multidisciplinaire (MDO).

Les industries impliquées dans ce projet ont en commun des processus de design complexes faisant interagir différentes disciplines spécialisées, intimement couplées les unes aux autres. Traditionnellement, les cycles de design impliquent un va-et-vient constant entre les disciplines avec un flux d'information en cascade entre celles-ci. Ce processus linéaire trouve sa limite à mesure que le produit se complexifie et que les contraintes de design se durcissent. Les modifications aux variables de design apportées par une discipline affectent d'autres disciplines et beaucoup de ressources sont consacrées à corriger ces interférences. La MDO cherche à résoudre ce problème en fournissant les outils pour déployer une synergie entre les disciplines impliquées lors d'un design.

Ainsi, le projet MOSAIC vise à fournir aux participants industriels des technologies d'intégration et explorer les possibilités de l'optimisation multidisciplinaire dans le design de systèmes aérospatiaux. Le projet comporte une douzaine de tâches qui peuvent se répartir en trois catégories : (1) Le développement de protocoles d'échange de données. (2) Le développement de logiciels d'analyse et d'optimisation adaptés à la MDO. (3) Le développement de solutions intégratrices de ces logiciels d'analyse et d'optimisation adaptées aux besoins des participants industriels. Le présent travail s'inscrit dans le projet MOSAIC en participant à une tâche intégratrice qui consiste à

[1] systèMe d'Optimisation basé sur des Standards et des Analyses et permettant une Intégration Configurable

[2] Université Concordia ; École polytechnique de Montréal ; École de technologie supérieure

[3] Bombardier Aéronautique ; Pratt & Whitney Canada ; Bell Helicopter Textron

créer un environnement immersif où plusieurs collaborateurs interagissent—au moyen des protocoles d'échange de données—avec des logiciels d'analyse et d'optimisation pour guider leurs décisions de design.

Problème à résoudre

Le mandat qu'octroie cette tâche au sein de MOSAIC offre une passionante opportunité de recherche que le présent travail se propose d'explorer. Le principal objectif de cette recherche est ainsi de proposer une architecture autorisant plusieurs participants à collaborer en temps réel dans un environnement immersif virtuel. Plus spécifiquement, cet environnement doit permettre la visualisation scientifique de résultats provenant des autres disciplines, l'interaction intuitive en immersion avec l'environnement, le contrôle distant de processus informatiques et la participation synchrone de plusieurs intervenants à une session de design collaboratif.

Aperçu du mémoire

Nous explorerons d'abord (chapitre 1) les domaines de la visualisation scientifique et des Environnements Virtuels de Collaboration au cours d'une revue bibliographique, laquelle sera suivie d'une synthèse des caractéristiques que l'on retrouve dans de tels environnements. Nous passerons ensuite en revue (chapitre 2) les besoins recueillis dans le cadre de MOSAIC pour notre environnement avant de proposer des scénarios de collaboration et les éléments (identifiés dans la revue de littérature) que nous retiendrons pour bâtir un prototype de l'Environnement Collaboratif de Design en Immersion (ECDI). Nous exposerons alors en détail (chapitre 3) la mise en place de ces caractéristiques par une description de l'architecture de l'ECDI. S'en suivra une discussion (chapitre 4) qui, s'appuyant sur des tests usagers, tentera d'évaluer les

apports du prototype sur l'étude de la télé-collaboration. Nous regarderons enfin les directions futures de ce travail de recherche avant de conclure.

CHAPITRE 1

VISUALISATION ET ENVIRONNEMENTS VIRTUELS DE COLLABORATION

Cette revue de littérature cherche à faire un tour d'horizon des connaissances, techniques et outils reliés aux environnements virtuels de collaboration. Ce travail s'inscrit dans le cadre du projet MOSAIC pour l'ECDI que nous cherchons à définir et dont le domaine d'application est celui du design en aéronautique.

Un des objectifs de cet environnement étant de visualiser des modèles géométriques et des données scientifiques, nous débuterons ce document par un aperçu du domaine de la visualisation scientifique. Par la suite, nous aborderons le cœur du sujet en examinant les différents paradigmes de collaboration afin de dégager des caractéristiques de comparaison. Ces dernières seront utilisées pour passer en revue les environnements de collaboration existants les plus pertinents pour notre travail, autant ceux issus de la recherche académique que ceux développés commercialement.

1.1 Visualisation scientifique

La raison d'être du calcul assisté par ordinateur, disait Richard Hamming, c'est la compréhension et non les nombres. La visualisation est un outil pour atteindre une compréhension intuitive des nombres. Rogowitz (Marchak et al. 1993) écrit que la visualisation est le processus associant des valeurs numériques à des dimensions perceptuelles. Le système visuel humain est capable de décoder de l'information sur plusieurs dimensions, entre autre sur les plans spatial, chromatique et temporel. Concevoir un système de visualisation en tenant compte des forces et faiblesses de l'appareillage hu-

main permet d'exploiter les capacités du système visuel à reconnaître les structures et formes et permet de déjouer les limitations de mémoire et de concentration.

La visualisation sert dans plusieurs contextes scientifiques et techniques bien précis. Bergeron (Butler et al. 1993) divise les objectifs de la visualisation en trois catégories : *visualisation descriptive*, *visualisation analytique* et *visualisation exploratoire*. La visualisation descriptive est utilisée lorsque le phénomène représenté dans les données est connu mais que l'usager désire effectuer une vérification visuelle claire de ce phénomène et, éventuellement, l'exposer à d'autres personnes. La visualisation analytique (recherche dirigée) est le processus que l'on suit pour étudier la présence d'un phénomène connu dans les données. La visualisation exploratoire (recherche non dirigée) est nécessaire lorsque l'on ne sait pas d'avance quoi chercher dans les données ; la visualisation pouvant alors aider à comprendre la nature des données, par exemple en dégageant des motifs. Un système versatile de visualisation doit offrir ces trois catégories d'activités qui sont complémentaires dans tout processus d'analyse ou de conception.

1.1.1 Modèle de flux de données

Un modèle très utilisé pour décrire le processus de visualisation est celui du flux de données, dont la paternité est généralement attribuée à Haber et McNabb (1990). Ce modèle, représenté à la figure 1.1, décrit le processus de visualisation comme une séquence de transformations qu'on peut découper en trois étapes : une première étape d'enrichissement des données ou de filtrage donne un ensemble dérivé de données ; lors de la deuxième étape on projette ces données dérivées de façon isomorphique vers un espace d'abstractions visuelles, en générant des Objets d'Abstraction Visuelle (OAV) ; la dernière étape permet d'effectuer le rendu des OAV pour en faire des images ou séquences d'images affichables sur une sortie graphique de l'ordinateur.

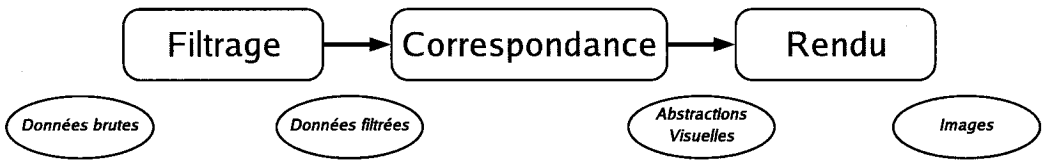


FIGURE 1.1 Modèle de flux de données

Le modèle de flux de donnée se prête bien à la conception d'environnements de visualisation dits à programmation graphique, aussi appelés Environnements de Visualisation Modulaires (EVM). Ces EVM—tels que *AVS* (Upson et al. 1989), *IRIS Explorer* (Foulser 1995) ou *IBM Visualization Data Explorer* (Abram et Treinish 1995)—offrent à l'usager la possibilité de construire graphiquement le pipeline de visualisation en spécifiant comment chacune des trois étapes (filtrage, correspondance, rendu) doit être exécutée. L'avantage de tels systèmes est qu'il permet à l'usager de bâtir sa propre application de visualisation en fonction de ses besoins sans avoir recours à la programmation en langage informatique. Cependant, certains auteurs (Hibbard 2000) ne sont pas convaincus qu'un tel modèle se prête bien à des visualisations complexes, argumentant plutôt en faveur de « bonnes » librairies de visualisation.

1.1.2 Modèle d'état des données

Le modèle de flux de données vu précédemment peut être décrit à l'aide d'un graphe dont les nœuds sont les transformations appliquées aux données et dont les arcs sont le flux qu'empruntent ces données. Le modèle d'état des données (Chi et Riedl 1998) renverse en quelque sorte ce graphe pour assigner aux nœuds un état précis des données et aux arcs les opérateurs de transformation faisant passer les données d'un état vers un autre. Les données s'expriment, selon ce modèle, par quatre états successifs : données brutes, abstraction analytique, abstraction de visualisation et vue. L'auteur argumente que cette formulation fait ressortir explicitement le type des données (leur état) mais surtout qu'il propose un modèle centré sur les opérateurs

permettant ainsi la réutilisation des opérateurs et un découplage entre vue et données. L'auteur assure que ce modèle est aussi expressif (Chi 2002) que le modèle de flux de données et propose une taxonomie (Chi 2000) des programmes de visualisation selon son modèle. Bien qu'il offre une plus grande flexibilité quant aux types d'opérateurs applicables sur les données, ce modèle n'apporte pas d'atout majeur à la visualisation scientifique où le problème n'est pas d'offrir la gamme la plus large d'opérateurs de transformations mais de construire une représentation interactive pertinente selon la tâche de visualisation, le type et la structure des données.

1.1.3 Modèle communicationnel/linguistique

Le modèle dit « communicationnel/linguistique » (Ozell et Camarero 1995) conçoit l'acte de visualisation scientifique comme un acte de communication et en décrit l'articulation par analogie avec la linguistique structurelle. Chaque acte de visualisation s'établit sur deux axes : *syntagmatique* et *paradigmatique*. Le paradigme est le lexique du domaine, c'est-à-dire l'ensemble des signes du langage de visualisation se définissant les uns par rapports aux autres et constituant ainsi les différents concepts que l'on cherche à communiquer. Le syntagme est la combinaison de ces concepts—selon une certaine grammaire précisant les configurations valides—pour former des « phrases » visuelles, des images à l'écran. Ce modèle prend nativement en compte la nature discoursive de la visualisation, ce qui, dans le contexte d'un environnement de collaboration, est primordial : tout processus de collaboration consiste en un échange continu d'informations (communication) et une coordination des actions.

Ce modèle linguistique a été mis en pratique dans un environnement de visualisation développé par Ozell et Camarero (1996). Cet environnement (Pic et Ozell 2000), nommé *VU*, offre une grande facilité de configuration à travers un langage décrivant le processus de communication visuelle désiré par l'usager. Dans le cadre de MOSAIC une

approche similaire serait souhaitable pour l'aspect « visualisation » de l'environnement de collaboration. En effet, la tâche no 1 au sein de MOSAIC prévoit développer un protocole d'échange de données. Ce travail, nécessitant la construction d'une ontologie du domaine, fournira le découpage sémantique des concepts visualisables, lesquels sont directement applicables dans le modèle communicationnel/linguistique.

1.1.4 Vers la collaboration

L'activité de visualisation est, plus souvent qu'autrement, de nature collaborative. Les données toujours plus complexes à analyser nécessitent la participation d'expertises variées dont les détenteurs sont géographiquement dispersés. Cependant, la plupart des environnements de visualisaton ne se prêtent qu'à une utilisation mono-utilisateur ; au mieux, des extensions collaboratives leur sont ajoutées mais sans offrir un environnement intégré de collaboration. Une classification des activités de collaboration assistées par ordinateurs, très présente dans la littérature, s'énonce selon une matrice temps-lieu (Applegate 1991) (voir tableau 1.1). Brodlie et al. (1998) classifie aussi les systèmes de visualisation selon deux axes : le nombre d'humains participants et le nombre de machines (voir tableau 1.2). Nous nous intéresserons à la configuration (m,n) aux configurations « même temps–lieu différent » et « temps différent–lieu différent » comme base de définition fonctionnelle pour notre Environnement Virtuel de Collaboration (EVC).

Les prochaines sections vont donc traiter de ces environnements en dégageant les caractéristiques du Travail Coopératif Assisté par Ordinateur (TCAO) et de la réalité virtuelle immersive avant de présenter une revue des environnements existants.

TABLEAU 1.1 Modèle temps–lieu en TCAO

Lieu		Même	Différent
	Différent	Téléconférence	Courriel
	Même	Réunions	Quarts de travail
		Temps	

TABLEAU 1.2 Classification de systèmes de visualisation en combinaisons humains-machines

		Machines	
		1	m
Humains	1	Visualisation « classique »	Visualisation distribuée
	n	Visualisation coopérative	Visualisation distribuée et coopérative

1.2 Environnements virtuels de collaboration

1.2.1 Définition

L'activité de design est une activité à dominante fortement cognitive. Des domaines d'ingénierie tels que l'automobile ou l'aéronautique produisent des systèmes et des machines dont la complexité du design ne va qu'en accroissant. De plus en plus d'informations doivent être prises en compte, de plus en plus de contraintes sont à respecter. Les activités de design de ces industries font appel à la collaboration de disciplines toujours plus spécialisées, autant dans les connaissances que dans les pratiques. Les EVC s'imposent de plus en plus, offrant un lieu virtuel intégré de

travail pour manipuler collaborativement les artefacts du processus de production et les outils servant à créer, à modifier et à partager ces derniers.

Les EVC découlent de la recherche sur le TCAO où l'on cherche à définir des paradigmes et des technologies afin d'offrir aux utilisateurs l'intégration de leurs outils informatiques dans le but d'une meilleure synergie collaborative. Typiquement, les systèmes de TCAO présentent des canaux de communication audio et vidéo entre les participants d'une session coopérative ainsi que des « collecticiels » tels que des éditeurs de textes partagés ou des tableaux blancs partagés (Brodlie et al. 1998). Cependant, les EVC peuvent se concevoir comme étant le résultat d'une convergence des intérêts de recherche du TCAO et de la Réalité Virtuelle (RV) (Benford et al. 2001). Au lieu d'offrir aux utilisateurs des canaux et médias parallèles de communication, les EVC promeuvent la création d'un espace cybernétique intégré ; paradigme idoine à l'interface immersive que nous nous proposons ici de réaliser.

1.2.2 Caractéristiques

Au vu des EVC étudiés (et présentés plus loin à la section 1.3) il est possible de dégager plusieurs caractéristiques qui définissent les EVC :

Monde partagé. Un EVC présente le plus souvent un monde virtuel peuplé d'objets. Pour permettre la collaboration il faut que tous les usagers voient le même monde virtuel et donc qu'il y ait des mécanismes de partage des objets.

Présence et co-présence. Ces termes visent à qualifier le sentiment qu'ont les collaborateurs, respectivement, d'être transportés dans un autre lieu et de partager ce lieu avec les autres collaborateurs. Cela est généralement implémenté avec l'usage d'avatars et la reproduction d'une partie du langage non verbal des usagers.

Audio- et Vidéo-conférence. Ce sont des mécanismes qui permettent aux colla-

borateurs de se parler et éventuellement de se voir.

Persistance du monde virtuel. La persistance qualifie l'autonomie qu'a le monde virtuel par rapport aux sessions d'utilisation des usagers. À un bout du spectre, les mondes virtuels évolutifs ont une vie qui leur est propre et sont généralement implémentés avec un serveur qui continue de faire évoluer le monde en l'absence même d'usagers (par exemple pour des calculs numériques très longs ou pour les jeux vidéo en ligne massivement multi-joueurs). À l'autre bout du spectre, l'environnement peut ne présenter aucune persistance au delà de la session d'utilisation. Il y a évidemment des niveaux intermédiaires de persistance, par exemple une persistance statique avec la possibilité de sauvegarder l'état courant du monde virtuel pour une reprise ultérieure de la session.

Synchronisme. Cette caractéristique indique quelles combinaisons de temps de collaboration (immédiate ou différée) sont possibles (*cf* tableau 1.1).

Conversations/manipulations privées et publiques. Certains EVC font la différence entre conversations privées et publiques. Cela est une façon d'exprimer que le monde virtuel offre à la fois des objets partagés et des objets locaux (qui n'apparaissent et ne peuvent être modifiés que chez l'un des collaborateurs). Cette caractéristique qualifie aussi la possibilité de créer des sous-mondes privés, restreints à certains usagers du monde virtuel (un peu comme les chambres privées dans les environnements de clavardage tels qu'IRC).

Gestion de la cohérence. On entend par ces termes les mécanismes mis en place autour des objets partagés pour s'assurer qu'ils sont correctement répliqués d'un site de collaboration à l'autre ainsi que les mécanismes de verrouillage des objets pour empêcher que les usagers tentent de se saisir des mêmes objets en même temps.

Gestion de l'attention. Cette caractéristique, surtout présente dans les EVC accueillant un nombre important d'usagers simultanément, décrit que des mécanismes sont utilisés pour filtrer les données à répliquer à travers le réseau et

qu'on applique une distribution sélective des données, basée sur des critères de visibilité, de pertinence, de niveau de détail...

Interopérabilité. Ceci indique que l'EVC peut aussi bien être utilisé en environnement immersif comme non-immersif ; cela soulève généralement des questions au niveau de l'interface : faut-il créer deux interfaces différentes ou chercher des paradigmes compatibles avec les deux ?

1.2.3 Communication réseautique

Les EVC sont des applications très orientées-réseau et l'architecture des communications réseau a un grand impact sur la performance ainsi que sur les types d'environnements possibles. Il est possible de diviser l'architecture réseau des EVC selon trois plans : les types des données à transmettre, les types de topologies des canaux de communication et les types des flux qu'empruntent les données sur ces canaux.

Types de données

On divise le plus souvent les données selon leur taille et leur fonction :

Petite taille – événements. Les données de petite taille correspondent à des données transmises soit de façon non fiable comme les données des dispositifs de repérage soit de façon fiable comme les données indiquant des changements d'état du monde. Ces données requièrent le plus souvent un traitement prioritaire avec une faible latence de réseau.

Moyenne taille – atomique. Ce sont des données de taille assez petite pour être résidentes en mémoire physique du client local et qui doivent être traitées de façon atomique comme, par exemple, la géométrie 3D d'un objet de la scène.

Grande taille – segmenté. Ce sont des données de taille trop grande pour tenir

en entier en mémoire qu'on ne peut donc accéder que de façon segmentée. Les données scientifiques de très grande taille et des flux vidéo font partie de cette catégorie.

Type de Topologie

La topologie de l'EVC dépend grandement de l'usage que l'on compte en faire. Voici les catégories de topologies que l'on rencontre le plus souvent :

Répliquée – homogène. Cette topologie est caractéristique des applications de réalité virtuelle militaires où chaque client détient une base de données complètement répliquée du monde virtuel et où le partage se fait par envoi en multidiffusion de messages à tous les participants (généralement sur des canaux sécuritaires). Sans système de contrôle centralisé, un nouveau client, à son arrivée, doit attendre de recevoir l'état du monde virtuel de la part des autres clients.

Centralisée – partagée. Dans cette approche, les données partagées sont toutes conservées dans un serveur central simplifiant la gestion de multiple clients dans le cas où une gestion stricte de la cohérence est nécessaire. Cependant la disponibilité du système est tributaire de la stabilité du serveur et la latence du système peut être affectée par le fait que chaque message doit faire le chemin en double.

Distribuée – poste-à-poste. Cette approche simule une structure de mémoire partagée sur le réseau virtuel créé par l'interconnexion en poste-à-poste des clients entre eux. Les objets créés chez un client sont automatiquement répliqués chez les autres. Ce paradigme simplifie le développement de l'EVC au prix de la performance réseau car chaque client doit établir une connexion avec tous les autres ce qui rend cette topologie peu extensible à mesure que le nombre de clients augmente. Mais pour un nombre restreint d'usagers simultanés cette approche est

un bon compromis entre la performance et l'effort de développement.

Centralisée avec regroupements. Cette topologie répartit la base de données du monde virtuel sur plusieurs serveurs interconnectés et chaque client choisit son serveur selon des critères de proximité ou d'intérêt à un sous-monde de l'environnement général. On retrouve cette topologie notamment dans les jeux vidéo massivement multi-joueurs.

Type de Diffusion

Les topologies vues précédemment appellent à différentes stratégies de diffusion, parfois aussi à une combinaison de celles-ci :

Diffusion individuelle (*unicast*). C'est le type de diffusion le plus portable est qui est adapté à une interconnexion en poste-à-poste.

Diffusion générale (*broadcast*). Dans ce schéma de diffusion, les clients envoient leurs messages sur le sous-réseau à l'intention de tous les autres. L'avantage est la simplicité mais ces messages ne traversent pas le sous-réseau, à moins que les routeurs ne le fassent explicitement.

Multidiffusion (*multicast*). Ce type de diffusion est très efficace et utilise de façon optimale la bande passante mais requiert que le matériel réseau la supporte spécifiquement (IPv6 par exemple).

1.2.4 Immersion

Les EVC peuvent présenter une interface immersive. Le principal avantage de l'immersion est que l'usager est réellement *dans* le monde virtuel et qu'il peut manipuler directement les objets plutôt que leur représentation. Dans un paradigme classique, le monde virtuel (qui est tridimensionnel) est projeté sur un écran en 2D en perdant,

le plus souvent, une partie de l'information de profondeur.

En immersion, l'usager voit la scène comme s'il y était et peut mieux apprécier les proportions et la perspective. Cela ouvre aussi la porte à la réflexion sur de nouveaux paradigmes d'interaction avec les mondes virtuels. Les environnements immersifs nécessitent cependant matériels et logiciels adaptés :

Affichage. Le premier critère est la stéréoscopie. Il existe essentiellement deux moyens (courants) de l'obtenir : (1) par projection des deux images sur la même surface puis par dé-multiplexage temporel ou chromatique avec des lunettes ou (2) à l'aide d'un visiocasque présentant un petit écran en face de chaque œil. Pour les environnements à projection on rencontre le plus souvent des configurations murales (Immersadesk) ou des configurations cubiques où chaque face (ou presque) correspond à un écran (Cruz-Neira et al. 1993).

Périphériques d'entrée. L'immersion permet de manipuler en 3D le monde virtuel. Ainsi les périphériques offrent cette capacité. Les bras de repérage (*wand*) renvoient une position et une orientation dans l'espace sur les trois axes. Les gants de réalité virtuelle ajoutent en plus les angles de rotation des phalanges permettant (moyennant une interface assez complexe) d'utiliser un langage gestuel.

Paradigmes d'interaction. L'interface est bien évidemment très importante et la recherche est très active à ce sujet. Notons, qu'en plus des périphériques de repérage, on rencontre aussi la reconnaissance vocale (Ozell et al. 2000) et l'utilisation de dispositifs portables (comme des ordinateurs de poche) pour des manipulations plus fines ou pour explorer un autre espace parallèle au monde virtuel (par exemple un méta-espace comme la description de la scène).

1.2.5 Facteurs humains

Latence du réseau

Pour la téléconférence, une latence supérieure à 200 ms dégrade généralement qualité de la conversation. Pour la manipulation collaborative d'objets partagés, on fixe le seuil acceptable autour de 100 ms à 200 ms, seuil qui change selon l'expérience des usagers et la précision requise des manipulations. Une piste de solution pour y pallier est de représenter graphiquement la latence (Gutwin et al. 2004).

L'environnement de collaboration : Espace ou Lieu

Des études provenant autant du domaine de l'architecture que de celui du TCAO cherchent à définir et à tirer profit de la dichotomie entre les concepts d'*espace* et de *lieu*. La plupart des EVC reposent sur un paradigme d'espace virtuel pour gérer les interactions distantes entre les utilisateurs. En articulant le design de ces EVC autour de notions d'espace imitant l'organisation spatiale du monde réel, on s'attend à ce que les utilisateurs agissent et interagissent selon des patrons de comportements largement analogues à leur quotidien dans le monde physique. En d'autres termes, un système spatialement organisé supportera des comportements spatialement inspirés (Harrison et Dourish 1996).

Un espace est une région plus ou moins bornée, un sous-ensemble de l'espace tridimensionnel. Dans un espace, on ne peut qu'*être* tandis qu'un lieu se définit d'abord par ce que l'on y *fait*. C'est le lieu, et non l'espace, qui fixe le contexte du comportement. Harrison et Dourish font la différence entre espace et lieu de façon plus précise. Le lieu naît de la tension entre la *connectivité* et la *distinction* d'un espace, plutôt que par sa seule structure tridimensionnelle. Par connectivité, on entend le degré de

cohérence ou de continuité d'un espace par rapport à son voisinage. Par distinction, on entend le degré de son originalité ; combien il se démarque de son voisinage.

C'est dans la résolution de ce paradoxe—un espace différent et unique mais qui n'est pas non plus hors contexte—qu'émerge le lieu. Cette définition, bien qu'un peu abstraite, peut nous guider dans le design d'un EVC. Harrison et Dourish nous mettent cependant en garde : la relation entre espace et lieu est de nature sociale, et non technologique. En tant que concepteur d'un EVC, on ne peut construire le lieu mais uniquement équiper les utilisateurs de moyens de communication et d'action afin qu'ils peuplent eux-mêmes leur espace et qu'ils en créent un lieu.

1.3 Revue des environnements de collaboration existants

1.3.1 COVISA

Le projet COVISA (Collaborative Visualization & Scientific Analysis) est, à l'origine, une extension collaborative à l'Environnement de Visualisation Modulaire IRIS Explorer 3.0 initiée par Wood et al. (1995b) maintenant intégrée à Iris Explorer depuis la version 5. Le principe central de l'approche de Wood est d'étendre le modèle de pipeline de visualisation de Haber et Mc Nabb en ajoutant à chacune des trois étapes la possibilité de partager les données et le contrôle sur l'opérateur (filtre, correspondance, rendu). Ce paradigme permet plusieurs scénarios de collaboration selon la répartition du contrôle et le flux des données entre les participants.

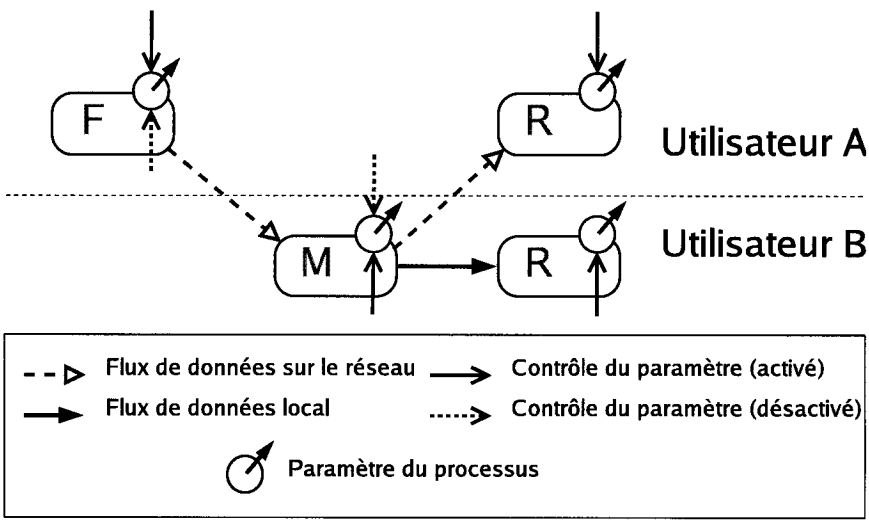


FIGURE 1.2 Modèle de flux de données augmenté pour la collaboration

La figure 1.2 présente un exemple d'un graphique de flux de données augmenté pour un scénario possible (Wood et al. 1995a). Dans ce scénario, l'utilisateur A accueille l'opérateur de filtrage sur sa machine tandis que l'utilisateur B se charge de l'opérateur de correspondance. Les données issues de la première étape (données filtrées) sont envoyées par réseau à l'utilisateur B. La sortie de cette étape (abstractions graphiques)

est, d'une part, envoyée par réseau à l'utilisateur A et, d'autre part, injectée dans l'étape suivante du pipeline local, soit le rendu. Les deux utilisateurs sont alors autonomes quant au contrôle du rendu. Ils peuvent cependant s'échanger ici le contrôle des paramètres des deux premières étapes : par exemple, bien que l'opération de filtrage s'effectue sur la machine de l'utilisateur A, l'utilisateur B peut néanmoins demander le contrôle de ces paramètres ; il y a alors un mécanisme de verrouillage par jeton pour assurer la cohérence des modifications entre les usagers.

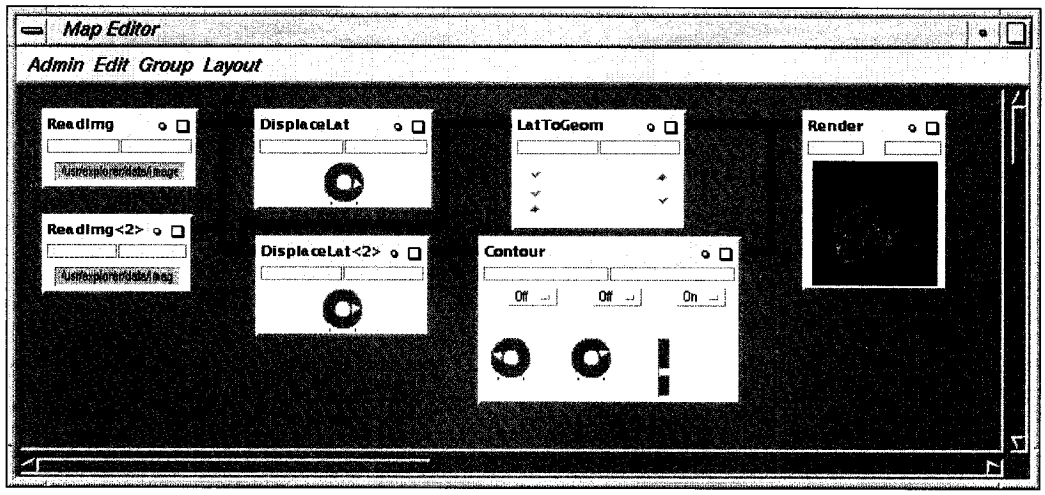


FIGURE 1.3 Éditeur de graphe de flux de données – IRIS Explorer 5.0

Interface et interactions

COVISA étant une extension de IRIS Explorer, l'interface est celle d'un paradigme de programmation graphique. À l'aide de la souris, les usagers construisent graphiquement leur pipeline de visualisation en choisissant les modules appropriés à leur tâche (voir figure 1.3). Des modules spéciaux pour la collaboration (dits *Collaboratively Aware Modules*) sont fournis pour chacun des types de données de base. En choisissant ces CAMs, les collaborateurs définissent le partage des données et des contrôles. D'autres modules spéciaux, les *Advisor Modules* permettent à l'un des collaborateur de jouer le rôle de tuteur en manipulant le pipeline d'un autre usager distant. Au-

cune communication audio ou vidéo n'est intégrée dans le système, les usagers doivent recourir à des moyens externes pour cette fin.

Architecture et topologie

Le point de départ de l'architecture est celui d'un EVM. L'environnement fournit un ensemble de modules et des types de données prédéfinis mais reste extensible en permettant l'ajout de modules personnalisés. Un langage de script permet d'automatiser la création et la manipulation du graphe de flux de données. Un serveur central gère la liste des participants, des contrôles et des flux de données. Un module spécial, appelé « serveur local » doit être instancié par les collaborateurs en lui fournissant les paramètres du serveur central COVISA. Les CAMs peuvent alors directement communiquer avec le serveur central qui demande la création (par des commandes en langage script) des mêmes CAMs chez les clients des autres collaborateurs. Ces modules apparaissent dans leurs éditeurs et ils peuvent les mettre à leur guise dans leur pipeline.

1.3.2 CSpray

Collaborative Spray (*CSpray*) est une extension collaborative de l'environnement de visualisation *Spray*, développé par Pang (1994) à l'université de Californie, Santa Cruz. *Spray* utilise un paradigme de visualisation original qui le démarque des systèmes basés sur le flux de données. Dans ce système, les données sont visualisées en les « aspergeant » avec un aérosol à particules appelées *Smart Particules* (*sparts*). Ces *sparts*, conceptuellement, se situent dans une certaine mesure à la croisée des systèmes de particules et de l'animation comportementale (comme les simulations de vols d'oiseaux, bancs de poissons, etc). Elles peuvent être programmées pour réagir spécifiquement aux données qu'elles rencontrent. Le rendu des données s'apparente alors à une activité de peinture : à l'aide d'un aérosol, on choisit quelles *sparts* envoyer, leur débit, leur distribution, etc. pour révéler un certain aspect des données. Par exemple, en utilisant des *sparts* qui opèrent sur un champ scalaire et ne retiennent que les valeurs atteignant un certain seuil, on peut obtenir une isosurface. La figure 1.4 illustre une session de visualisation avec *Spray*. Les *sparts*, encapsulant différents algorithmes de visualisation, génèrent, des Objets d'Abstraction Visuelle (OAV) (tels que des courbes, surfaces, glyphes...), lesquels sont transformables par la suite en images.

En ce qui concerne l'aspect collaboratif, *CSpray* part de *Spray* et permet à plusieurs usagers de partager leurs données, leurs aérosols et les OAV qui en découlent. Ils peuvent, à l'instar du modèle de flux de données augmenté dans COVISA, échanger les données à différents niveaux du pipeline, soit les données brutes, les OAV et les images finales. Les auteurs (Pang et Wittenbrink 1995) privilégient cependant le partage des OAV pour plus de privauté des données, une gestion plus efficace de la bande passante et une manipulation libre des résultats pour tous les collaborateurs.

Interface et interactions

L'interface de *CSpray* est celle d'un environnement à fenêtres et se contrôle avec la souris et le clavier. Une fenêtre principale affiche le monde virtuel alors que des barres d'outils autour permettent de sélectionner et paramétrer les sources de données, les aérosols et les *sparts*. Une autre fenêtre, la fenêtre publique, donne un autre point de vue de la scène, par exemple, celui d'un collaborateur (voir la figure 1.5). Les autres collaborateurs sont représentés par des avatars en forme d'œil, nommés *eyecons*, indiquant la position, l'orientation et la direction du regard des protagonistes ainsi que leur nom en surimpression. Il n'y a pas de moyen audio ou vidéo de communication mais les collaborateurs peuvent annoter les objets et une synthèse vocale peut lire ces notes.

Les usagers ont le contrôle sur le partage des aérosols qu'ils possèdent (ceux qu'ils ont créés) et peuvent ainsi travailler en privé sur une visualisation avant de choisir de la partager. Afin de garantir la consistance du monde partagé, une gestion du contrôle des objets (*Floor Control*) est implantée. Les objets partagés (essentiellement les aérosols publics) peuvent être dans un des quatre états de disponibilité (d'un point de vue local) : libre, possédé (par l'usager local), pris (par un usager distant) et en demande. Le système utilise un code de couleur pour renseigner les utilisateurs de l'état des objets : vert pour les objets possédés, rouge pour les objets pris et jaune pour les objets en demande. Si un aérosol est possédé ou pris et qu'un autre usager tente de le contrôler, celui-ci devient en demande. L'usager ayant le contrôle de l'objet en est averti et peut explicitement relâcher son contrôle. Après un certain délai d'inactivité un objet sous contrôle redevient libre. Ce système permet de s'assurer qu'un objet partagé n'est modifiable que par une personne à la fois. Cependant, il nécessite l'envoi de plusieurs messages de contrôle pour gérer les états, ce qui peut nuire à l'interactivité avec le monde virtuel.

Architecture et topologie

La couche communication de CSpray utilise une architecture basée sur des flux (*streams-based*). Il s'agit d'une abstraction d'une communication orientée connexion implémentables dans un UNIX System V comme des *Streams* ou des interfaces de connexion (*sockets*) TCP/IP. Ces flux se déclinent en flux de priorité, d'information et de contrôle. Différents flux sont assignés à différentes tâches permettant ainsi différentes qualités de services des informations à véhiculer. Les événements sont gérés en passant des messages entre les hôtes collaborant à distance à travers des connexions établies sur des flux. Les événements peuvent être locaux (et ne sont pas transmis) ou collaboratifs. Ces derniers se déclinent selon qu'ils nécessitent une gestion de la cohérence ou non.

Dans cette architecture distribuée, un module interne à chaque instance de l'application se charge de la gestion de tous les messages avec les autres hôtes collaboratifs. Ce module reçoit aussi les messages en provenance des collaborateurs et les relaie localement aux autres modules de *CSpray*. Il s'agit ici d'une topologie répliquée ; un hôte joignant une session de collaboration doit se connecter à l'un des collaborateurs auprès duquel il s'enregistre, reçoit de sa part les objets en partage puis la liste des autres collaborateurs auprès desquels il répète la procédure d'enregistrement. Chaque événement devant être partagé génère un message à envoyer à tous les collaborateurs. Les auteurs (Pang et Wittenbrink 1997) reconnaissent qu'une telle architecture peut s'avérer difficilement extensible mais suggèrent qu'avec des réseaux à grande vitesse, une implantation efficiente et l'usage de la multidiffusion (*multicasting*) cette limitation reste maîtrisable.

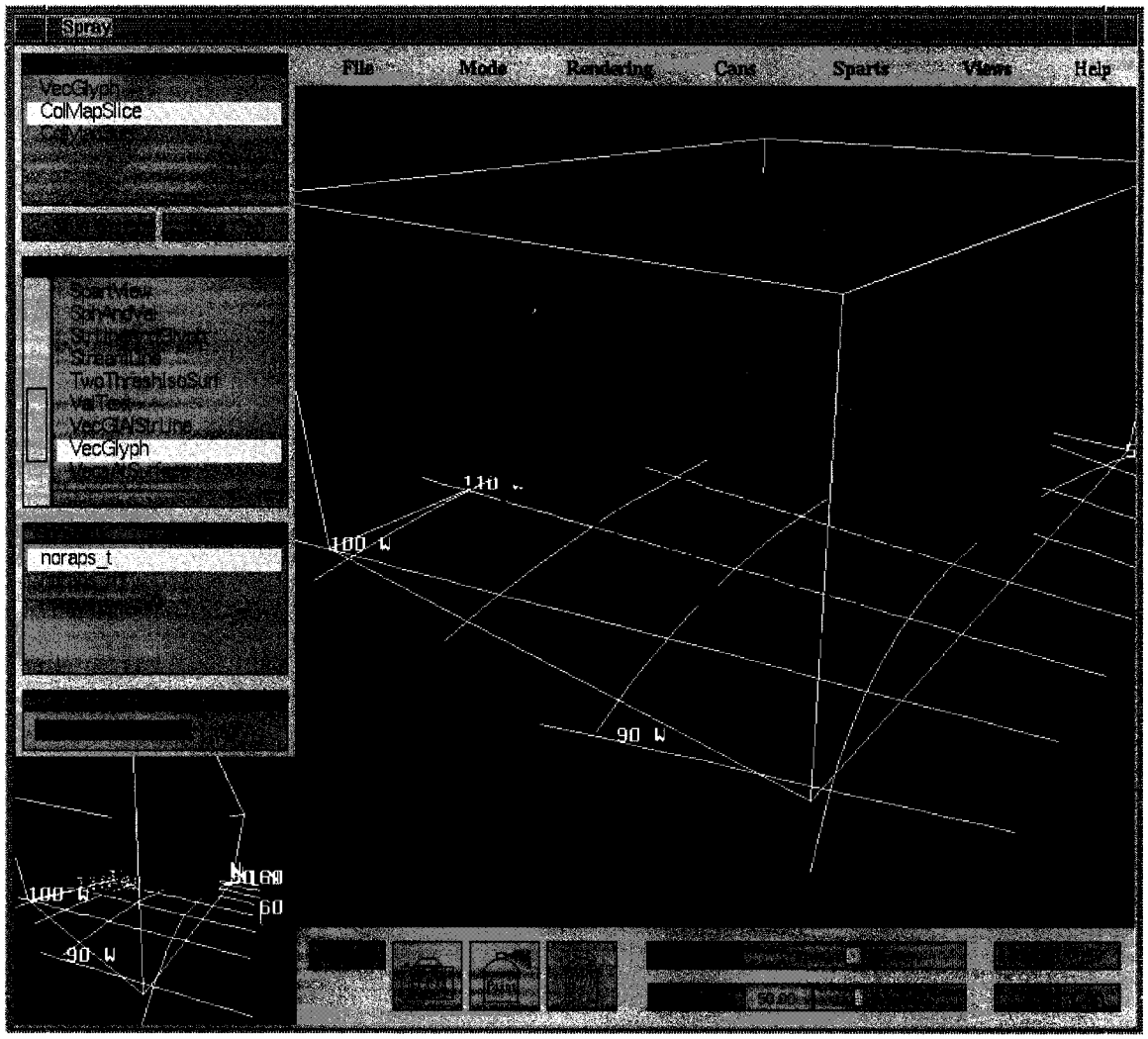


FIGURE 1.4 Interface de *Spray* (tiré de Pang et Wittenbrink (1995)).
Différents aérosols et *sparts* sont utilisés pour visualiser une isosurface de température avec les valeurs d'humidité, un champ de vecteur représentant la vitesse et la direction du vent et une coupe du champ de température. La fenêtre réduite du coin inférieur gauche montre le point de vue de l'aérosol courant.

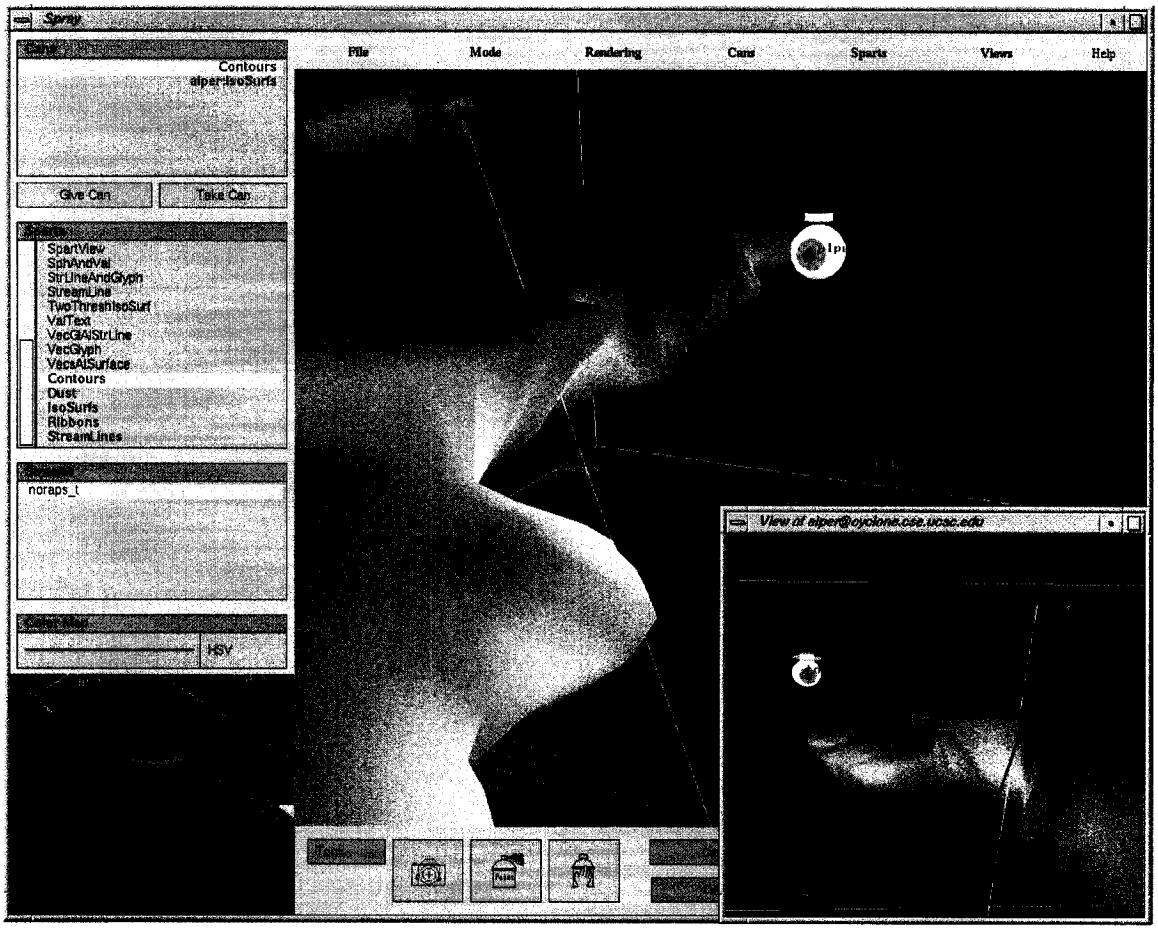


FIGURE 1.5 Interface de *Spray*—en mode coopératif—illustrant une session avec deux collaborateurs. La fenêtre principale montre la vue du premier usager d'où l'on voit l'avatar de l'autre usager. La fenêtre publique permet au premier usager de voir le point de vue du second (tiré de Pang et Wittenbrink (1997)).

1.3.3 CAVE6D

Le système à l'origine de CAVE6D, CAVE5D (Wheless et al. 1996) est un cadre d'applications en réalité virtuelle co-développé par Glen Wheless et Cathy Lascara de l'université Old Dominium et Bill Hibbard de l'université du Wisconsin-Madison. CAVE5D permet d'explorer des données numériques multidimensionnelles, telles que des données atmosphériques ou océanographiques, sur un média immersif comme un Immersadesk ou une *CAVE Automatic Virtual Environment* (CAVE). CAVE5D s'appuie principalement sur deux librairies : les CAVELib[1] pour l'affichage en immersion et Vis5D[2] pour la visualisation scientifique.

CAVE6D (Wheless et al. 1998) émerge de l'intégration de CAVE5D et de la librairie CAVERNSoft pour produire un environnement télé-immersif permettant ainsi à plusieurs usagers de CAVE5D d'interagir simultanément sur des données partagées. CAVERNSoft (Leigh et al. 1999b) est une librairie développée à l'université de L'Illinois à Chicago par l'*Electronic Visualisation Laboratory* dans le cadre du réseau CAVERN (CAVE Research Network).

Ce réseau —un collectif d'industries et d'institutions de recherche dotées d'environnements immersifs, de ressources computationnelles et de réseaux haut débit— a comme agenda le développement d'outils et applications pour l'ingénierie télé-immersive, l'éducation et la formation, la visualisation scientifique et le contrôle de calculs numériques (*computational steering*). CAVERNSoft est conçu comme un intergiciel pour la création d'un EVC, offrant des fonctionnalités d'affichage immersif, de partage du monde, de gestion des avatars, de communication et de persistance (Leigh et al. 1997).

[1] `http://www.vrco.com`
[2] `http://vis5d.sourceforge.net`

Architecture et topologie

L'architecture de la couche réseau est celle de CAVERNSoft étant donné que CAVE6D repose largement sur les fonctionnalités offertes par cette librairie. Le concept central à CAVERNSoft est celui d'un IRB (*Information Request Broker*) qui est un référentiel autonome de données persistantes accessible via une variété d'interfaces réseau. L'emphase est mise sur la flexibilité en fournissant une structure capable de fournir différentes topologies et différentes qualités de service. La figure 1.6 nous montre comment les principales topologies peuvent être exprimées avec des IRBs. L'IRB combine les fonctionnalités de base de données et d'interface réseau. Une application qui veut communiquer avec l'extérieur invoque son IRB à travers une interface, dite IRBi. Son IRB est alors en communication avec les IRB des autres clients ou serveurs. Notons que cette architecture ne fait pas la différence entre un client et un serveur, ainsi tous les participants interconnectés peuvent autant recevoir qu'émettre de l'information. Chaque donnée à publier est associée à une clé unique. Les IRB s'abonnent à ces clés et sont avertis lors de changements.

CAVE6D présente une topologie centralisée. Les données à visualiser (un fichier Vis5D) sont répliquées chez chaque collaborateur avant de démarrer l'application. Les clients se connectent à un serveur-IRB central à travers leurs IRB et envoient continuellement l'information sur leur état local au serveur qui relaie chaque paquet reçu aux autres clients. À cause d'une incompatibilité entre les fils d'exécution de CAVE5D et ceux de CAVERNSoft, les IRBs n'ont pas pu être directement intégrés à CAVE5D. Pour y pallier, les développeurs ont fait le choix d'utiliser deux zones de mémoire partagée (par site de collaboration), l'une servant à écrire les données à publier par l'IRB local, l'autre à récupérer celles recueillies par l'IRB local et issues d'un IRB distant. L'IRB locale emmagasine l'information dans les clés correspondantes ; celles-ci sont alors transmises aux IRB distants. Typiquement, l'information échangée contient l'identificateur de l'avatar, les données issues des dispositifs de repérage

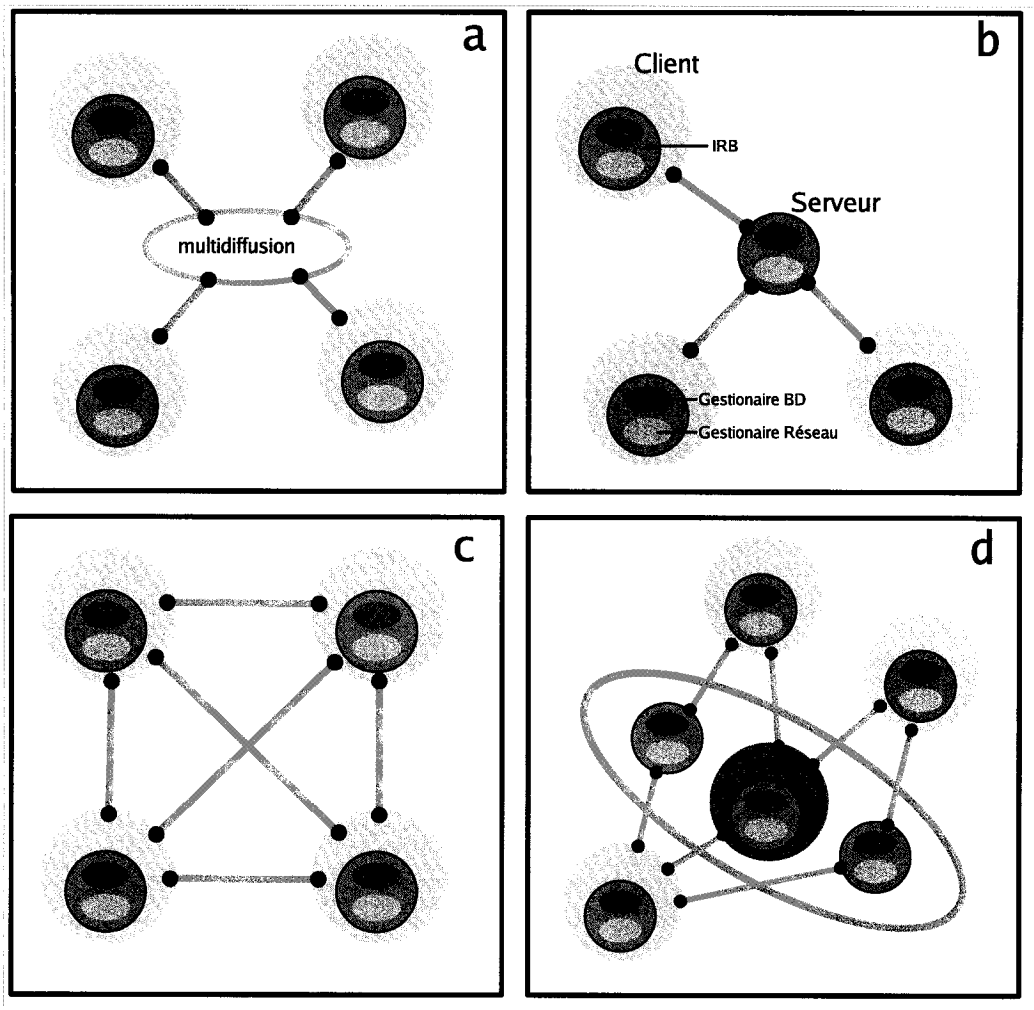


FIGURE 1.6 Différentes configurations des IRB permettent de construire les plus importantes topologies pour les EVC :
(a) Données entièrement répliquées entre les clients avec multidiffusion.
(b) Clients IRB connectés à un serveur IRB pour une topologie centralisée.
(c) Clients IRB entièrement connectés entre eux pour une topologie distribuée avec des mises à jour en poste-à-poste.
(d) Clients et serveurs IRB connectés pour former une topologie distribuée avec regroupements.

(position et orientation de la main et de la tête), une estampille pour l'ordonnancement temporel, l'état des boutons paramètres graphiques ainsi que la position de ces paramètres.

Interface et interaction

CAVE6D présente une interface très similaire à CAVE5D. Un menu fixe est affiché dans les coordonnées locales de l'usager et le suit quelque soit son point de vue. Les usagers voient le monde de leur perspective et voient les objets de visualisation ainsi que les autres collaborateurs. Ils peuvent également adopter le point de vue d'un collaborateur pour des fins de démonstration ou de tutorat. Les collaborateurs sont représentés par des avatars de différentes couleurs comprenant une tête orientée, un corps et une main munie d'un long pointeur. Le pointeur peut servir à communiquer une intention aux collaborateurs ou à interagir avec le menu. Ils peuvent aussi communiquer leurs intentions par des mouvements de tête mais l'audio-conférence n'est pas intégrée au système. Les auteurs, reconnaissant l'absolue nécessité d'un tel canal pour permettre une vraie collaboration, utilisent un outil externe à cette fin. Voir la figure 1.7 pour une saisie d'écran de CAVE6D.

Le menu permet, entre autre, d'afficher ou de masquer les objets graphiques et de changer l'état de ces OAV qui peuvent être soit locaux soit globaux. Un OAV local est une vue privée dont les modifications ne sont pas propagées aux collaborateurs, leur permettant, par exemple, d'explorer indépendamment des dimensionnalités différentes de leur visualisation. Un objet global est constamment synchronisé chez tous les clients. Tous les usagers peuvent changer l'attribut local/global des OAV. Un simple mécanisme de verrouillage est implanté lorsqu'un OAV global est en cours de modification (comme une translation) prévenant la possibilité que des contrôles contradictoires lui soient appliqués.

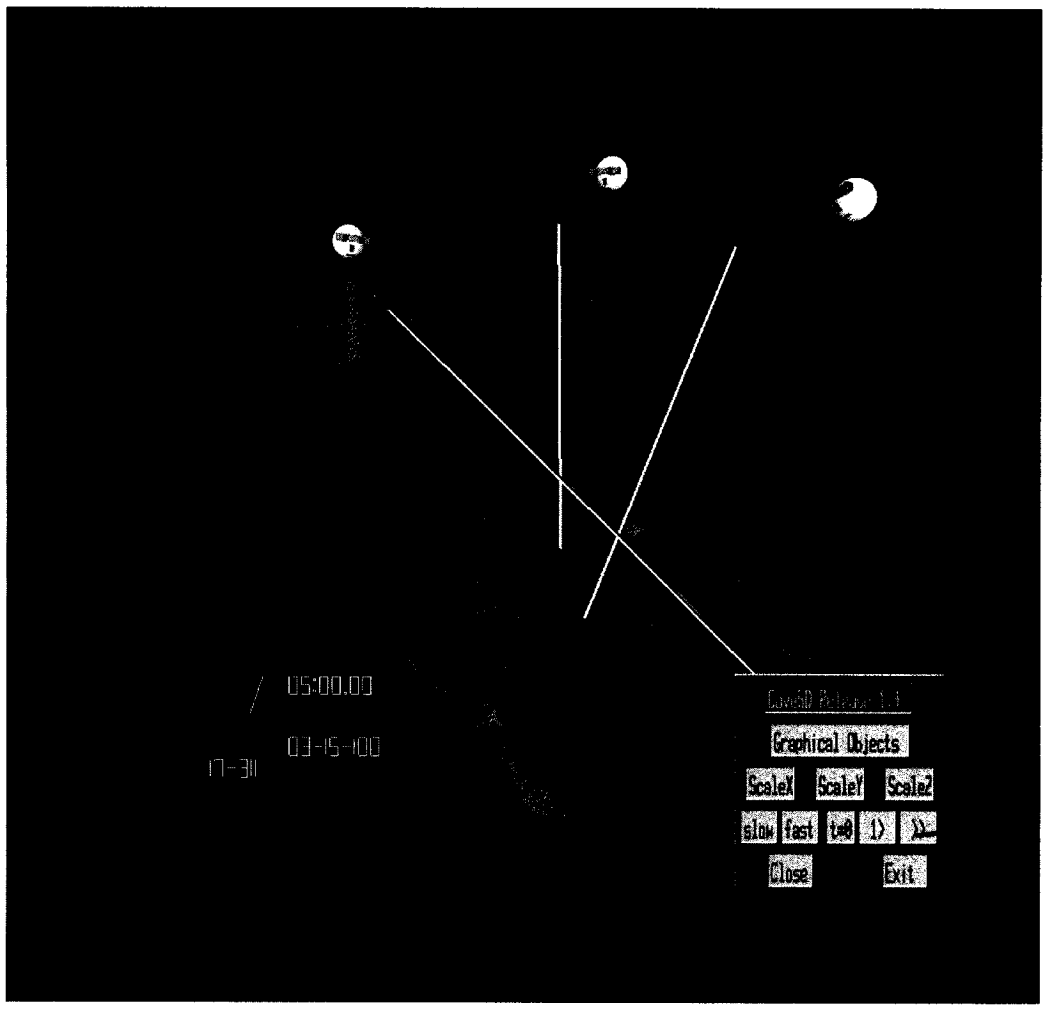


FIGURE 1.7 Télé-collaboration dans CAVE6D
Quatre usagers participent à une session de visualisation collaborative autour de données océanographiques. Cette saisie d'écran représente la vue subjective de l'usager local. Les collaborateurs utilisent un pointeur pour interagir avec leurs menus ainsi qu'avec la scène.

1.3.4 DIVE

Le système DIVE (*Distributed Interactive Virtual Environment*) (Carlsson et Hagsand 1993; Frécon et Stenius 1998), développé au *Swedish Institute of Computer Science*, est un cadre d'application pour le développement d'environnements virtuels distribués, d'interfaces usager et d'applications. DIVE existe depuis un nombre substantiel d'années (les premières versions datent de 1991) et a servi au développement de prototypes d'environnements virtuels aux applications variées, et ce, sur une large palette d'interfaces et de plateformes d'exécution. Les exécutables binaires des dernières versions sont disponibles[3] pour un usage non commercial. DIVE est optimisé avant tout pour le développement rapide d'EVC surtout pour de la téléconférence augmentée.

Architecture et topologie

Le concept central de l'architecture de programmation de DIVE est celui d'une base de données hiérarchique distribuée d'objets appelés « entités », constituant le monde virtuel. En plus d'informations graphiques, une entité peut contenir des données définies par l'utilisateur ainsi que des descriptions de comportements autonomes. Les usagers autant que les applications interagissent à travers cette base de données, le système ne faisant pas de distinction fonctionnelle entre ceux-ci.

L'emphase de cette architecture est mise sur la réactivité du système aux interactions de l'usager, c'est-à-dire que ses actions ont une incidence locale immédiate avant de se propager aux usagers distants. Le système tolère que les hôtes distants ne soient pas immédiatement en phase avec les dernières modifications au profit d'une meilleure interactivité locale. La topologie est distribuée et repose sur de la multidiffusion poste

[3] `http://www.sics.se/dive`

à poste. La base de données est (partiellement) répliquée chez chaque hôte abonné aux groupes de multidiffusion qui le concernent. Ainsi les mondes virtuels, bien que répliqués, résident en fait sur le « réseau virtuel » formé par les hôtes. Chaque hôte peut abriter des processus différents (conférant au système sa topologie distribuée) qui, agissant sur la base de données, affectent le ou les mondes virtuels existants sur le « réseau ».

Au démarrage, chaque processus DIVE se connecte à un serveur de résolution de nom spécial, DIVESERVER, qui permet d'obtenir des adresses multidiffusion en échange de noms de mondes virtuels. Le processus se connecte alors au processus distant le plus proche à qui il envoie des requêtes sur l'état du monde virtuel. Le premier processus connecté charge l'état initial du monde depuis des adresses internet (supportant le format internet de DIVE ou le VRML). Les entités du monde sont persistantes tant et aussi longtemps qu'un processus DIVE est à leur écoute.

Il existe deux types de messages envoyés par DIVE, correspondants aux différents types de données envoyées pendant une session : les modifications à la base de données sont envoyées avec un protocole multidiffusion fiable, assurant l'arrivée et contrôlant l'ordonnancement des paquets ; les flux de données continus (comme l'audio et la vidéo) sont envoyés à l'aide d'un protocole multidiffusion non fiable. DIVE repose sur MBone pour la multidiffusion mais fournit une dorsale logicielle, DIVEBONE, pour interconnecter des îles de multidiffusion entre elles.

DIVE est surtout un cadre d'application pour développer rapidement des EVC, offrant plusieurs interfaces de programmation. Typiquement, une application DIVE manifeste trois comportements différents (tous optionnels) sur l'environnement : introduire des entités (partagées) au sein de l'environnement ; recueillir les événements issus de parties choisies de l'environnement, généralement—mais pas nécessairement—d'entités introduites ; réagir à ces événements en modifiant la base de données. Le modèle de

programmation (Frécon 2004) pour DIVE offre plusieurs avenues à cette fin. On peut programmer une application monolithique en langage de haut niveau (C/C++ ou Java) en compilant des composantes des librairies fournies créant ainsi une application indépendante. Une autre avenue est de charger une application dynamiquement à l'exécution à partir d'une ou plusieurs applications monolithiques. Il y a deux interfaces majeures pour le chargement dynamique. La première est d'écrire un plugiciel (extension) en C ou C++ pouvant être chargé soit au lancement de l'application soit au cours de l'exécution, en réaction par exemple à un événement de l'environnement. La seconde permet réellement le prototypage rapide avec un langage scripté basé sur le *Tool Command Language* (TCL). Chaque entité de la base de données peut être associée à un script DIVE/TCL décrivant son comportement. Finalement, des applications externes peuvent être interfacée avec une application DIVE au travers d'une *DIVE Client Interface* (DCI), avec un modèle client-serveur sur une connexion TCP. L'application externe est représentée dans l'environnement comme une entité spécifique et le langage de communication est le DIVE/TCL.

Interface et interaction

La version courante officielle de DIVE (3.3) permet principalement de créer des applications sur un paradigme classique de périphériques (écran-clavier-souris) et sur un paradigme immersif à l'aide d'un visiocasque et d'interfaces de pointage repérées. Nous pouvons citer quelques applications développées à l'aide de DIVE, représentatives de ses possibilités :

Téléconférence. Une des premières applications de DIVE est la téléconférence. Un tel environnement (voir figure 1.8) permet aux collaborateurs sur stations de travail de se rencontrer dans une salle de conférence virtuelle. Ils communiquent par audio et peuvent s'échanger des documents multimédia. Ceux-ci peuvent être liés à des

sources externes de données, le système intégrant, par exemple, un navigateur WEB et un interpréteur MIME (*Multipurpose Internet Mail Extension*) afin de déterminer comment visualiser chaque type de fichier.

London Travel Application. Cette application, développée au University College London par Steed et al. (1999) dans le cadre du projet COVEN (Frécon et al. 2001), est un prototype visant à étudier un EVC à grande échelle représentant un monde assez vaste pouvant accueillir plusieurs télé-collaborateurs. L'application permet aux usagers de visiter virtuellement un site existant (une partie de Londres de 160 km^2) et de planifier une rencontre à l'avance.

CAD et RV. Dans ce projet (Törlind et al. 1999), le système DIVE a été adapté afin de permettre de l'interfacer avec un logiciel existant de Design Assisté par Ordinateur (DAO), l'emphase étant mise sur le design collaboratif. Les auteurs privilégient l'approche TCAO qui est d'utiliser une palette d'outils pour bâtir un environnement collaboratif. Ainsi ils utilisent DIVE pour le monde virtuel, *Smile!* (Johanson 1998) pour la vidéoconférence et I-DEAS comme logiciel de DAO. Une interface en CORBA a été développée pour donner accès à DIVE à la géométrie des pièces ainsi qu'aux informations relatives à la gestion du processus. Les usagers dans un environnement de téléconférence peuvent, par exemple, discuter de résultats de simulation ou de séquences d'assemblage, transférer des modèles géométriques aux autres, interagir avec la géométrie (inspecter, annoter) et vérifier l'interférence entre deux pièces dans un assemblage.

Notons que des travaux de recherche (Steed et al. 2001) ont mené à dériver des versions immersives de DIVE fonctionnant sur plateformes SGI-IRIX avec les CAVELibs pour l'affichage. Ces versions ont permis d'effectuer des recherches sur la collaboration

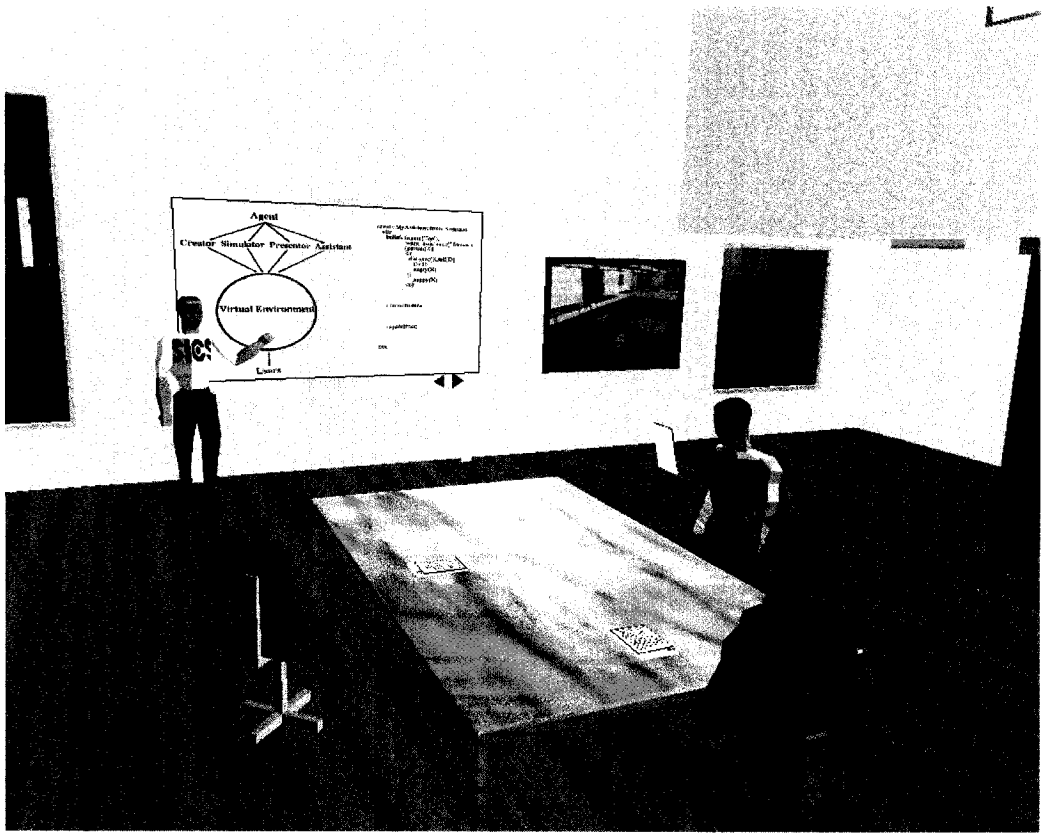


FIGURE 1.8 Une session d'une application de téléconférence créée avec DIVE. Trois usagers participent à cette conférence et communiquent avec du texte et par canal audio. Les documents sur la table et le tableau sont actifs, affichant textes et graphiques provenant d'outils externes.

téléimmersive pour étudier par exemple l'impact de la co-présence sur la performance de tâches collaboratives (Mortensen et al. 2002).

1.3.5 MASSIVE

L'environnement MASSIVE (Model, Architecture and System for Spatial Interaction in Virtual Environments)—développé initialement par Chris Greenhalgh de l'université de Nottingham (Greenhalgh et Benford 1995)—est un système de téléconférence en réalité virtuelle. Il permet aux usagers de communiquer sur trois médiums principaux (texte, audio et graphiques 3D) et repose sur un modèle d'interaction spatiale développé par Steve Benford. Ce modèle, fortement ancré dans un paradigme d'espace, permet une gestion efficace des informations circulant d'un processus distant à un autre en se basant sur un critère de perception ou de conscience d'autrui (en anglais *awareness*). Le système MASSIVE a connu trois versions principales où d'importants changements architecturaux et topologiques ont pu voir le jour. Nous allons discuter de la première version, MASSIVE-1, car les versions subséquentes ont été bâties sur le modèle spatial d'interaction mais uniquement en l'optimisant, sans changements majeurs du paradigme.

MASSIVE-1 – Modèle spatial d'interaction

La première version de MASSIVE est une implantation directe du modèle spatial d'interaction (Benford et Fahlén 1993; Benford et al. 1994). Ce modèle permet d'évaluer le degré de conscience qu'ont les objets d'un espace virtuel les uns envers les autres. Le modèle se définit autour de plusieurs abstractions : l'*espace*, les *objets*, le *médium*, l'*aura*, le *focus*, le *nimbus*, les *adaptateurs* et les *frontières*.

L'*espace* est considéré comme étant un espace métrique de dimensionnalité arbitraire. Ce espace est peuplé par des *objets* qui peuvent aussi bien représenter des personnes, de l'information ou tout autre artefact informatique. Les objets communiquent entre eux à travers certains *médiums* (textuels, visuels, auditifs...).

L'*aura* permet de déterminer quelles paires d'objets peuvent interagir dans l'environnement. L'*aura* est ainsi définie comme étant le sous-espace (généralement une boule) dans lequel un objet, à travers un certain médium, exprime sa présence et peut potentiellement communiquer. Si les *auras* de deux objets s'intersectent c'est qu'il y a possibilité d'interaction entre eux ; ils sont alors mis en contact et sont, par la suite, responsables de contrôler leurs interactions. Ils se basent, pour ce faire, sur des niveaux quantifiables de *conscience* mutuelle. Le niveau de conscience n'est pas nécessairement symétrique et, comme pour l'*aura*, il dépend du médium. Ce niveau est calculé en multipliant le *focus* de l'objet observant par le *nimbus* de l'objet observé. Le *focus* est une mesure du degré d'attention qu'a un objet observateur tandis que le *nimbus* est une mesure du degré de projection qu'a un objet observé. Ce niveau de conscience ou d'attention permet alors de filtrer ou moduler les interactions. Par exemple, plus un objet est attentif à un émetteur de flux audio plus le volume sera fort ; un objet peut décider de ne recevoir que les version écourtées d'un flux textuel si son niveau d'attention pour l'émetteur est en bas d'un certain seuil.

Les paramètres servant à établir le niveau l'attention des objets—l'*aura*, le *focus* et le *nimbus*—peuvent être modifiés de quatre façons. D'abord de manière implicite avec des actions spatiales telles que des translations ou rotations ; ces paramètres étant généralement attachés à l'objet en mouvement. Ils peuvent aussi modifiés explicitement à travers quelques réglages simples : il est possible, par exemple, de passer d'un *focus* large à étroit avec un clic de souris ou tout autre action d'un périphérique d'entrée. Une façon plus extensible de les modifier peut s'effectuer à travers les *adaptateurs*. Ceux-ci permettent de changer l'*aura*, le *focus* ou le *nimbus* des objets qui les activent ; par exemple, un adaptateur de type « podium » permet d'amplifier l'*aura* et le *nimbus* (pour l'audio) de l'objet qui l'active. Finalement les *frontières* de l'espace offrent des mécanismes pour partitionner l'espace en plusieurs régions, contrôler les mouvements et influencer les propriétés d'interaction de l'espace (à travers les trois paramètres définissant l'attention).

Ce modèle spatial d'interaction s'inscrit dans le cadre plus large des techniques de gestion de l'intérêt dans les systèmes distribués. Cette gestion permet de minimiser la quantité d'informations à échanger entre les hôtes d'un système distribué en quantifiant l'intérêt ou l'attention que les objets manifestent les uns envers les autres. Notons que cette gestion prend surtout son sens pour les systèmes distribués où le nombre d'hôtes et d'objets sont très élevés.

La figure 1.9 montre un exemple d'utilisation de MASSIVE appliqué à la télévision augmentée (Benford et al. 1998).

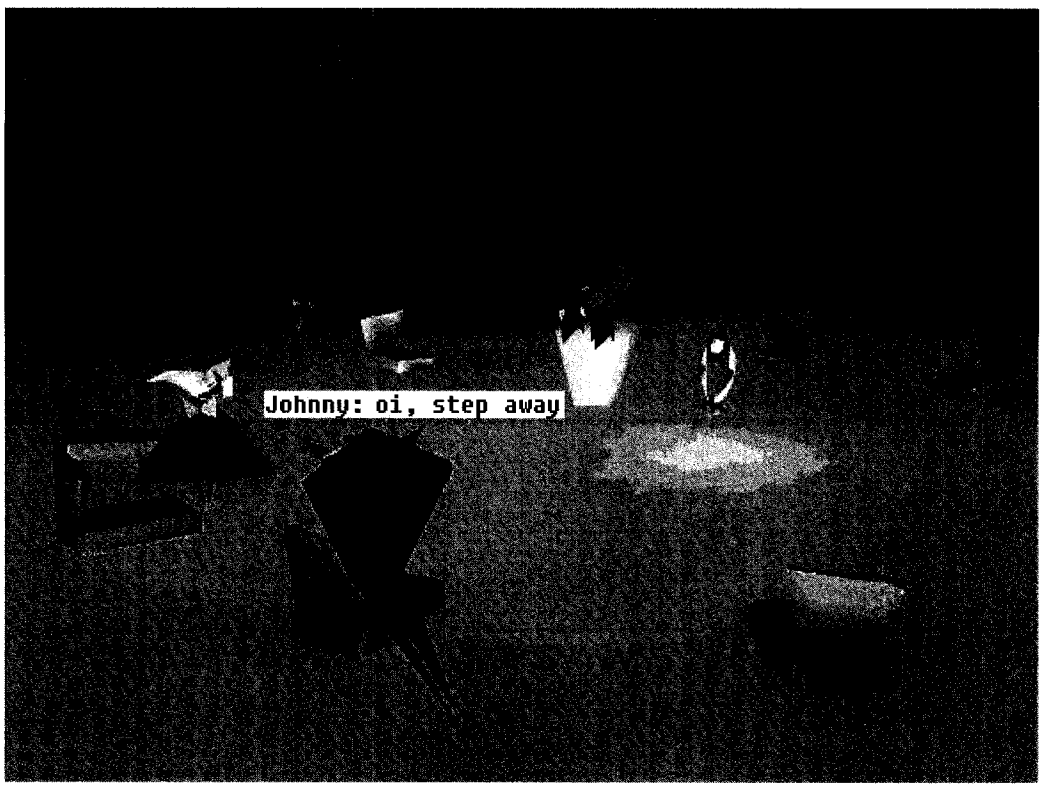


FIGURE 1.9 En août 1997, une émission spéciale était présentée à la télévision britannique Channel 4. Il s'agissait d'un jeu télévisé d'une heure, présentée en temps réel, où les participants (deux célébrités) devaient compléter plusieurs jeux dans un monde virtuel créé avec MASSIVE. Ces participants pouvaient demander de l'aide de la part des autres spectateurs/acteurs présents dans ce monde virtuel. L'événement a connu ce soir-là une audience allant jusqu'à 135 participants virtuels et 200 000 téléspectateurs.

1.3.6 VisualEyes

Collaborative VisualEyes (Smith et al. 2000) est une extension collaborative à l'environnement VisualEyes. Cet environnement, développé au sein des divisions recherche et développement de la compagnie General Motors, permet de visualiser des modèles de véhicules automobiles ainsi que d'effectuer des revues de design. Les ingénieurs, designers et cadres de chez GM ont à leurs disposition plusieurs types d'affichage : soit une projection immersive comme un ImmersaDesk ou une CAVE (pour visualiser les intérieurs), soit une projection sur un *Wall*, un mur d'écrans offrant une large surface d'affichage, permettant ainsi d'afficher le modèle en stéréoscopie et en grandeur réelle, ce qui est adapté à la visualisation extérieure du véhicule.

L'extension collaborative a été effectuée en collaboration avec le *Electronic Visualisation Laboratory* qui développe la librairie CAVERNSoft.

Architecture et topologie

L'architecture de VisualEyes s'articule autour de trois couches logicielles principales : la couche système est en fait une version « maison » des CAVELibs et s'occupe donc de l'affichage et de l'interaction. La couche du graphe de scène gère la hiérarchie des objets ainsi que l'importation de la géométrie depuis les formats des logiciels de DAO utilisés chez GM. La couche outils permet de charger dynamiquement des outils définis comme des machines à états, permettant relier les actions de l'utilisateur à des comportements. Une couche principale coordonne ces trois couches par exemple en reliant au démarrage, avec des fonctions de rappel, la couche outils à des fonctions primitives de la couche système ou du graphe de scène telles que l'intersection d'un rayon avec la scène, la sélection d'une géométrie ou la modification d'un nœud du graphe de scène. Un programme externe gère les entrées des périphériques et les rend

disponibles à VisualEyes à travers un espace de mémoire partagée

Cet espace de mémoire partagée est aussi établi pour faire communiquer VisualEyes et les fonctions de CAVERNSoft. Les données partagées sont ainsi, d'une part, le graphe de scène et, d'autre part, les données acquises par les périphériques de pointage et de repérage. Selon le paradigme de CAVERNSoft, ce sont les IRB qui assurent la communication entre les sites distants à travers des clés. Les clés publiées par les IRB sont des nœuds du graphe de scène tels que les propriétés des matériaux des objets, les transformations et l'éclairage, mais pas la géométrie ni les textures (qui doivent être localement présentes à chaque site de collaboration). Les IRB se chargent alors de synchroniser les clés publiées. Les clés empruntent des canaux dont la nature peut être spécifiée (TCP, UDP ou multidiffusion). Un mécanisme de verrouillage est utilisé, faisant en sorte qu'une clé ne peut être modifiée par plus d'un site à la fois.

Interface et interaction

Un scénario typique de l'utilisation de VisualEyes est la revue de design où un designer sur un poste de travail modifie le design dans un modeleur tel que Alias tandis que des collaborateurs dans une CAVE ou en face d'un *Wall* commentent les modifications. La figure 1.10 montre l'interaction de deux télé-collaborateurs discutant du design de l'intérieur de l'automobile.

FIGURE 1.10 Session télé-collaborative de VisualEyes dans une CAVE. L'usager évalue le design de l'intérieur du véhicule avec un télé-collaborateur dont l'avatar est représenté sur le siège du passager. Pour un meilleur réalisme, un véritable siège d'automobile est installé dans cette CAVE. (Leigh et al. 1999a)

1.3.7 Autres EVC

Nous avons rencontré d'autres EVC dans notre recherche mais nous ne les détaillons par car il n'apportent pas de nouveaux concepts, uniquement des améliorations ou optimisations. Nous pouvons penser à MASSIVE-2 et MASSIVE-3 (Greenhalgh 1996; Greenhalgh et al. 2000) qui amènent des améliorations respectivement au niveau de la topologie réseau et de la gestion de l'attention.

DIVERSE (Kelso et al. 2003) est un ensemble de logiciels modulaires pour la création de simulations virtuelles distribuées utilisant Performer[4], une architecture de mémoire partagée distribuée, un affichage immersif et une topologie centralisée.

TIDE (Sawant et al. 2000) est un environnement immersif distribué de visualisation scientifique dont l'architecture est basée sur CAVERNSoft (voir section 1.3.3) et permettant la visualisation collaborative de larges ensembles de données multidimentionnelles.

NPSNET (Macedonia et al. 1995) est un système distribué de simulations de manœuvres militaires créé par la *Naval Postgraduate School* en Californie qui permet à plusieurs centaines d'usagers d'interagir simultanément dans le monde virtuel. Le système montre une topologie répliquée en multidiffusion et implémente une gestion de l'attention évoluée : premièrement, le monde est divisé en zones hexagonales et les usagers envoient les changements d'état du monde à ceux qui sont d'abord dans leur zone. Deuxièmement, un mécanisme d'estimé de position (*dead reckoning*) permet d'éviter l'envoi inutile d'informations : un calcul prédictif de la trajectoire des objets est implémenté autant chez l'émetteur que chez le récepteur. Si la position prédite est la même que la position courante (à une erreur près) la nouvelle position n'est pas envoyée, économisant ainsi la bande passante lorsque les objets ne bougent pas

[4] `http://www.sgi.com/products/software/performer`

ou lorsqu'ils ont une trajectoire linéairement prévisible.

1.4 Résumé

Nous avons vu dans cette revue de littérature quelques modèles de visualisation scientifique et les concepts les plus souvent rencontrés dans les environnements virtuels de collaboration. Nous avons répertorié l'immersion, le partage du monde, la présence et la co-présence, le synchronisme, les conversations publiques et privées et la gestion de la cohérence et de l'attention. Nous avons ensuite entrepris d'examiner plusieurs EVC bien implantés dans leur domaine et avons illustré l'expression de ces concepts dans ces EVC.

Nous retenons de cette revue de littérature un ensemble de caractéristiques que présentent les EVC : l'immersion, la téléconférence, la persistance du monde, l'aspect distribué, la gestion de l'attention, la gestion de la cohérence et la visualisation scientifique. Nous présentons, pour finir cette revue, une comparaison synthétique des différents EVC étudiés dans le tableau 1.3.

TABLEAU 1.3 Comparaison des différents EVC étudiés

EVC	**Caractéristiques**						
	Immersif	Télé-conférence	Monde persistant	Distribué	Gestion de l'attention	Gestion de la cohérence	Visualisation scientifique
COVISA				√			√
CSpray				√		√	√
CAVE6D	√						√
DIVE		√	√	√	√	√	
NPSNET		√	√	√	√	√	
MASSIVE		√		√	√	√	
VisualEyes	√	√	√			√	

CHAPITRE 2

ÉVALUATION DES BESOINS ET APPROCHE RETENUE

Nous avons vu au cours du précédent chapitre les différentes caractéristiques que peuvent présenter les EVC ; les possibilités sont nombreuses et il nous faut en restreindre l'étendue pour se concentrer sur les besoins du projet et les hypothèses de travail que nous voulons explorer.

Nous allons d'abord décrire quelles sont les attentes d'intégration avec les autres tâches de MOSAIC et les contraintes pour une collaboration immersive et non-immersive ainsi que pour une interface intuitive. Puis, nous exprimerons les besoins fonctionnels en les illustrant par des scénarios de collaboration. Nous serons alors en mesure de présenter nos choix et hypothèses en exposant synthétiquement les éléments retenus qui satisferont les contraintes et permettront de répondre aux besoins exprimés dans les scénarios proposés.

2.1 Attentes d'intégration avec les autres tâches

L'objectif global au sein de MOSAIC étant l'intégration multidisciplinaire, chaque tâche du projet vise ultimement à pouvoir communiquer avec les autres. En plus d'une communication/collaboration entre utilisateurs, l'ECDI doit aussi offrir une communication avec d'autres programmes. Il est prévu à terme que les communications inter-programmes s'effectuent par le biais de protocoles développés également au sein de MOSAIC. Nous allons ici discuter de la nature des informations à échanger et des différentes façons de le faire.

2.1.1 Données à échanger

L'ECDI doit servir à des fins de visualisation scientifique et de contrôle distant de simulations dans une activité de design. Il s'agit d'abord de pouvoir échanger la géométrie représentant les modèles à visualiser. Il y a ainsi deux choix à faire pour la géométrie seulement : quelle représentation et quel support utiliser. Pour la représentation, on considérera les surfaces uniquement : discrètes (ensemble de polygones reliés approximant linéairement la surface) et paramétrées (nous choisissons à cet effet une représentation par surfaces NURBS). Cette restriction sera suffisante pour nos besoins car plusieurs tâches de MOSAIC peuvent fournir leurs résultats géométriques sous forme de surface. Quant au support, étant donné que plusieurs utilisateurs de MOSAIC se servent de la librairie `pirate` pour manipuler géométries et topologies, nous utiliserons le format de fichier de `pirate`—qui correspond aussi à celui de `VU` (Pic et Ozell 2000).

En ce qui concerne le dialogue avec les autres programmes des tâches de MOSAIC nous considérerons d'abord les programmes qui, en entrée, acceptent un ensemble de variables de design scalaires (entières ou réelles) et offrent, en sortie, une géométrie ainsi mise à jour. Il sera possible de communiquer, par exemple, avec des programmes générant des surfaces à partir de paramètres de design (géométriques ou autres).

2.1.2 Infrastructure de communication

Le projet MOSAIC intègre le développement de protocoles d'échange entre les différentes tâches. Le moyen de choix retenu à cette fin est d'utiliser des *Web Services* pour gérer les requêtes et réponses entre les programmes ainsi que pour publier les offres de services de chacun. Aussi, un travail plus approfondi de développement d'ontologies des domaines permettra à chacun des programmes de demander des données aux

autres avec un niveau d'abstraction indépendant des représentations. Par exemple, un programme requérant une surface en entrée demandera sa préférence au *Web Service* qui appellera les convertisseurs nécessaires (traduction, discrétisation, interpolation...) pour présenter les données selon la représentation et le support demandés. Cette voie nécessite donc que chaque programme collaborateur aie une interface *Web Service Client* dérivée de la description WSDL du ou des services en question.

Cependant, une solution intermédiaire est également nécessaire pour les échanges où le *Web Service* n'est pas encore prêt. Il s'agit alors d'une communication plutôt ad hoc, locale ou distante, dont se charge un script de commandes opérant comme façade mandataire et relayant un vecteur d'arguments scalaires.

2.2 Collaboration immersive et non-immersive

Un objectif essentiel de ce travail étant l'immersion dans une activité de collaboration, notre EVC doit évidemment offrir une visualisation immersive sur des supports comme une CAVE ou un ImmersaDesk. Le système doit donc permettre à des collaborateurs dans de tels environnements de communiquer et d'interagir sur une scène partagée.

Toutefois, ces environnements immersifs ne sont pas très répandus ; notre EVC doit donc être aussi fonctionnel sur un poste de travail classique (écran 2D, clavier et souris). Nous devons alors nous assurer que l'ECDI permette la collaboration entre toutes les combinaisons d'environnements, immersifs comme non-immersifs.

2.3 Interface intuitive

Notre système étant avant tout immersif, nous voulons un paradigme d'interface qui soit différent de ce que l'on rencontre sur un poste de travail classique. En effet, nous

sommes si habitués à une interface en deux dimensions qu'il est tentant, à prime abord, de reproduire ces concepts familiers de gadgets logiciels[1] comme des menus déroulants, boutons radio, icônes, etc.

En revanche, l'expérience et la littérature indiquent que ces concepts conviennent mal à un environnement nativement tridimensionnel. Sans non plus chercher un paradigme sophistiqué d'interaction immersive (un domaine très riche et très actif mais ne faisant pas l'objet de ce travail de recherche), nous voulons un mécanisme simple et intuitif d'interaction.

Enfin, il nous faut aussi considérer, tout comme précédemment, la compatibilité sur poste de travail classique où l'affichage et le périphérique d'entrée principal (la souris) s'effectuent en deux dimensions.

2.4 Besoins fonctionnels

Nous allons maintenant exprimer les besoins fonctionnels de l'ECDI en illustrant des situations de design en collaboration. Les scénarios sont introduits par une brève description pour ensuite en détailler l'interface en immersion, puis, le cas échéant, discuter des interactions entre collaborateurs.

On considère que le périphérique d'entrée principal pour l'environnement en immersion est un bras de repérage (*tracking wand*), c'est-à-dire un dispositif de localisation à 6 degrés de libertés équipé de 3 boutons. La communication vocale, essentielle, entre les télé-collaborateurs s'effectuera, par audio-conférence avec un logiciel tiers de téléphonie IP.

Nous présentons ainsi quatre scénarios : design préliminaire, design dirigé, revue de

[1] en anglais *widgets*

design et contrôle interactif d'un processus extérieur.

2.4.1 Design préliminaire

Ce scénario fait intervenir un ou plusieurs usagers qui, dans l'environnement en immersion, peuvent éditer des surfaces NURBS. Ils peuvent charger et sauvegarder leurs modèles et déplacer des points de contrôle. Il peuvent travailler sur le même modèle ou sur des modèles séparés pour ensuite les comparer.

Interface en immersion

Quand l'usager entre dans le monde virtuel, une « bibliothèque à fichiers » se trouve sur sa gauche et permet à l'usager de charger ou sauvegarder les modèles. Une corbeille se trouve sur sa droite et permet de supprimer des objets de la scène. L'interaction avec l'environnement se déroule comme suit :

- À l'aide de la gâchette du bras de repérage, l'usager choisit un fichier de la bibliothèque à fichiers et, en effectuant un glisser-déposer du nom de fichier en dehors de la bibliothèque, lance le chargement de ce fichier. Le modèle apparaît alors à l'endroit du lâcher.
- S'il approche le bras de repérage d'un point de contrôle, celui-ci change de couleur pour indiquer une possibilité d'interaction. Toujours à l'aide d'un glisser-déposer, l'usager peut déplacer ce point de contrôle et modifier l'allure de la surface.
- Si l'usager approche le bras de repérage d'un objet, il peut le déplacer la faire pivoter avec la gâchette.
- S'il déplace un objet vers la bibliothèque à fichiers, la sauvegarde du modèle est déclenchée.
- S'il déplace un objet dans la corbeille et l'y lâche, l'objet est alors retiré de la scène.

Interaction entre les télé-collaborateurs

Les usagers sont représentés dans le monde virtuel par des avatars. Dans l'environnement immersif, la direction de leur regard et la position de leur bras de repérage sont ainsi représentés. Lorsqu'un usager sélectionne un objet (fichier, modèle, point de contrôle...), il obtient également l'exclusivité de sa modification ; les autres usagers voient l'objet être modifié mais ne peuvent s'en emparer tant qu'il est détenu par le premier usager.

2.4.2 Design dirigé

Dans ce scénario, des usagers de l'environnement immersif collaborent avec un usager manipulant un logiciel CAD (tel CATIA) sur un poste de travail. Ce dernier peut exporter son modèle dans un format de fichier d'échange (tel STEP), le rendant accessible à l'environnement de collaboration. Les usagers de l'ECDI peuvent charger ce modèle, l'étudier, en discuter et demander au designer (par audio-conférence) d'y apporter des modifications. Une fois les modifications exécutées dans CATIA, le designer exporte de nouveau le modèle. L'ECDI détecte une nouvelle version du modèle et propose à ses usagers de le charger dans l'environnement. Les usagers de l'ECDI peuvent aussi apporter des modifications mineures au modèle, par exemple :

- Modifier certaines primitives : cylindres, sphères, NURBS...
- Déplacer et faire pivoter les sous objets constituant le modèle.

Les modifications effectuées peuvent alors être sauvegardées dans un fichier texte (en XML) décrivant explicitement le type et le contenu de chacun des changements. Ce document est transmis au designer pour l'aider à mieux interpréter les requêtes de ses collaborateurs. Les usagers ont également la possibilité de charger un fichier de description de modifications préalablement enregistré : le modèle associé est alors

chargé et les modifications sont exécutées.

Interface en immersion

L'interface est similaire au scénario précédent. Les utilisateurs ont toujours accès à des manipulations de base sur les objets à l'aide de la gâchette du bras de repérage.

Interaction entre les télé-collaborateurs

Les usagers de l'ECDI communiquent par audio avec le designer sur sa station CATIA. La collaboration est asynchrone entre ces deux parties car cela peut prendre du temps entre une requête de modification et le résultat. On prévoit ainsi un système de requêtes/notification : un requête soumise est représentée dans le monde virtuel par un objet en icône. Lorsque le résultat est disponible une notification visuelle ou sonore est alors émise : les usagers de l'environnement peuvent alors saisir l'icône représentant la nouvelle version et l'amener dans la zone de travail provoquant son chargement.

2.4.3 Revue de design

Dans ce scénario les usagers peuvent visualiser un modèle géométrique, comparer plusieurs versions et annoter des objets du modèle. Les annotations sont soit auditives, soit textuelles. Dans le cas de notes textuelles, le scénario fait intervenir un collaborateur sur un poste de travail qui rédigera les notes pour les usagers en immersion. On peut ensuite associer une ou plusieurs annotations à un sous objet du modèle et sauvegarder la session. Celle-ci peut ultérieurement être restaurée par une opération de chargement.

Interface en immersion

L'interface contient des éléments communs aux scénarios précédents : bibliothèque à fichiers et corbeille. On ajoute ici une « boîte à notes » qui permet d'ajouter et de retirer des annotations au modèle. La boîte à notes contient deux objets : un pour les notes textuelles et un pour les notes auditives. Pour créer une note auditive l'usager prend l'objet de note auditive et le sort de sa boîte, une copie est maintenant à sa disposition. L'objet contient des boutons clairement identifiables : lecture, enregistrement et arrêt. L'usager en immersion peut, similairement, créer une nouvelle note textuelle mais il ne peut pas éditer son contenu. On peut prévoir de mettre à sa disposition une liste préétablie de notes mais s'il veut un contenu spécifique, il doit compter sur son collègue collaborant sur un poste de travail. Une fois l'annotation prête, on peut la déplacer dans la zone de travail. L'usager alors la met en place en la plantant comme un panneau dans une région du modèle. Les manipulations des autres objets sont semblables à ce qui a été décrit pour les scénarios précédents.

Interface sur poste de travail

Une spécificité de ce scénario tient au fait que l'usager sur poste de travail détient une tâche qui lui est exclusive, de part la nécessité d'utiliser le clavier pour rédiger les notes textuelles. Les usagers sur poste de travail se servent du menu ou des boîtes à outils pour créer une nouvelle note et entrent au clavier le contenu dans une boîte de dialogue. Une fois validée, la note apparaît dans le monde virtuel et est disponible à tous les usagers pour qu'ils puissent la déplacer et l'associer à une zone du modèle. L'usager sur poste de travail peut également modifier une note existante en cliquant deux fois dessus pour faire apparaître la boîte de dialogue.

2.4.4 Contrôle interactif d'un processus extérieur

Dans ce scénario, les usagers peuvent participer à une session de design en faisant générer des modèles géométriques par un programme externe. Ils ont la possibilité de modifier la valeur de certains paramètres simples contrôlant la géométrie et d'appeler ce programme externe en les lui passant. Le résultat de l'exécution du programme est ensuite visualisable dans l'environnement en tant que géométrie. L'encapsulation des commandes pour piloter le processus extérieur et en récupérer les résultats se fait en utilisant les protocoles d'échange développés dans le cadre de MOSAIC. Typiquement, une session de design sera décrite au moyen d'un fichier XML. L'environnement, en lisant ces fichiers, propose des contrôles pour altérer les paramètres de chaque session et pour lancer l'exécution des programmes extérieurs.

Interface en immersion

À chaque session de design correspond un panneau de contrôle de paramètres contenant des gadgets logiciel (*widgets*) utilisés pour modifier la valeur de ces paramètres. Ils peuvent apparaître comme une case à cocher pour un paramètre booléen ou un bouton radio (*knob*) pour un paramètre unidimensionnel borné. Une fois les paramètres ajustés, l'usager lance l'exécution du calcul par pression d'un bouton de validation. La nouvelle géométrie apparaît dans l'espace de travail.

2.5 Approche retenue

Nous avons exploré plusieurs scénarios qui nous ont aidé à préciser différentes situations de collaboration répondant aux besoins recueillis dans le cadre de MOSAIC et illustrant concrètement notre objectif de recherche. Nous pouvons maintenant exposer

l'approche retenue pour répondre au problème d'une télé-collaboration. Nous allons pour ce faire énumérer les caractéristiques que nous retiendrons parmi celles identifiées au chapitre 1 pour notre ECDI puis nous détaillerons les hypothèses de travail et choix architecturaux pour notre prototype expérimental.

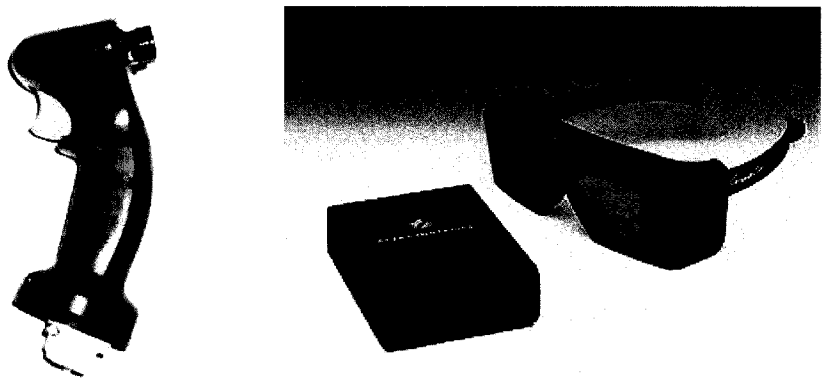
2.5.1 Éléments de l'ECDI

Notre ECDI est avant tout un EVC et nous pouvons ici énumérer les aspects nécessaires à ce travail :

Immersion. Ce sera une caractéristique majeure de notre environnement. Nous nous concentrerons cependant sur les périphériques d'affichage immersif à projection plutôt que par visiocasque ; les premiers permettant mieux la collaboration locale, par exemple avec un invité partageant la vision de l'usager principal. L'environnement devra, cependant, être également utilisable sur un affichage classique.

Périphériques d'entrée. Le périphérique de choix sera le bras de repérage—voir figure 2.1(a)—bien que les gants soient très intéressants dans la mesure où il permettent une interface gestuelle. Mais la complexité d'implémentation et de la prise en main nous éloigneraient de notre objectif d'une interface intuitive. Aussi, il ne faudra pas négliger que, pour un environnement non immersif, l'utilisation de la souris reste possible comme moyen d'interaction.

Gestion de la cohérence. Nous opterons pour une gestion minimale de la cohérence : les utilisateurs, ayant avant tout à cœur la bonne marche de leur collaboration, coopéreront volontiers à ne pas se « marcher sur les pieds ». Un simple mécanisme protégeant chaque objet de modifications tierces, une fois acquis par un des usagers,



(a) Bras de repérage

(b) Lunettes stéréoscopiques et émetteur de synchronisation

FIGURE 2.1 Périphériques de repérage

devrait suffire à notre cause.

Gestion de l'attention. Nous cherchons à permettre la collaboration entre un nombre restreint de personnes (de 2 à 5 environ) et nous supposons que le réseau informatique possède suffisamment de bande passante pour nos besoins. Nous n'aurons alors pas besoin de mécanismes de choix de réplication de données par gestion de l'attention.

Visualisation. Nous arrêtons notre choix pour le prototype sur la visualisation de surface NURBS et de leurs points de contrôle. Il sera également possible de visualiser de la géométrie polygonale mais en ce qui concerne la visualisation d'autres OAV (comme des isosurfaces par exemple) nous la reportons à une phase future, une fois la collaboration testée et validée.

Persistance du monde virtuel. Nous optons pour une persistance limitée : le monde virtuel cesse d'exister une fois le dernier usager déconnecté mais il sera quand même possible de sauvegarder l'état de la session de collaboration pour la reprendre

ultérieurement. Cette limitation sera suffisante pour nos besoins actuels.

Téléconférence. Comme nous l'avons vu précédemment, l'apport d'une liaison audio sur la qualité de la collaboration est essentiel. Nous ne prévoyons pas l'implémenter directement au sein du prototype mais cela sera quand-même disponible au niveau de l'environnement. Nous utiliserons à cet effet un logiciel extérieur étant donné qu'il en existe[2] de très bons, multi-plateformes et gratuits.

2.5.2 Choix architecturaux

Plus concrètement nous sommes maintenant confrontés à différents choix de technologie où de paradigmes d'implémentation pour réaliser notre prototype :

Topologie réseau

Étant données les hypothèses d'un nombre restreint de collaborateurs et d'une bande passante suffisante nous pouvons adopter une topologie distribuée avec mises à jour en poste-à-poste. Chaque client est alors responsable de gérer sa connexion aux autres, de leur demander, à son entrée dans une session existante, l'état actuel du monde partagé et de leur communiquer les changements apportés localement sur les objets partagés.

Interface

Le dispositif de repérage que nous utiliserons comporte une gâchette et deux boutons. Nous voulons simplifier le plus possible le « vocabulaire » d'interaction. Nous

[2]Nous pensons ici particulièrement à Skype™ (`http://www.skype.com`)

utiliserons alors la gâchette pour toutes les interactions avec les objets de la scène et le premier bouton pour la navigation au sein de la scène. L'interaction avec les objets se fera essentiellement par des opérations de glisser-déposer. La navigation se fera en pointant vers la direction de déplacement voulue et en pressant le bouton de navigation.

Les éléments présents dans l'interface seront de quatre types :

1. La géométrie : ce sont les objets que l'on peut visualiser. Dans le cas de surfaces NURBS, les points de contrôle sont affichés lors d'une interaction et on peut déplacer ceux-ci pour déformer de la surface.
2. La bibliothèque de fichiers : permet de voir le contenu d'un répertoire. Les fichiers que l'environnement peut prendre en charge apparaissent comme des livres au sein de la bibliothèque. L'action de glisser-déposer un livre hors de la bibliothèque déclenche le chargement de ce fichier et ajoute l'objet contenu dans ce fichier à la scène.
3. La corbeille permet de retirer des objets de la scène.
4. Les panneaux d'interaction avec des paramètres. Ils servent à afficher des paramètres scalaires définis dans un fichier décrivant une session de design. Chaque paramètre est associé à un bouton que l'on peut faire tourner sur son axe à l'aide de la gâchette et qui modifie la valeur de ce paramètre. Un bouton de validation permet d'appeler le programme extérieur qui prend en charge ces paramètres. La géométrie nouvellement calculée apparaît alors dans la scène.

Couches logicielles

Tout d'abord, le langage de programmation utilisé pour bâtir notre programme est le C++. Le choix de ce langage s'impose de lui-même : paradigme orienté objet, production de code optimisé pour une application graphique en 3D et interactive,

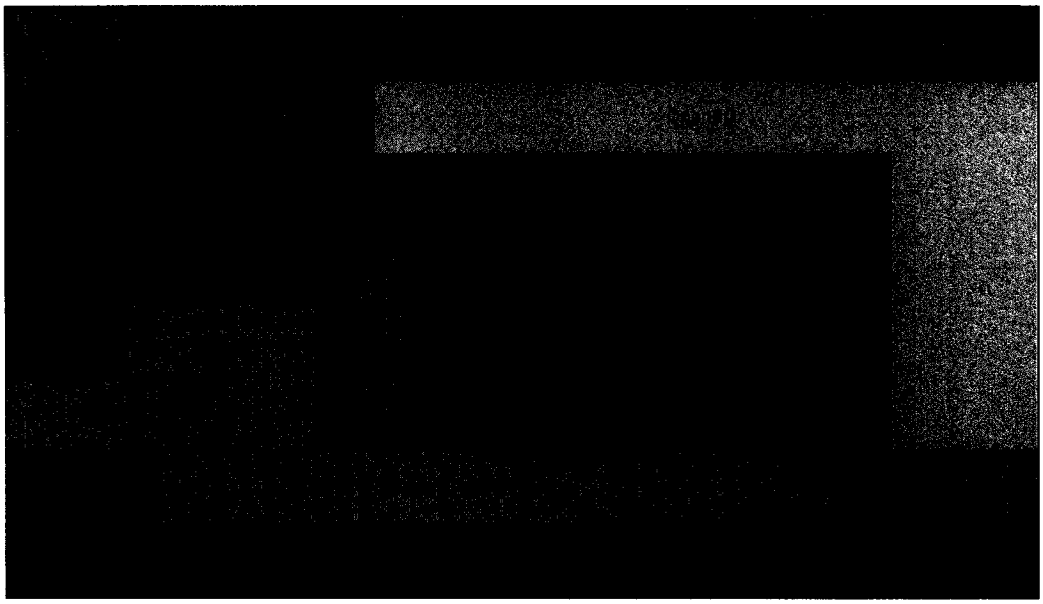


FIGURE 2.2 Couches logicielles de l'ECDI

même langage que les librairies que nous voulons utiliser. La plateforme cible de notre application est Linux pour des raisons de disponibilité et de modularité des outils de développement. Nous pensons que notre système sera facilement portable vers d'autres plateformes mais nous ne nous préoccuperons pas de nous en assurer.

Nous nous appuierons sur différentes librairies pour nous aider à bâtir un prototype de l'ECDI. Deux librairies principales nous serviront respectivement à faire abstraction du matériel et à gérer le graphe de scène, soit VR Juggler[3] (Cruz-Neira et al. 2002) et OpenSG[4]. Nous utiliserons également la librairie ANTLR[5] qui fournit un générateur d'analyseurs syntaxique et lexical et qui nous servira à décrire la grammaire du format de fichier de `VU` et à générer le code source qui le prendra en charge. Voir la figure 2.2 pour une représentation des couches logicielles utilisées pour notre ECDI.

La librairie VR Juggler se présente sous un aspect très modulaire et nous utiliserons

[3] `http://www.vrjuggler.org`
[4] `http://www.opensg.org`
[5] `http://www.antlr.org`

extensivement ses nombreux sous-modules. Le premier (VR Juggler Portable Runtime – VPR) est une couche d'abstraction du système d'exploitation et nous fournira, entre autre, la couche réseau avec une encapsulation orientée objet des interfaces de connexion (*sockets*). Le module JCCL (Juggler Configuration and Control Library) fournit des outils pour la gestion de la configuration et re-configuration en cours d'exécution à travers des fichiers XML. Le module Gadgeteer fournit l'abstraction des périphériques d'entrée. Le module Tweek permet de faire communiquer le système à travers une interface CORBA avec des interfaces extérieures (comme un ordinateur de poche de type Palm) mais nous n'utiliserons pas cette fonctionnalité. Enfin le module central, VR Juggler, gère l'affichage sur différents périphériques, immersifs ou non, stéréoscopiques ou non, en configuration de grappe d'affichage ou de monoposte autonome.

La librairie OpenSG offre une gestion optimisée et automatique du graphe de scène et s'appuie sur la librairie OpenGL. Elle prend en charge l'organisation de la scène en une arborescence de nœuds pouvant contenir de la géométrie, des transformations, la description de l'apparence surfacique, etc. Elle s'occupe du parcours du graphe pour des fins de rendu, de lancer de rayon et autres. Elle fournit également l'importation de géométrie dans différents formats, notamment le VRML.

CHAPITRE 3

MÉTHODOLOGIE ET ARCHITECTURE DU PROTOTYPE

Nous avons, au cours du chapitre précédent, mis en évidence les contraintes, hypothèses et objectifs pour atteindre notre but d'un environnement de collaboration. Nous allons maintenant présenter la méthodologie que nous avons adoptée en détaillant l'architecture du prototype bâti dans le cadre de ce travail pour valider nos hypothèses.

Ce chapitre va d'abord exposer l'architecture du prototype en reprenant les choix énumérés à la section 2.5.1. Nous verrons ainsi comment nous avons implémenté l'immersion, les périphériques d'entrée, la gestion de la cohérence, la gestion de l'attention, la visualisation de surfaces NURBS, la persistance du monde virtuel et la téléconférence. Ensuite nous détaillerons l'implémentation des choix architecturaux (section 2.5.2), soit la topologie réseau et les paradigmes d'interactivité. Enfin, nous donnerons un aperçu des principaux aspects logiciels et patrons de conception impliqués dans l'implémentation de notre prototype.

3.1 Éléments de l'ECDI

3.1.1 Immersion

Notre environnement offre une interface immersive grâce à du matériel et des librairies logicielles adéquats. L'utilisation de VR Juggler comme librairie logicielle d'abstraction de l'affichage nous permet d'afficher automatiquement l'ECDI dans un environnement immersif comme sur poste de travail classique. Nous utilisons l'environnement de la CAVE à l'École polytechnique pour tester et valider notre prototype en immersion.

3.1.2 Périphériques d'entrée

Nous utilisons les périphériques disponibles dans la CAVE de l'École polytechnique. Nous avons à notre disposition des lunettes stéréoscopiques et un bras de repérage. Ces deux périphériques sont munis de dispositifs de repérages Flock of Birds fabriqués par la compagnie Ascension Technologies[1] qui donnent la position et l'orientaion des lunettes et du bras de repérage. Ce dernier, fabriqué par la société Fakespace[2], comporte trois boutons-poussoirs ainsi qu'un bouton analogue à deux directions. L'état de ces périphériques est rendu disponible par un démon VRPN[3], lequel est accédé par la composante `Gadgeteer` de VR Juggler. Cependant, seul l'état des boutons-poussoirs est accessible à cause d'une incompatibilité au niveau des pilotes VRPN.

[1] `http://www.ascension-tech.com`
[2] `http://www.fakespacesystems.com`
[3] `http://www.cs.unc.edu/Research/vrpn/`

3.1.3 Gestion de la cohérence

Rappelons (voir section 2.5.2) que nous nous sommes arrêtés sur une architecture réseau distribuée avec mises à jour en poste-à-poste (*peer-to-peer*). Notre architecture de réseau n'étant pas centralisée, chaque client a sa propre représentation du monde. Il s'agit alors, pour assurer la cohérence de ce monde virtuel chez tous les pairs, d'implanter un processus qui synchronise les différentes représentations entre elles. Nous nous y sommes pris en appliquant deux mécanismes :

1. Dès qu'un client se connecte à un autre, il lui demande la liste de ses objets partagés. Il compare les identificateurs uniques (nous utilisons les GUID[4] de VR Juggler) de ces objets avec ceux des siens et renvoie une requête pour les objets manquants.
2. Dès qu'un objet partagé du monde subit un changement, il sérialise les changements de cet objet aux autres clients connectés.

Pour implémenter ces mécanismes, nous avons donc besoin de paquets pour les requêtes et réponses de début de connexion, de marquer pour chaque objet les changements effectués depuis le dernier envoi et enfin de sérialiser les changements de ces objets.

Gestionnaire d'objets partagés

Le gestionnaire principal du réseau `NetworkManager` délègue la gestion des objets partagés au sous-gestionnaire `SharedObjectsManager`. Celui-ci se charge de garder un registre des objets partagés, de collecter les objets ayant changé, de recevoir des autres pairs des objets partagés et de les comparer avec les siens. Si l'objet reçu

[4] *Globaly Unique IDentifier* en anglais.

correspond à un objet local, les changements sont apportés. Sinon, l'objet est ajouté à la scène et au registre.

Synchronisation initiale

La synchronisation initiale nécessite que les représentations internes des autres pairs (classes `Peer`) aient un état intermédiaire entre « non connecté » et « connecté », soit un état de « nouvelle connexion ». Le `NetworkManager` n'envoie pas de paquets de sérialisation des objets aux pairs qui sont dans cet état afin qu'il ne reçoivent pas des objets qu'ils n'ont pas encore synchronisé.

Ainsi, quand une nouvelle connexion est établie avec un pair, le `SharedObjectsManager` prépare un paquet `SharedObjectsListPacket` et y met la liste des GUID des objets partagés inscrits au registre. Quand un tel paquet arrive au `SharedObjectsManager` de l'autre pair, il compare la liste des GUID avec son propre registre et renvoie une autre liste contenant les GUID des objets qui sont absents de son propre registre. Dès que cette dernière liste est reçue par le premier pair, il sérialise au second chacun des objets demandés en prenant soin de marquer complètement leurs masques de changements afin que tous les champs soit envoyés. Le premier pair peut maintenant changer l'état de son `Peer` (correspondant au second) à « connecté » maintenant qu'il est synchronisé par rapport à lui (voir figure 3.1).

La même procédure a lieu simultanément dans l'autre sens : si le second pair arrivait en « scène » avec des objets partagés qui lui sont propres (par exemple chargés d'un fichier de configuration local), ils seront synchronisés de la même manière.

En fait, il y a toujours au moins un objet qu'il faut synchroniser initialement : la représentation du bras de repérage qui est unique pour chacun. Nous avons décidé de la gérer en temps qu'objet partagé (plutôt que par le `TrackerDataManager`) car

nous pensions avoir besoin de changer dynamiquement l'allure du bras de repérage. Étant donné que le `SharedObjectsManager` relaie automatiquement tous les changements apportés à un objet (et pas seulement sa matrice de transformation) cela nous simplifiait la tâche.

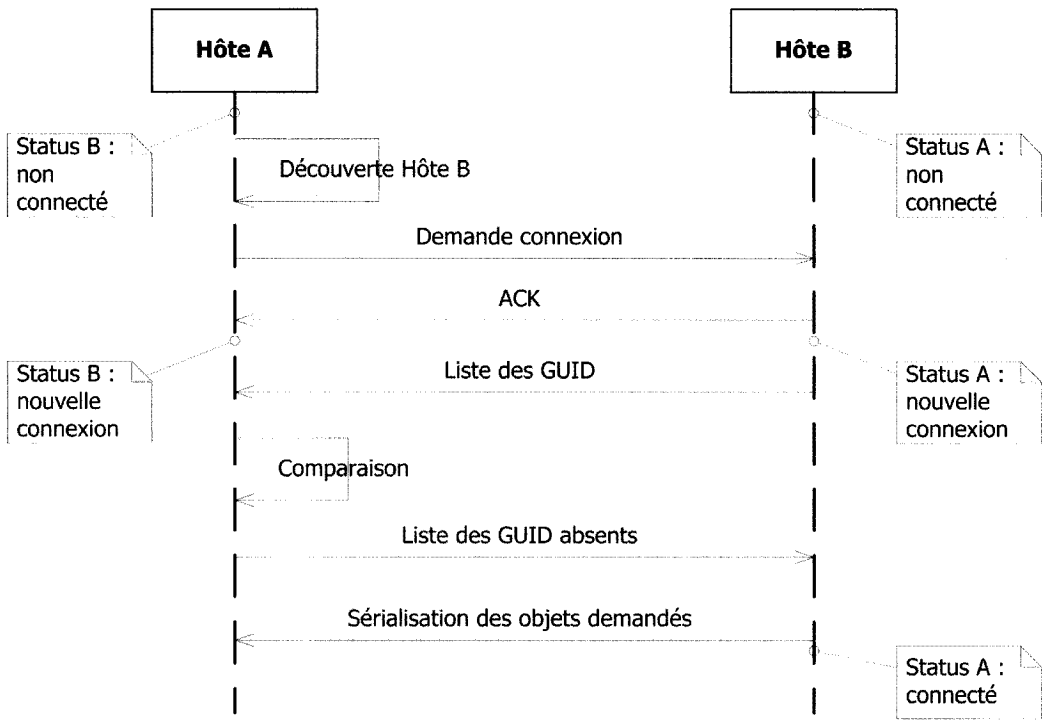


FIGURE 3.1 Synchronisation initiale entre deux hôtes

Masques de changements et sérialisation

Notre mécanisme de partage des objets repose sur l'envoi des changements d'état des objets partagés plutôt que leur état au complet. Pour ce faire nous avons utilisé des masques de bits pour chacun des champs des objets de la scène (classe `GraphicObjectBase`). Quand on modifie le champ d'un objet (sa matrice de transformation, sa couleur, sa taille...), son masque est modifié en conséquent (avec un *ou* bit à bit).

Quand vient le temps d'envoyer aux autres pairs les objets partagés, le `SharedOb-`

`jectsManager` parcourt son registre et demande à chaque objet partagé s'il a changé (c'est-à-dire que son masque est différent de zéro ou, récursivement, que l'un de ses enfants a changé). Dans un tel cas, il demande sa sérialisation dans un paquet `SharedObjectsPacket`.

Cette sérialisation (et dé-sérialisation à l'autre bout) s'effectue avec les méthodes `writeObject()` et `readObject()` qu'implémentent tous les objets graphiques. Ceux-ci, dans leur méthode `writeObject()`, commencent par appeler leur parent afin que tous les champs dérivés, depuis la racine, soient également sérialisés. On teste alors pour chaque champ si le masque de bits montre un changement (avec un *et* bit à bit), auquel cas le champ est sérialisé dans le vecteur d'octet courant. Le premier champ qui est sérialisé en haut de la hiérarchie (donc dans `GraphicObjectBase`) est le masque lui-même.

Ainsi, à l'autre bout, l'objet à sa dé-sérialisation (`readObject()`) lit d'abord le masque. Il teste ainsi chaque champ (dans le même ordre que lors de l'écriture) avec ce masque et, le cas échéant, extrait du vecteur d'octets la valeur du champ. Nous avons écrit des fonctions-patron (*templates*) et surchargé les opérateurs << et >> avec tous les types que nous manipulons (de l'entier jusqu'à l'objet graphique complexe, en passant par le vecteur, la matrice...) pour faciliter et unifier l'écriture de toutes les opérations de sérialisation et dé-sérialisation.

3.1.4 Gestion de l'attention

Nous avons choisi de ne pas implémenter de gestion avancée de l'attention car nous faisons l'hypothèse d'une bande passante suffisante. Cependant, comme le montre la section précédente, une forme d'optimisation du trafic réseau est implémentée à travers les masques de changements.

3.1.5 Visualisation scientifique

La visualisation scientifique est rendue possible par l'utilisation du format de fichier `VU/pirate`. Nous avons utilisé à cette fin le générateur d'analyseurs lexical et syntaxique ANTLR et avons écrit les grammaires du format de fichier `VU/pirate`. La librairie ANTLR génère, à partir de ces grammaires, des fichiers source en C++ qui permettent l'analyse lexicale et syntaxique ainsi que la génération d'un arbre syntaxique abstrait. Cet arbre est ensuite parcouru et chaque entité rencontrée génère la création d'OAV dans la scène. Cependant, nous n'implémentons dans ce prototype que la lecture des entités de surfaces paramétriques (NURBS).

3.1.6 Persistance

Notre architecture ne supporte pas de persistance dynamique du monde. La vie de celui-ci commence avec l'entrée du premier collaborateur et finit quand le dernier quitte l'application. Entre les deux, les collaborateurs peuvent entrer et sortir à loisir et joindre la session en cours grâce aux mécanismes de synchronisation initiale (section 3.1.3). Il serait très possible d'ajouter une persistance statique, en permettant la sauvegarde et le chargement de l'état courant du monde dans un fichier (avec les mécanismes de sérialisation des objets) mais nous ne l'avons pas implémenté dans le prototype actuel.

3.1.7 Téléconférence

La téléconférence n'a pas été intégrée au prototype mais nous comptons sur l'usage d'un programme externe, Skype[5], à cette fin. Cet outil est idoine pour une téléconfé-

[5] `http://www.skype.com`

rence avec quelques participants, ce qui correspond à nos besoins.

3.2 Topologie réseau

3.2.1 Découverte des pairs

Dans notre architecture distribuée, nous n'avons donc pas de serveur central et chaque client est responsable de gérer ses connexions avec les autres. Le premier problème est alors pour un client de découvrir ses autres pairs.

Une approche possible est que les clients—à leur initialisation, périodiquement ou à la demande—envoient des messages en diffusion générale (*broadcast*) sur leur sous-réseau local pour signaler leur présence ou chercher celle des autres. Cependant cette méthode a un inconvénient majeur : les messages en diffusion générale ne sortent pas du sous-réseau, à moins qu'un routeur soit spécifiquement configuré à cette fin (ce qui est rarement compatible avec les mesures de sécurité réseau).

Nous avons ainsi opté pour une solution passant par un fichier de configuration en XML qui énumère les clients participant à une session de collaboration, leur adresse IP, numéro de port et autres détails. Il faut alors que les clients chargent tous localement le même fichier de configuration pour pouvoir participer à la même session.

Configuration et re-configuration dynamique

La lecture du fichier de configuration s'effectue par l'entremise des fonctionnalités de configuration et re-configuration dynamique offertes par VR Juggler. Pour ce faire, il faut faire hériter une classe (ici `NetworkManager`) de l'interface `ConfigElementHandler` et implémenter les méthodes nécessaires. Le noyau de VR Juggler appelle

ces méthodes pour ajouter les éléments de configuration qui nous intéressent.

Un avantage spécifique à utiliser ces fonctionnalités de VR Juggler plutôt que d'implémenter soi-même la lecture des fichiers XML (outre l'économie de développement) est la re-configuration dynamique. Il est ainsi possible, par une interface CORBA et avec un éditeur externe fourni dans la suite VR Juggler, d'accéder dynamiquement, en cours d'exécution, aux éléments de configuration, de les modifier et d'en ajouter. Nous n'avons pas eu besoin de cette fonctionnalité dans notre projet mais cela reste une potentialité très intéressante en vue d'applications futures de notre ECDI.

Ajout des pairs

À chaque fois que le `NetworkManager` (voir figure 3.2) se fait appeler sur son interface de configuration XML avec un nouveau pair, il extrait les informations relatives à ce pair (adresse, port, nom et couleur de l'avatar...) et, selon que l'adresse est locale ou non, crée soit un objet `Peer` soit un objet `PeersAcceptor`. L'objet `Peer` créera à son tour un objet `PeersConnector` pour joindre les autres pairs sur leur `PeersAcceptor`. Les deux derniers objets s'inspirent du patron *Connector-Acceptor* Schmidt (2000) qui nous permet de découpler la gestion de la connexion de celle de la transmission des données sur un canal établi.

Notons que les classes `Peer`, `PeersAcceptor` et `PeersConnector` dérivent toutes trois de la classe `ThreadedEnginePart` qui ajoute la fonctionalité de s'exécuter sur un autre fil d'exécution, permettant ainsi d'éviter les complications liées à des interfaces de connexion non bloquantes.

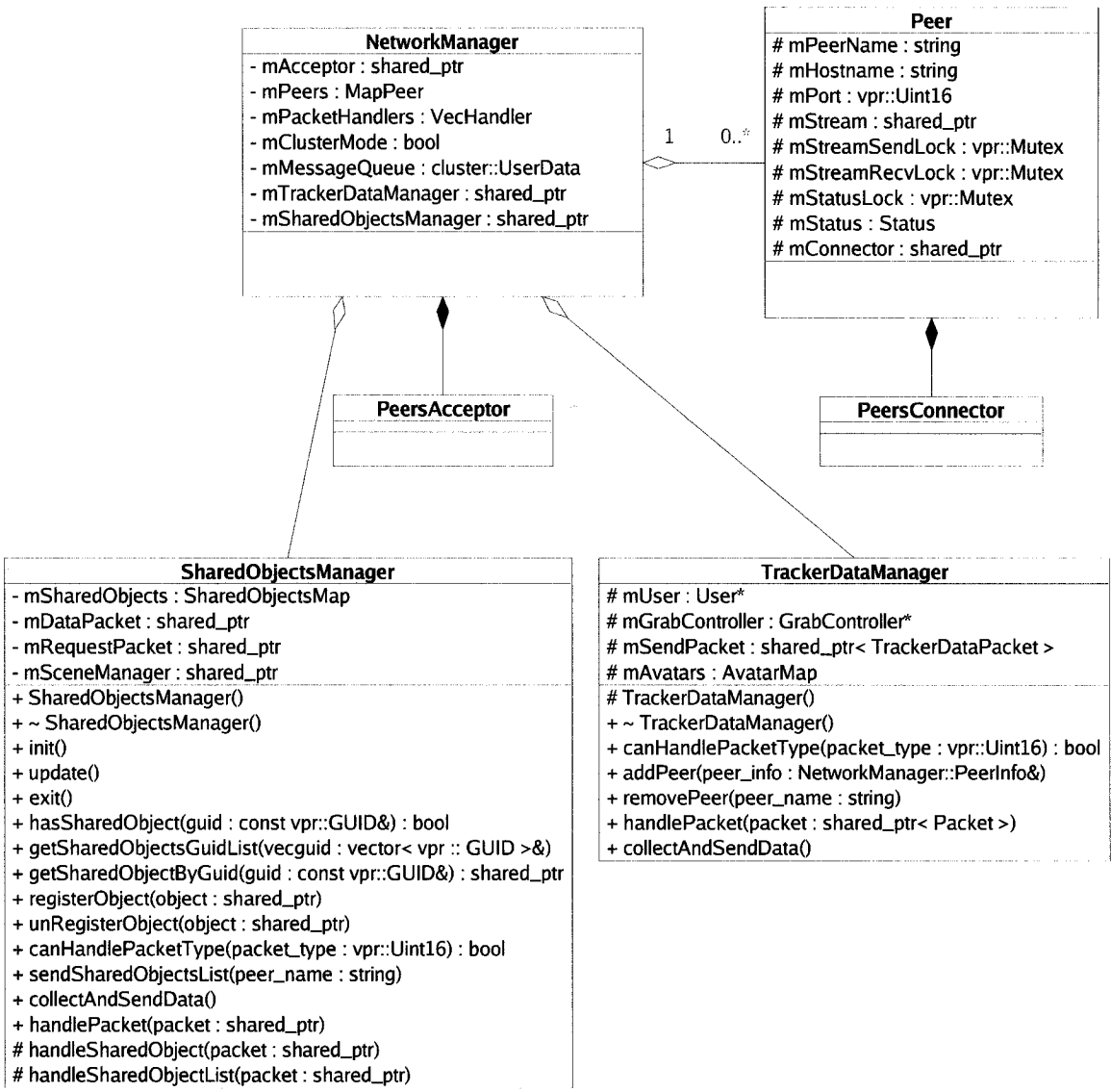


FIGURE 3.2 Architecture réseau

3.2.2 Connexion et déconnexion

L'objet `Peer` s'initialise dans un état non connecté et tente (à travers son `PeersConnector`) de se connecter à l'adresse et au port spécifié. L'objet `PeersAcceptor` lui se met à l'écoute des demandes de connexion provenant des autres pairs. Quand une connexion est établie, l'objet `Peer` correspondant est mis dans un état connecté et il ajoute à la scène un objet de type `ObjAvatar` pour représenter l'usager distant.

Pendant la durée de vie du canal, les objets `Peer` sont responsables de recevoir les paquets sur ce canal et de les relayer au `NetworkManager` afin qu'il les traite adéquatement. Cette réception des paquets se fait sur une interface de connexion bloquante dans un fil d'exécution parallèle. Si un problème arrive sur cette connexion (le plus souvent qu'elle soit coupée), le `Peer` en question notifie le `NetworkManager`, demande le retrait de l'avatar de la scène et se met dans un état non connecté.

3.2.3 Transmission de données

Types de paquets

Les données sont toutes encapsulées dans une hiérarchie de paquets. Un paquet est représenté par un objet de type `Packet` et contient un en-tête (objet de type `Header`) qui spécifie le type du paquet ainsi que la taille en octets des données qu'il contient. Les données dans le paquet sont représentées par un vecteur STL d'octets.

Les paquets ont, de plus, la fonctionnalité de pouvoir s'envoyer ou se recevoir eux-mêmes : pour l'envoyer, il suffit d'appeler le paquet sur sa méthode `send()` avec en paramètre le canal établi. Pour recevoir un paquet, la méthode statique `Packet::getNextPacket()` se charge de recevoir l'en-tête, d'extraire le type et la taille du paquet, de créer un nouveau paquet (selon le type avec le générateur de classes d'objets `PacketFactory`), de recevoir les données dans le nouveau paquet et de retourner un pointeur partagé sur ce paquet.

Les paquets sont de plusieurs types : `ConnectionAck` et `ConnectionReq` pour gérer les connexions, `TrackerDataPacket` pour transmettre la position de l'avatar, `SharedObjectsListPacket` pour gérer la synchronisation initiale d'un nouveau pair à une session existante, `SharedObjectsPacket` pour sérialiser les objets partagés de la scène et enfin `RemoveObjectPacket` pour signifier le retrait d'un objet de la scène.

Prise en charge des paquets entrants

Le `NetworkManager` tient un registre des objets pouvant prendre en charge les paquets et qui doivent ainsi présenter une interface `PacketHandler`. Quand le `NetworkManager` doit distribuer un paquet reçu, il demande à chaque `PacketHandler` inscrit s'il prend en charge le type du paquet reçu, auquel cas le pointeur partagé du paquet lui est transmis.

Transmission de l'avatar

Nous avons vu que l'avatar des autres pairs est créé à la connexion par le `Peer`. À ce moment, le `NetworkManager` active un sous-gestionnaire, le `TrackerDataManager` qui se charge de récolter à chaque `update()` la position de la tête et de l'envoyer dans un paquet `TrackerDataPacket`. Ce paquet est reçu aussi par le `TrackerDataManager` qui en extrait la matrice de transformation de la tête pour l'appliquer à l'`ObjAvatar` correspondant au pair du paquet reçu.

3.2.4 Grappe d'affichage

L'infrastructure réseau que nous avons décrit jusqu'à maintenant ne vaudrait que dans le cas où notre ECDI est exécuté en monoposte. En effet, pour que l'application marche aussi sur la CAVE, nous tombons sur des complications qu'il faut tenir en compte. La CAVE à laquelle nous avons accès (à l'École polytechnique) est en fait une grappe d'affichage, c'est-à-dire que chaque mur/écran de la CAVE est géré par un ordinateur différent.

Synchronisation de données au sein de la grappe

La librairie VR Juggler nous simplifie déjà grandement la tâche en offrant une abstraction de l'affichage et en gérant automatiquement le cas d'une grappe d'affichage : l'application est lancée simultanément sur chacun des nœuds de la grappe et VR Juggler se charge de synchroniser les affichages et les données des dispositifs de repérage entre ceux-ci. Tout ce qu'il y a à faire est de fournir un fichier de configuration décrivant les adresses IP des nœuds formant la grappe et spécifiant quels nœuds sont responsables de la synchronisation et de la collecte des données des dispositifs de repérage. Cependant, pour notre ECDI, il faut pouvoir attribuer la gestion des événements réseau à un seul nœud de la grappe d'affichage et propager ensuite ces informations aux autres nœuds.

VR Juggler offre à cette fin un mécanisme pour synchroniser des données externes dont un seul nœud est responsable. Il faut tout d'abord que les données à synchroniser soient dans une structure de données supportant l'interface `vpr::Serializable` qui requiert que cette structure puisse écrire et lire son contenu sur un vecteur STL d'octets. Ensuite il faut encapsuler cette structure de données dans un objet de type `cluster::UserData<>`. Enfin, il faut indiquer, dans un fichier de configuration, le nœud qui sera responsable de mettre à jour ces données et leur associer un identificateur unique (VR Juggler utilise des GUID à 128 bits). À l'initialisation de cette structure, on fournit le même GUID et VR Juggler se charge ensuite de le synchroniser automatiquement. Une méthode de `cluster::UserData` permet de savoir si nous sommes sur le nœud responsable et s'il faut ainsi mettre à jour les données.

Queue de messages

Dans notre projet nous avons ainsi rencontré la nécessité de centraliser la gestion des événements réseau sur un seul nœud de la grappe. Nous avons à cette fin implémenté un système de queue de messages : le nœud responsable encapsule chaque événement réseau dans un message, lequel est mis en file dans la queue. Cette queue ainsi que les messages qui la composent sont sérialisables et VR Juggler se charge automatiquement de la répliquer chez les autres nœuds. Lorsque synchronisée, on peut dépiler chaque message de la queue et le traiter selon sa nature.

Nous avons eu besoin de trois types de messages : les messages de commande, les messages-paquets et les messages de synchronisation des GUID. Les messages de commande servent à propager les informations relatives à l'ajout de nouveaux pairs et de leur état (connectés ou non). Les messages-paquets encapsulent les paquets présentés précédemment. Les messages-GUID servent à ce que les GUID générés (aléatoirement) soient les mêmes à travers la grappe. Nous utilisons des GUID pour identifier de façon unique chaque objet partagé de la scène.

3.3 Interface et interactivité

Afin que l'utilisateur puisse interagir avec les objets de la scène, nous les munissons de décorateurs (voir section 3.4.4). Nous avons implémenté deux décorateurs pour notre prototype, `Grabable` pour la manipulation générique des objets et son dérivé, `EditableNurbs`, pour manipuler spécifiquement les surfaces NURBS.

Lorsqu'on décore un objet de type `GraphicObjectBase` avec un décorateur `Grabable`, ce dernier ajoute à l'objet deux comportements : (1) un test d'intersection avec la représentation du bras de repérage et (2) une mise à jour de « l'état d'interaction » (encapsulé dans un objet de type `GrabState`) en fonction de l'état de la gâchette du

bras de repérage et du résultat du test d'intersection.

3.3.1 Machine à état des interactions

La classe `GrabState` est une machine à quatre états discrets : libre, sélectionné, pris et sous-objet. Les critères de transition sont indiqués à la figure 3.3.

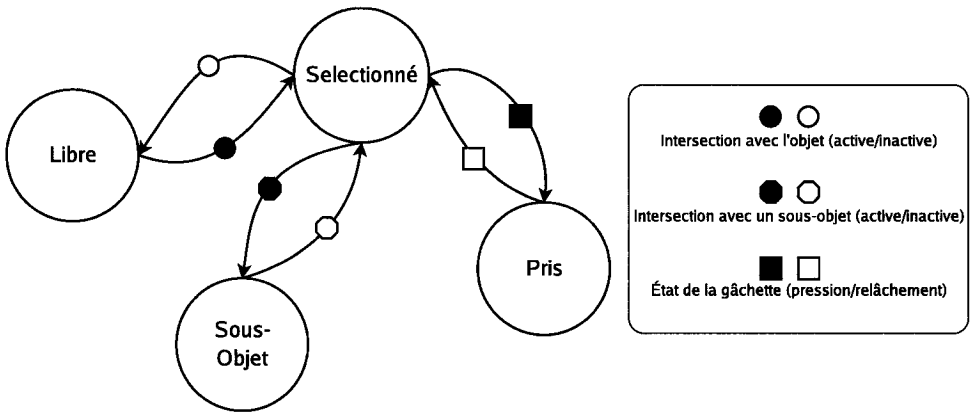


FIGURE 3.3 Transitions de la machine à état des interactions

L'état « sélectionné » est activé par un critère de proximité entre l'objet et le bras de repérage (qui agit comme un curseur en 3D). Le critère peut être soit l'intersection avec le volume englobant de l'objet (*bounding box*) soit l'intersection avec la géométrie de l'objet. Pour des raisons d'efficacité, lorsque le deuxième critère est choisi, le premier est quand même testé avant car il est beaucoup moins coûteux, surtout avec une géométrie complexe. En effet, le test d'intersection se fait par trois lancer de rayon de dimensions de l'objet représentant le bras de repérage (ici, un cylindre). La librairie OpenSG fournit le lancer de rayon mais il n'est pas optimisé pour de la géométrie complexe car chaque triangle est testé.

L'état « pris » est activé quand l'objet est sélectionné et tant que la gâchette reste pressée. L'état « sous-objet » est activé quand l'objet est sélectionné et qu'un de ses sous-objets est à son tour sélectionné. Cet état est utile pour gérer les situations

d'objets composites (par exemple pouvoir déplacer le point de contrôle d'une NURBS sans que celle-ci ne soit affectée par le mouvement).

Lorsque la machine à états `GrabState` voit son état changer, elle prévient (à travers son interface `Subject`) l'objet `Grabable` qui lui est associé (sur son interface `Observer`). Ce mécanisme générique permet à n'importe quel autre objet implémentant l'interface `Observer` d'être à l'écoute et de réagir aux changements d'état d'un objet `Grabable` donné.

3.3.2 Stratégies d'interaction

Le comportement que suit le décorateur `Grabable` lorsqu'il change d'état n'est pas figé : nous faisons encore appel à un patron *Strategy* pour nous offrir une meilleure flexibilité. Les classes `HighlightStrategy` et `GrabStrategy` nous permettent respectivement de choisir la représentation visuelle de l'état d'interaction et le comportement de l'objet pendant le « glisser » (i.e. quand il est dans un état « pris »). La figure 3.4 illustre les classes en jeu autour de `Grabable`. En ce qui concerne `GrabStrategy` nous n'en avons implémenté qu'une seule : `AttachWandGrabStrategy` qui fait en sorte que l'objet suive les transformations du bras de repérage. Il est possible aussi de spécifier des contraintes : pas de translation, rotation uniquement autour de certains axes...

Pour `HighlightStrategy` nous avons, par contre, implémenté plusieurs stratégies :

- `WiredCubeHLStrategy` : Affiche le volume englobant (aligné sur les axes) autour de l'objet lorsque non libre. Un code de couleur spécifie l'état : vert pour sélectionné, rouge pour pris et bleu pour sous-objet.
- `BrillanceHighlightStrategy` : Rend l'objet brillant lorsque non libre.
- `TransparentHLStrategy` : Rend l'objet transparent lorsque non libre.
- `ScaleHighlightStrategy` : Agrandit l'objet d'un certain facteur lorsque non libre.

– `EmptyHighlightStrategy` : C'est l'opérateur neutre, aucun effet quel que soit l'état.

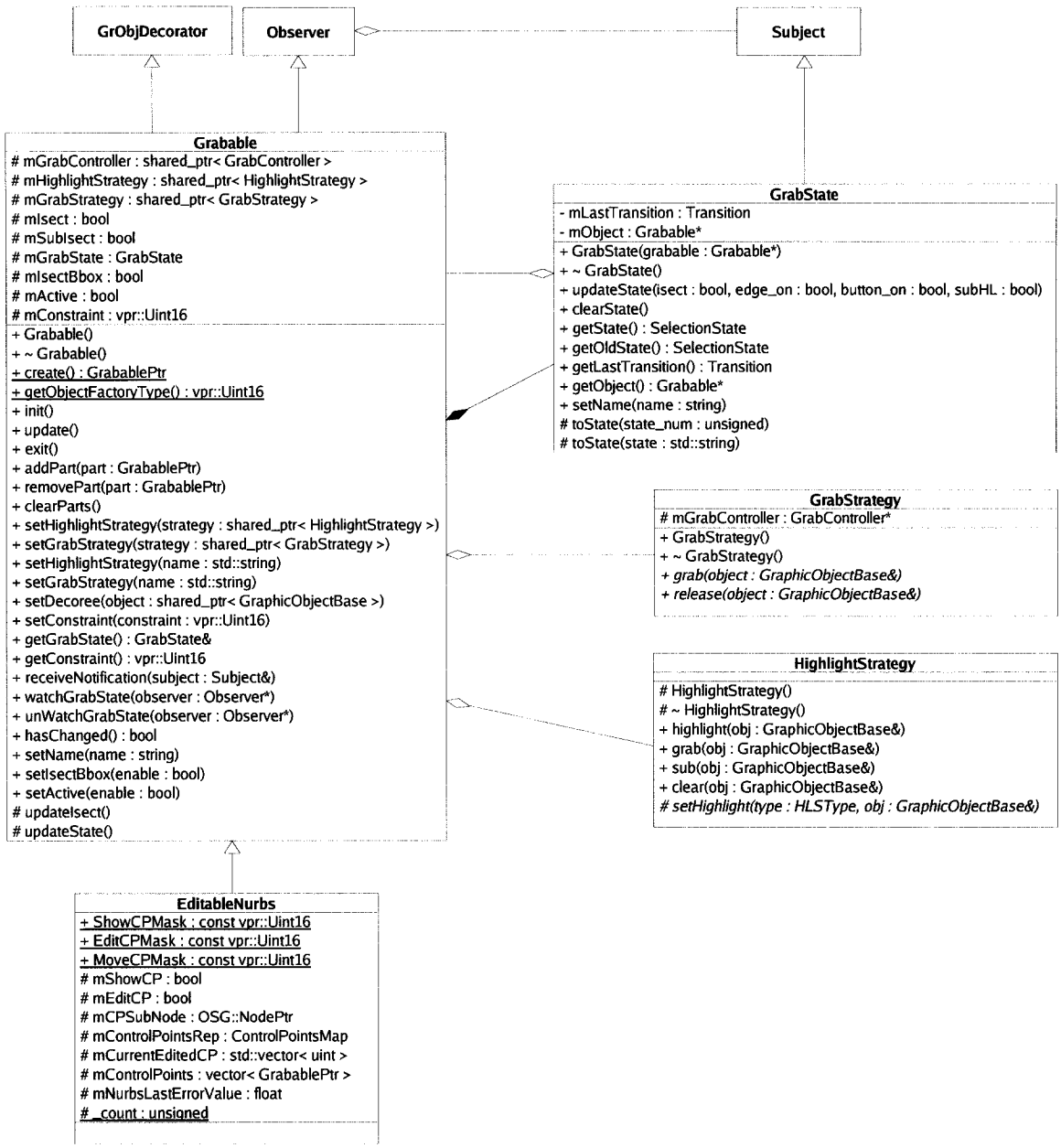


FIGURE 3.4 Décorateurs et stratégies d'interactivité avec les objets

3.3.3 Interaction avec une NURBS

Pour l'interaction avec une surface NURBS nous avons créé un décorateur spécifique : `EditableNurbs`. Celui-ci, à son initialisation, cherche les points de contrôle de l'objet `ObjNurbs` qu'il décore et crée des représentations graphiques pour ces points (cubes ou sphères par exemple), chacune décorée d'un décorateur `Grabable`.

Quand l'objet est sélectionné, ce décorateur fait afficher les points de contrôle et les cache lorsque de nouveau dans l'état libre. Quand un point de contrôle est dans l'état « pris » il fait basculer l'affichage de la NURBS en fil de fer (pour bien voir les changements à la surface) et fait correspondre la position de la représentation du point de contrôle à celle même du point de contrôle associé. Ainsi le décorateur `EditableNurbs` est à la fois à l'écoute de son propre état d'interaction mais aussi de ceux de ses sous-objets (les représentations des points de contrôle).

3.3.4 Navigation

Systèmes de coordonnées

Pour nous repérer dans le monde virtuel, nous devons manipuler plusieurs systèmes de coordonnées :

- Les coordonnées *réelles* sont celles du monde avant la transformation de navigation. Elles correspondent aussi à celles des dispositifs de repérage.
- Les coordonnées *virtuelles* sont celles du monde après la transformation de navigation. Ce sont les coordonnées que nous utiliserons pour manipuler les objets de la scène.
- Les coordonnées *locales* sont celles de l'objet que l'on manipule dans un contexte donné.

Les deux premiers systèmes de coordonnées, réelles et virtuelles, nous sont nécessaires pour notre environnement immersif car nous n'avons, contrairement à une application 3D classique, pas le contrôle sur le point de vue ni sur la projection, ceux-ci étant déterminés par VR Juggler en fonction du dispositif d'affichage et de la position de la tête de l'usager. Si nous ne pouvons pas déplacer la « caméra » dans le monde, nous devons déplacer le monde autour d'elle. Nous pouvons aussi faire l'analogie avec un tapis volant : la CAVE constitue une cabine du monde réel qui se déplace dans le monde virtuel.

Implémentation

Nous avons eu recours à plusieurs classes pour implémenter la navigation (voir figure 3.5). La classe `User` modélise l'utilisateur avec ses dispositifs de repérage (tête, *wand* et boutons) ainsi que son « tapis volant ». Ce dernier est modélisé par la classe `ViewPlatform` qui met à jour la matrice de transformation de la scène à travers une stratégie de navigation. Cette stratégie, basée sur le patron *Strategy* (Gamma et al. 1995, page 315), permet d'adopter dynamiquement différents paradigmes de navigation. Nous en avons implémenté un seul pour notre prototype, dans la classe `Walk` : l'utilisateur pointe une direction avec son bras de repérage, le bouton 1 le fait avancer dans cette direction. Pour tourner, il suffit que l'angle entre la direction du bras de repérage et celle du regard soit proche de l'angle droit.

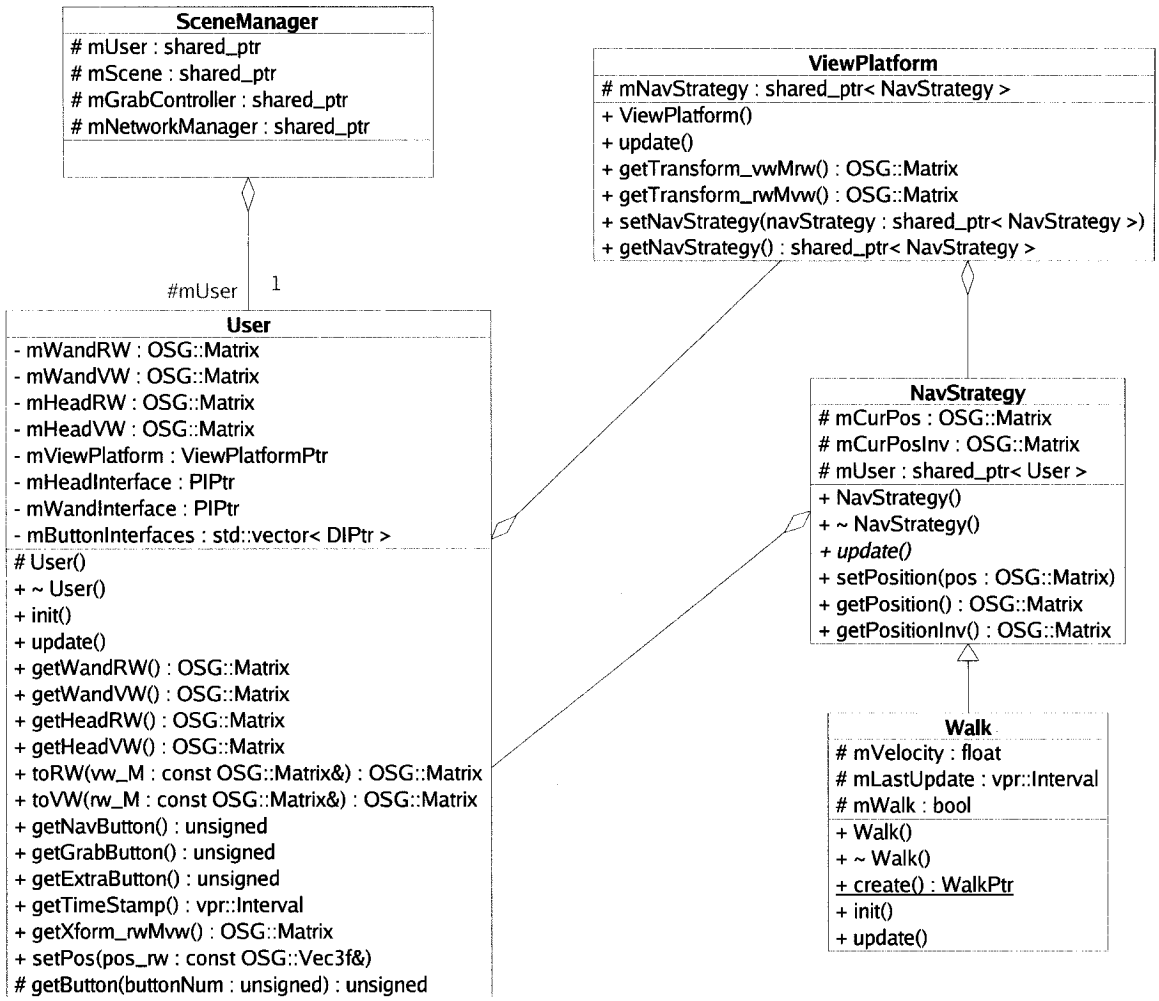


FIGURE 3.5 Stratégies de navigation

3.4 Aperçu des autres aspects logiciels

3.4.1 Environnement de programmation et librairies

Deux générations du compilateur ont été utilisées pour notre programme, soit GCC 3.X et 4.X. Nous nous sommes assurés que notre programme ainsi que ses dépendances puissent être compilés par ces deux versions. L'environnement intégré de programmation utilisé a été KDevelop sous le gestionnaire de fenêtres KDE[6]. Pour la gestion des configurations et des versions nous avons utilisé CVS[7]. Pour la documentation automatique nous avons utilisé Doxygen[8].

Pour les librairies, comme mentionné au chapitre 2, nous avons utilisé VR Juggler (version 2.0.1), OpenSG (version 1.6.0) et ANTLR (version 2.7.5). De plus, la librairie `boost`[9] (version 1.33) nous a beaucoup servi notamment pour les pointeurs partagés (`boost::shared_ptr`), le formatage typé (`boost::format`), les conteneurs hétérogènes (`boost::variant`) et la manipulation de fichiers et chemins (`boost::filesystem`).

3.4.2 Structure générale du programme

Notre prototype doit effectuer plusieurs tâches différentes. C'est une application graphique 3D interactive, un client dans un réseau poste-à-poste et un monde virtuel peuplé d'objets hiérarchisés. Nous allons découper les fonctionnalités du programme selon différents thèmes où nous exposerons les classes (en C++) utilisées pour représenter les concepts ou comportements en question.

---

[6] `http://www.kde.org`
[7] `http://www.nongnu.org/cvs`
[8] `http://www.doxygen.org`
[9] `http://www.boost.org`

Moteur et trame

Le premier concept utilisé pour modéliser le flux de contrôle dans notre programme est celui de « moteur ». En nous inspirant de ce qui se fait, par exemple, dans un jeu vidéo on modélise le flux de contrôle général de l'application autour de la trame (*frame* en anglais). Le moteur est l'agrégation de plusieurs composantes ayant chacune un but spécifique mais dont le cycle de vie s'inscrit dans celui de la trame. La trame est l'unité de temps (et de contrôle) entre deux rafraîchissements successifs de l'affichage ; c'est une séquence d'opérations qui ressemble le plus souvent à ce qui suit :

1. **scrutation** : On collecte les valeurs courantes des périphériques d'entrée.
2. **mise à jour** : On calcule le nouvel état de chacune des composantes du monde en fonction des entrées, de leur état courant, du temps écoulé, du hasard, etc.
3. **rendu** : On affiche la scène ainsi recalculée.

Cette séquence est répétée en boucle jusqu'à ce qu'une condition de sortie soit rencontrée (le plus souvent que l'usager demande de quitter le programme). On cherche, dans une application interactive, à ce que le nombre de trames par seconde soit suffisamment élevé (généralement 30 et plus) pour que la fluidité de l'affichage et le temps de réponse des commandes n'en souffrent pas trop.

Cycle de vie des composantes

Chaque composante du moteur a un cycle de vie qui découle de la trame. Nous avons modélisé une composante par la classe `EnginePart` (voir figure 3.6). La méthode `init()` est appelée une fois après la création de l'objet et sert à l'initialisation de son état et de ses dépendances. La méthode `update()` est appelée pour la mise à jour de l'objet. La méthode `exit()` est appelée avant que l'objet ne soit détruit pour

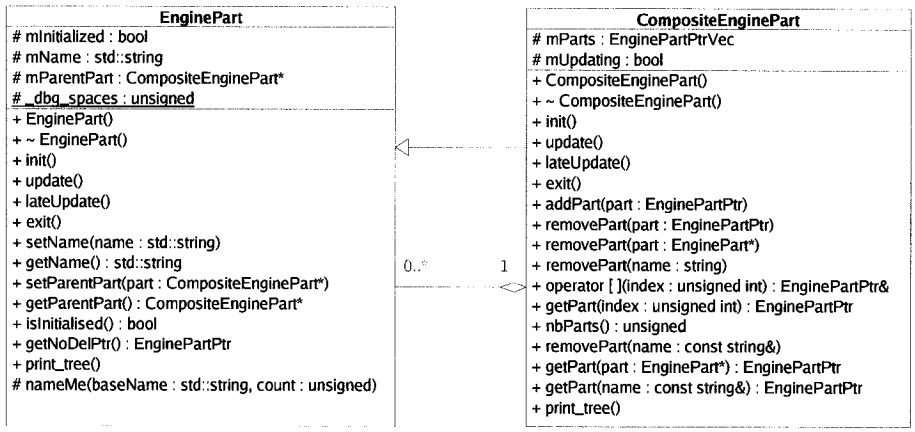


FIGURE 3.6 EnginePart et CompositeEnginePart

qu'il libère les ressources acquises et prévienne ses dépendances de son départ. Nous verrons l'utilité de la méthode `lateUpdate()` plus loin, lorsque nous examinerons l'ordonnancement des événements réseau.

Comme le montre aussi la figure 3.6, une autre classe sert de brique de base à nos composantes, la classe `CompositeEnginePart` qui, s'inspirant du patron *Composite* (Gamma et al. 1995, page 163), permet de hiérarchiser les composantes entre elles : une composante composite se charge d'initialiser, mettre à jour et terminer ses composantes filles.

Création et destruction d'une composante

Nous avons utilisé deux moyens pour gérer à la fois la création dynamique et la (responsabilité de la) destruction des composantes. Le premier, pour les objets dont une unique instance est requise à travers tout le programme, fait appel au patron *Singleton* (Gamma et al. 1995, page 127) et permet la création de l'objet dès qu'il est référencé la première fois et sa destruction (comme tout objet statique global) à la toute fin d'exécution du programme.

Le second, pour les objets partagés, fait appel à un pointeur « intelligent » muni d'un compteur de référence (`boost::shared_ptr`). À la création de l'objet, le compteur du pointeur qui l'encapsule est initialisé à 1. À chaque copie du pointeur, le compteur est incrémenté. À chaque destruction du pointeur le compteur est décrémenté ; lorsqu'il atteint zéro (signifiant que plus personne ne le référence) l'objet qu'il contient est détruit. Cette méthode permet très efficacement de découpler la responsabilité de création de celle de destruction des objets.

Classes d'objets

Nous avons eu besoin d'un autre mécanisme pour la création dynamique d'objets lorsqu'il est nécessaire de créer une instance d'un objet d'une certaine famille de classes sans qu'on sache à l'avance la classe concrète de l'objet voulu. Nous nous sommes inspiré du patron *Factory Method* (Gamma et al. 1995, page 107) et l'avons adapté à nos besoins.

Tous les objets de la scène dérivent d'une classe abstraite `GraphicObjectBase` ; une classe `ObjFactory` implémente la création de cette famille d'objets : chaque classe d'objet concrète (par exemple `ObjCube`) s'enregistre au démarrage du programme auprès de l'`ObjFactory` avec un numéro d'identification unique à cette classe. Lorsqu'on veut créer un objet de cette classe, on appelle la méthode `ObjFactory::createSharedObject()` en lui donnant comme paramètre l'identificateur de cette classe ; il nous est retourné alors un pointeur partagé vers l'instance nouvellement créée. Ce mécanisme nous sert notamment pour la sérialisation des objets que nous verrons plus loin.

3.4.3 Gestionnaires

Il y a dans notre application plusieurs gestionnaires qui gèrent le flux de contrôle entre les autres composantes. Premièrement, au niveau de l'application, la classe `MyOpenSGApp` dérive de la classe `vrj::OpenSGApp` fournie par VrJuggler pour créer une application OpenSG et implémente directement la séquence de trame vue plus haut. La classe `MyOpenSGApp` crée et contrôle deux gestionnaires principaux qui gèrent respectivement la scène et le réseau, soit `SceneManager` et `NetworkManager`. Il sont également tous deux des *Singletons* (voir la figure 3.7 pour une vue d'ensemble des gestionnaires).

La classe `SceneManager` gère l'ajout et la suppression d'objets à la scène et agit comme façade (Gamma et al. 1995, page 185) par rapport à la scène et la navigation (transformation de coordonnées virtuelles/réelles).

La classe `NetworkManager` a la responsabilité d'établir le réseau virtuel entre les autres pairs, de propager les connexions et déconnexions des pairs, de distribuer correctement les paquets entrants et sortants et enfin de gérer la propagation de tous ces événements réseau au sous-réseau de la grappe d'affichage dans le cas où le programme est exécuté dans la CAVE. Le `NetworkManager` délègue la gestion des certains paquets à deux sous-gestionnaires, `TrackerDataManager` et `SharedObjectsManager` pour gérer respectivement les données en provenance des dispositifs de repérage et les changements d'état des objets partagés.

3.4.4 Objets du monde virtuel

Comme mentionné précédemment, tous les objets de la scène dérivent d'une classe de base abstraite, `GraphicObjectBase`. Cependant, nous avons voulu implémenter la possibilité d'ajouter dynamiquement du comportement aux objets et, pour ce faire,

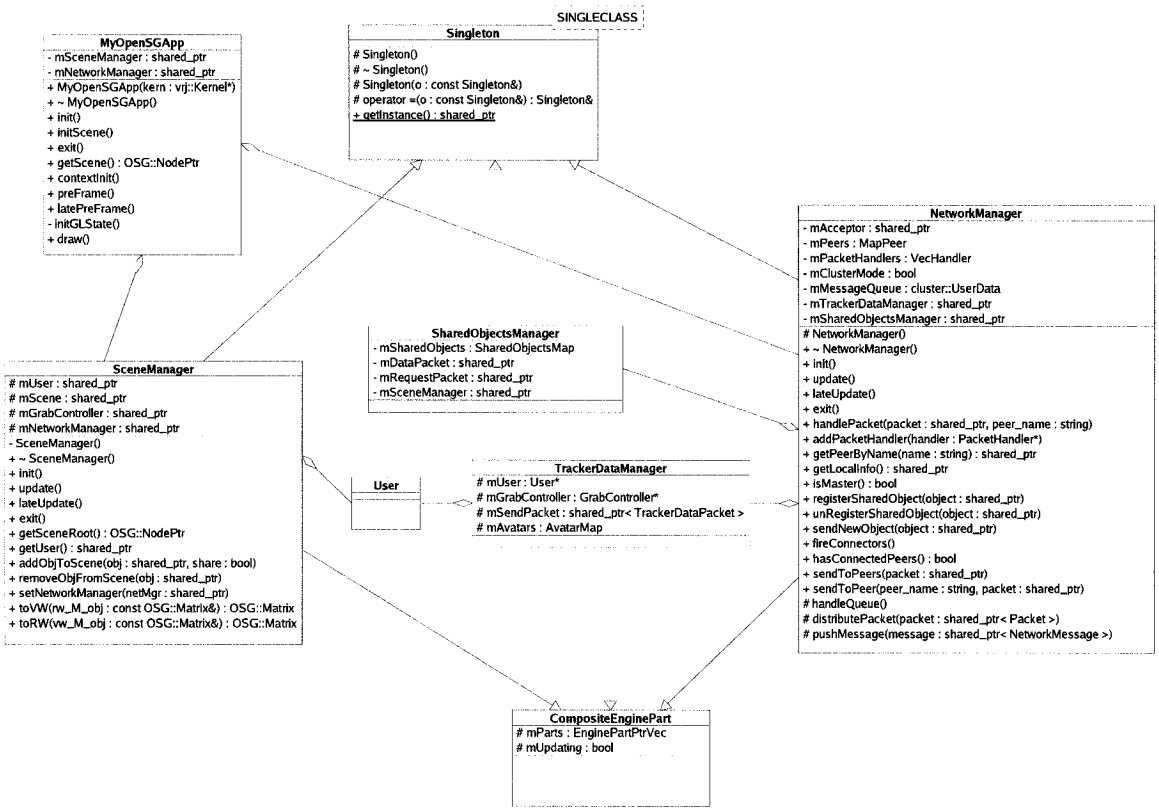


FIGURE 3.7 Gestionnaires principaux

nous avons utilisé le patron *Decorator* (Gamma et al. 1995, page 175).

De la classe `GraphicObjectBase` nous avons dérivé deux autres classes abstraites (voir figure 3.8) `GraphicObject` et `GrObjDecorator`. Le décorateur se comporte comme un `GraphicObjectBase` en relayant les requêtes à l'objet qu'il encapsule et en ajoutant le comportement voulu. Par exemple, nous avons défini un type concret de décorateur, `Grabable`, qui permet d'ajouter à un objet la possibilité d'interaction.

La classe `GraphicObjet` est véritablement la classe de base pour tous les objets graphiques. Elle implémente les caractéristiques génériques que peut posséder chaque objet graphique de la scène : matrice de transformation, matrice de mise à l'échelle, matériaux (couleur, texture et transparence) et graphe de scène local.

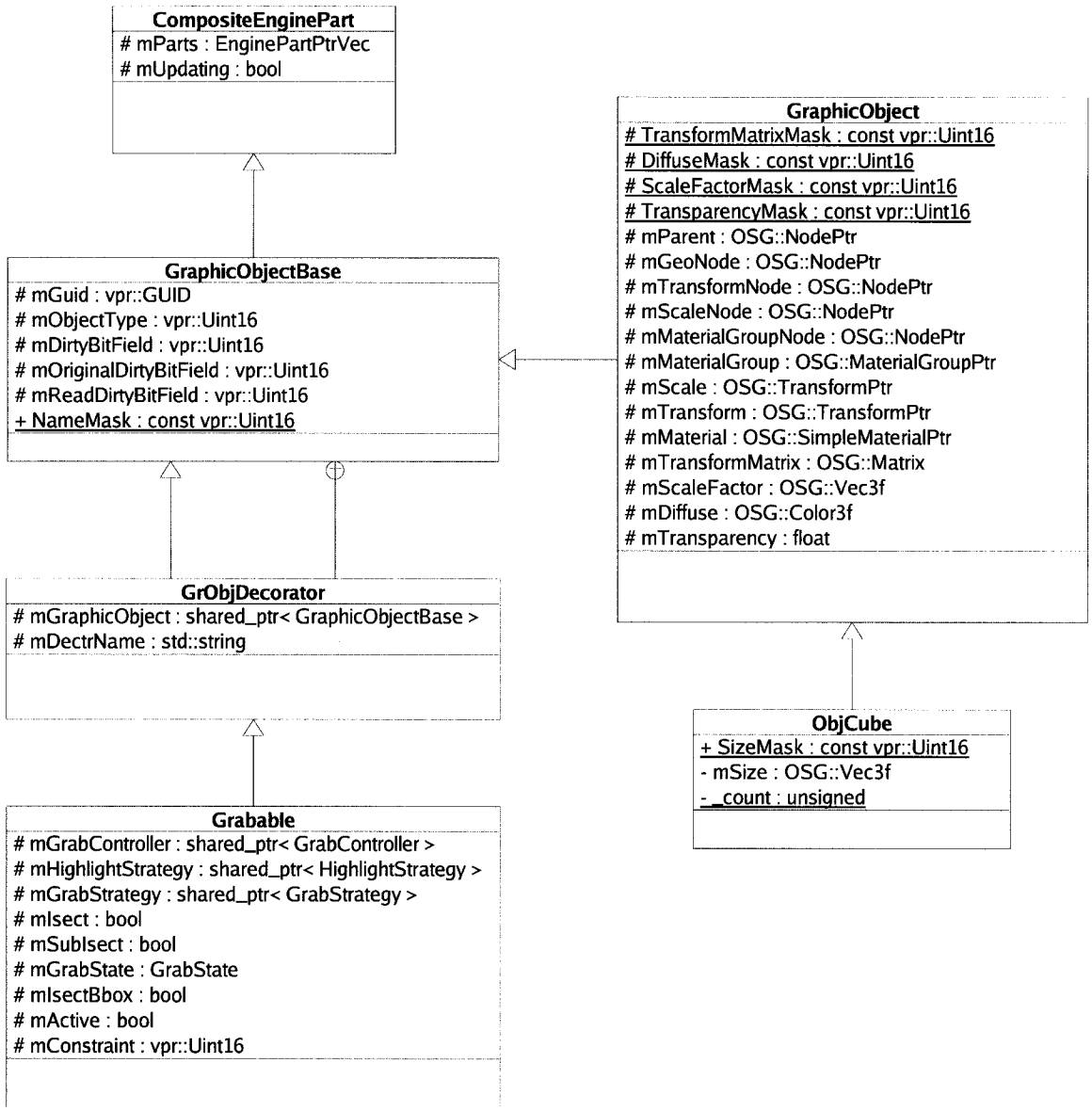


FIGURE 3.8 Objets graphiques et décorateurs

3.4.5 Communication entre les composantes

Il arrive que des composantes doivent communiquer à d'autres leurs changements d'état en parallèle avec le flux de contrôle général. Nous avons, à cette fin, utilisé un patron *Observer* (Gamma et al. 1995, page 293) qui met en interaction un sujet et un observateur. L'observateur s'enregistre auprès des sujets qui l'intéressent. Le sujet, à l'appel de sa méthode `notify()`, appelle tous ses observateurs sur leur méthode `receiveNotification()` avec en paramètre une référence vers le sujet appelant. Dans notre programme, il suffit de faire dériver les objets des classes `Subject` et `Observer` (voir figure 3.9) pour leur donner respectivement la fonctionnalité de sujet ou d'observateur.

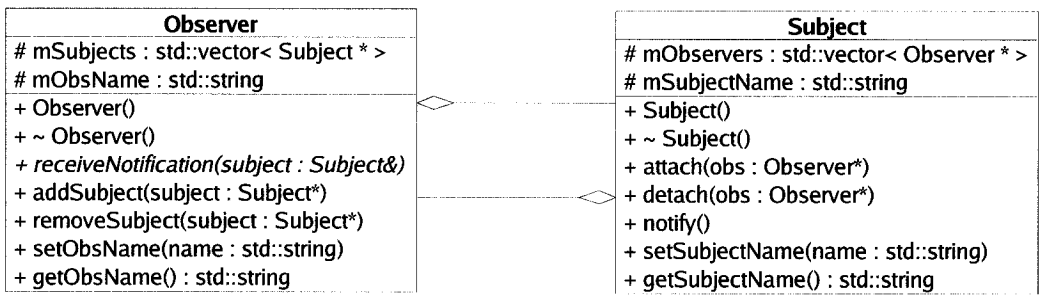


FIGURE 3.9 Sujet et observateur

3.4.6 Flux de contrôle

Nous pouvons maintenant exposer le flux de contrôle général (voir figure 3.10) de notre programme. Rappelons-nous que toutes les composantes dérivent de la classe `EnginePart` et peuvent être composites si elles dérivent de `CompositeEnginePart`. Au sommet de la hiérarchie, il y a l'instance unique de la classe `MyOpenSGApp` qui donne le contrôle aux gestionnaires principaux `SceneManager` et `NetworkManager`.

Le `SceneManager` gère la scène (objet de type `Scene`), l'objet usager (`User`) et un objet façade qui sert pour les interactions (`GrabController`). La scène gère tous les

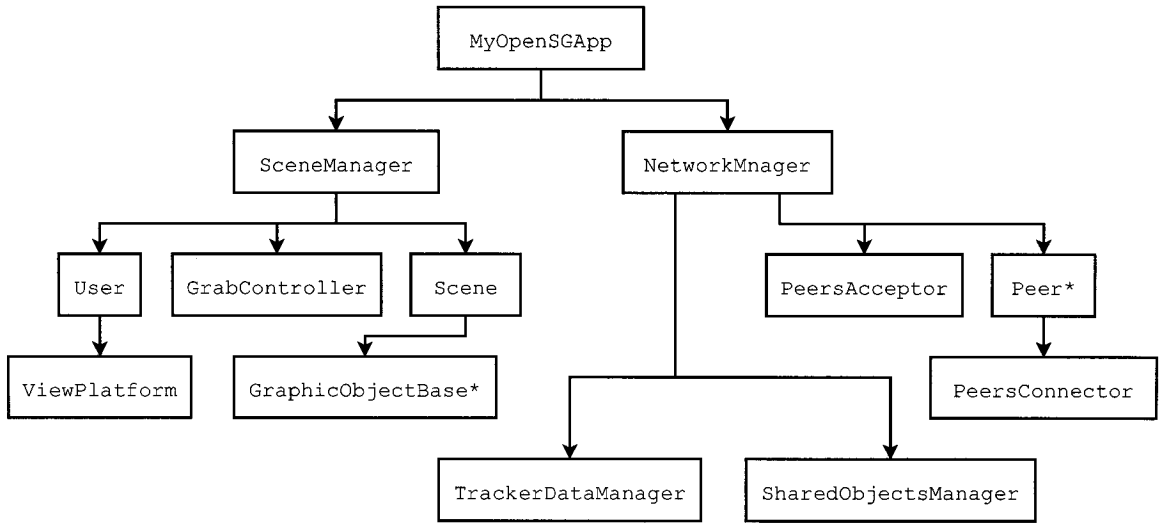


FIGURE 3.10 Flux de contrôle général

objets graphiques (décorés ou non) de type `GraphicObjectBase`.

Le `NetworkManager` gère les deux sous-gestionnaires (`TrackerDataManager` et `SharedObjectsManager`), les pairs connectés (`Peer`) et `PeersAcceptor`, l'accepteur de connexions. Les représentations des pairs `Peer` gèrent chacun un connecteur `PeersConnector` qui sert au moment de la connexion.

3.4.7 Graphe de scène

La librairie OpenSG nous offre une fonctionnalité de graphe de scène évolué qui facilite grandement la hiérarchisation des objets graphiques de la scène. Dans le paradigme de OpenSG le graphe de scène est un graphe acyclique dirigé composé de nœuds (*nodes)* dotés à leur tour d'un noyau (*core*). Les nœuds servent à la connectivité et les noyaux à la fonctionnalité. Les noyaux peuvent être de plusieurs types : transformation, matériau, groupe, géométrie, lumières, niveau de détail...

Le graphe de scène pour notre programme (voir figure 3.11) va comme suit : le nœud racine `SceneRoot` contient la transformation de navigation et a pour enfants

les nœuds d'éclairage et le nœud `WorldRoot`, parent de tous les objets de la scène. Chaque objet `GraphicObject` a aussi un graphe de scène local composé de trois nœuds : `TransformNode` pour la position et l'orientation, `ScaleNode` pour la mise à l'échelle et `GeoNode` pour contenir la géométrie de l'objet. Les objets graphiques de type `ObjGroup` peuvent à leur tour être parents d'autres objets graphiques permettant ainsi de hiérarchiser les objets entre eux (par exemple, les points de contrôle d'une NURBS sont enfants de celle-ci).

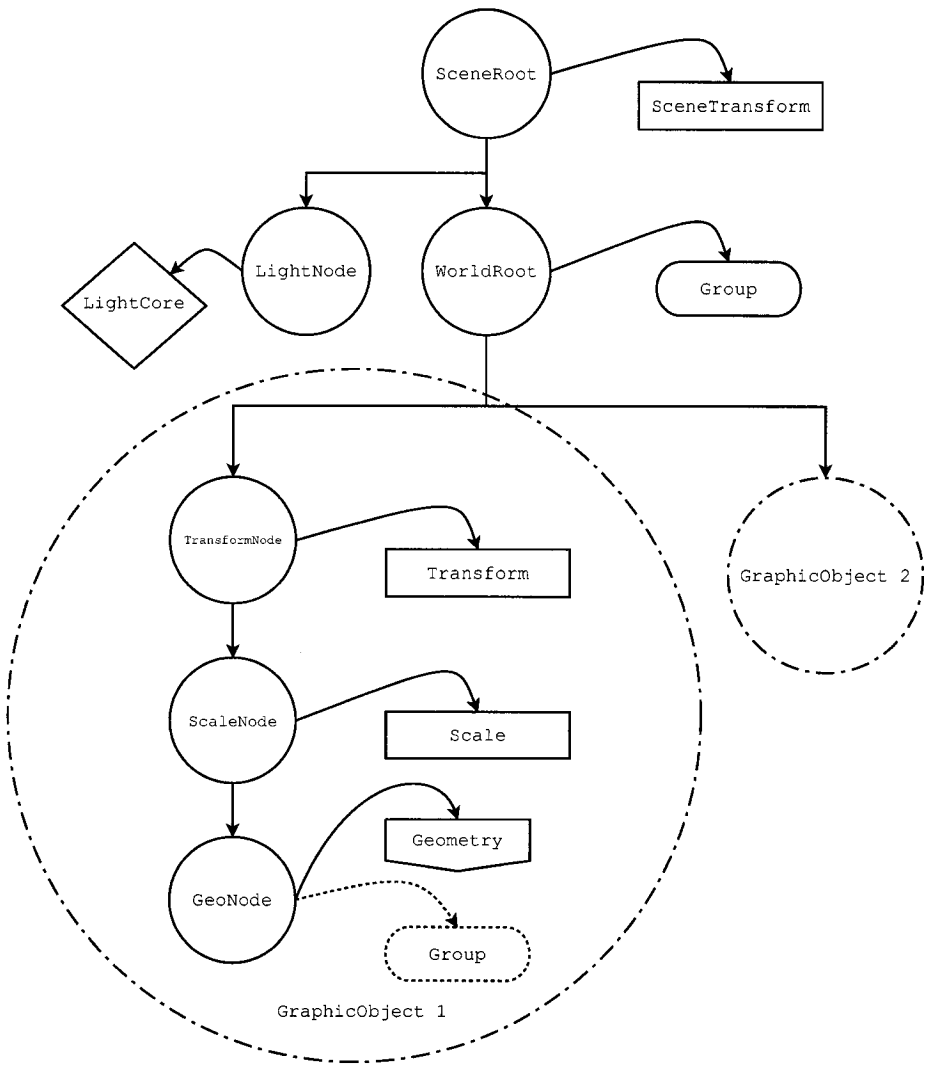


FIGURE 3.11 Graphe de scène

3.4.8 Séquences de synchronisation

Nous allons ici présenter une séquence de synchronisation d'objets partagés afin d'illustrer l'ordonnancement des opérations décrites précédemment. La figure 3.12 montre la synchronisation avec un objet déjà présent dans la scène. Nous pouvons détailler chacune des étapes :

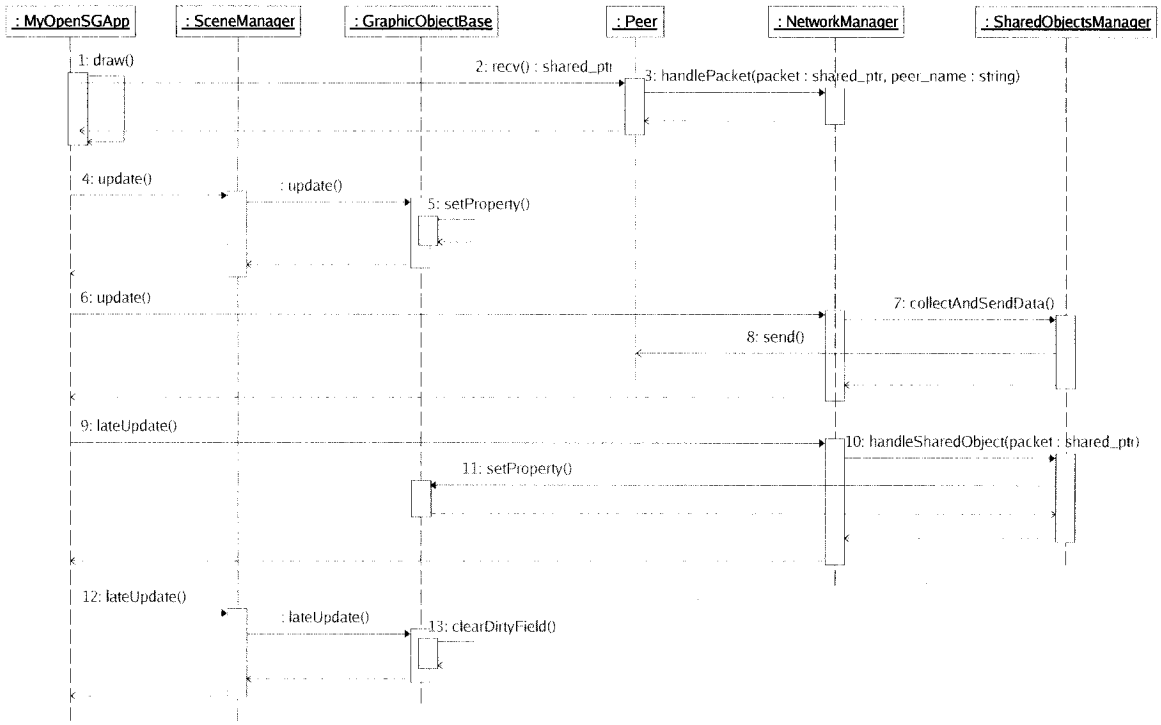


FIGURE 3.12 Synchronisation d'un objet existant

1. Nous n'implémentons pas cette fonction mais laissons la classe de base `vrj::OpenSGApp` s'en charger. Nous la montrons ici pour mettre en évidence les séquences de la trame.

2. Rappelons-nous que nos objets `Peer` sont sur un fil d'exécution différent du reste de l'application. En fait, comme nous utilisons les fonctionnalités offertes par VR Juggler pour les fils d'exécution, ceux-ci sont virtuels et n'ont véritablement la main que pendant l'affichage de la scène. C'est pourquoi les paquets entrants

arrivent à ce moment-là.

3. Le `NetworkManager` s'occupe du paquet entrant. Comme nous l'avons vu à la section 3.2.4 le paquet n'est pas distribué immédiatement mais mis en queue pour qu'il puisse être distribué aux autres nœuds dans le cas d'une grappe d'affichage.
4. Une fois l'affichage rafraîchi, les méthodes de mise à jour sont appelées, à commencer par celle du `SceneManager`. Celui-ci met à jour la scène qui fait de même avec chacun des objets graphiques.
5. C'est pendant ces mises à jour que les objets agissent les uns sur les autres. Nous illustrons ici qu'un des objets subit un changement sur un champ quelconque. Bien entendu, son masque de bits est affecté pour refléter que ce champ a été modifié.
6. Une fois la scène mise à jour, c'est au tour du `NetworkManager` qui appelle ses sous-gestionnaires à collecter et faire envoyer leurs données.
7. Ici, le `SharedObjectsManager` parcourt son registre, se rend compte que l'objet modifié en (5) a changé et l'encapsule dans un `SharedObjectsPacket`.
8. Il demande alors au `NetworkManager` d'envoyer ce paquet aux pairs connectés.
9. Nous sommes maintenant dans la deuxième phase de mises à jour. Dans le cas où nous sommes sur une grappe d'affichage, la queue de messages est dupliquée sur tous les nœuds et le paquet reçu en (3) est encapsulé dans un message de cette queue. On appelle le `NetworkManager` qui dépile la queue, trouve un message-paquet, en extrait le paquet et cherche le `PacketHandler` qui pourra le prendre en charge.
10. `SharedObjectsManager` répond à l'appel. Il extrait d'abord du paquet le GUID de l'objet et le compare avec son registre ; ici le GUID y est présent. Il dé-sérialise alors le paquet sur son objet partagé local.

11. Au cours de cette dé-sérialisation, les champs modifiés sont récupérés et appliqués correctement.
12. C'est enfin la deuxième mise à jour du `SceneManager`.
13. Tous les objets graphiques de la scène mettent leur masque de changements à zéro. S'ils ne sont pas modifiés à la prochaine trame, aucun paquet ne sera envoyé.

3.5 Aperçu du prototype

La figure 3.13 montre deux usagers interagissant avec une surface NURBS. Cette surface a servi dans le troisième scénario des tests usagers présentés au prochain chapitre. Voir l'annexe II pour d'autres saisies d'écran.

FIGURE 3.13 Deux collaborateurs éditant une surface NURBS. On peut voir tous les éléments d'interface présentés précédemment : le panneau de contrôle de paramètres au fond à droite, la corbeille en bas à droite et la bibliothèque à fichiers sur la gauche.

CHAPITRE 4

ÉVALUATION ET DISCUSSION

Nous avons présenté au chapitre précédent l'architecture mise en place pour répondre à nos contraintes au sein de MOSAIC et nos objectifs de recherche sur les environnements de télé-collaboration. Le prototype résultant de cette architecture doit nous servir à valider nos hypothèses. Dans ce chapitre, nous présentons les tests usagers effectués pour cette validation, nous évaluons nos choix architecturaux puis discutons de l'atteinte des objectifs de cette recherche avant de nous tourner vers les avenues futures de ce travail de recherche.

4.1 Tests Usagers

Afin de valider le prototype de notre ECDI, nous avons procédé à une séance de tests usagers. Nous avons invité plusieurs participants du projet MOSAIC, professeurs, étudiants et associés de recherche, à se rendre à la CAVE de l'École polytechnique. Ces participants représentent des utilisateurs cible de notre environnement ; ce sont donc toutes des personnes familières avec le design, l'analyse et l'optimisation en aéronautique. L'objectif de ces tests usagers est double : (1) évaluer d'adéquation de nos choix architecturaux avec notre but d'une télé-collaboration et (2) mettre en évidence l'apport de la télé-collaboration à une activité de design.

Tout au long de la séance, les participants se sont relayés d'un site de collaboration à l'autre pour essayer les deux interfaces (immersive et sur poste de travail). À la fin de la séance, nous avons recueillis les commentaires des participants et les avons invités à remplir un questionnaire d'évaluation (présenté à l'annexe I). Ce questionnaire cherche

à mesurer la facilité qu'ont eu les usagers à utiliser l'interface (en immersion comme sur poste de travail), à interagir avec les objets et à collaborer les uns avec les autres.

4.1.1 Mise en place

Les participants étaient invités à prendre place sur l'un des trois sites de collaboration : le premier en immersion dans la CAVE (composée de 4 machines d'affichage et d'une machine maître) et les deux autres sur des postes de travail. Ces dernières sont situées dans la même salle que la CAVE, ainsi les collaborateurs pouvaient communiquer entre eux de vive voix ; nous n'avons donc pas utilisé de logiciel de téléconférence pour cette fin. Cependant, les ordinateurs et la CAVE sont placés de telle façon que les usagers se tournent le dos et ne voient pas les écrans de leurs collaborateurs, les forçant à communiquer oralement pour se coordonner les uns avec les autres. Toutefois, dans le cas où nous aurions à faire collaborer des sites réellement distants, il n'y aurait aucune difficulté à utiliser un logiciel comme Skype pour assurer la téléconférence.

Les machines utilisées dans ces tests sont toutes sur système d'exploitation Linux et dotées de cartes graphiques adéquates pour les besoins de notre ECDI. Les utilisateurs étaient tous familiers avec l'environnement Linux et avec les concepts de design (éléments d'une aile d'avion, surfaces NURBS) mais ne l'étaient pas, pour la plupart, avec les environnements d'immersion (utilisation du bras de repérage et navigation). Une machine maître contrôle les 4 machines d'affichage de la grappe et permet d'avoir une vue générale de la scène.

Le monde virtuel se composait de :

- Une bibliothèque contenant deux fichiers : une aile déjà générée et une surface NURBS simple repliée sur elle-même.
- Une corbeille pour disposer des objets.

– Un panneau de design affichant 7 paramètres géométriques pour générer des ailes.

Les avatars des collaborateurs étaient représentés par un objet géométrique—comportant une tête et un torse—d'une couleur différente pour chacun. Les dispositifs de repérages étaient représentés par des cylindres semi-transparents de même couleur que l'avatar.

4.1.2 Déroulement des tests

Nous avons commencé par une brève introduction aux environnements immersifs et avons invité un participant à s'installer dans la CAVE et se familiariser avec la navigation. Nous avons ensuite invité un autre participant à s'installer sur un des sites monopostes et avons expliqué les commandes à la souris et au clavier pour simuler l'interaction avec le bras de repérage. Nous les avons laissés explorer l'environnement pour un temps avant de leur proposer trois scénarios de collaboration :

1. Générer deux ailes différentes avec le panneau de paramètres et les comparer.
2. Modifier à deux le profil d'une aile.
3. Collaborer à trois pour « déplier » une surface NURBS simple.

Nous n'avons pas reproduit tous les scénarios énoncés dans nos besoins fonctionnels à la section 2.4 mais nous pouvons cependant évaluer correctement notre interface et le design collaboratif en couvrant le design préliminaire (section 2.4.1) avec le deuxième scénario et le contrôle d'un processus extérieur (section 2.4.4) avec le troisième. Le premier scénario se rapproche d'une revue de design en proposant aux collaborateurs de comparer deux ailes.

Premier scénario : comparaison d'ailes

Dans ce scénario, un des participants s'emparait du panneau de design, modifiait les paramètres et lançait la génération d'une aile avec un programme extérieur développé dans MOSAIC. Il allait chercher l'aile pendant que son collègue en générait une autre. Ils se sont alors parlés pour aligner les deux ailes côte à côte. La plus grande difficulté semblait, pour l'usager dans la CAVE, de se déplacer pour faire le tour de l'aile et se placer en face de son collègue. Ils devaient communiquer leurs intentions pour ne pas prendre la même aile tous deux car, comme mentionné au chapitre 2, nous n'avons pas implémenté de verrouillage sur les objets partagés. Quand cela arrivait, l'aile sautait du bras de repérage de l'un à celui de l'autre mais une fois les usagers prévenus de la nécessité de leur coopération, ils se montraient plus vigilants et communiquaient leurs intentions avant que d'agir. Le scénario n'a cependant pas été très concluant car il n'était pas très facile de comparer les ailes uniquement de visu.

Deuxième scénario : modification du profil d'une aile

Deux usagers étaient présents en même temps dans ce scénario. Le premier commençait par sortir une aile pré-générée—fournie par une autre tâche de MOSAIC—de la bibliothèque. Les deux se sont ensuite réunis autour de l'aile et se sont mis à la déformer en déplaçant les points de contrôle. Une des difficultés rencontrées était due à un léger *bogue* dans l'application : normalement les points de contrôle d'une NURBS apparaissent quand un usager amène son bras de repérage proche de la NURBS et disparaissent quand il l'éloigne ; le problème survient quand un usager est en train d'interagir avec la NURBS et qu'un autre usager éloigne son bras, faisant du coup disparaître les points de contrôle et gênant le premier usager. Les usagers devaient, pour palier à cette situation, demander à leur collègue de s'approcher de nouveau.

Troisième scénario : dépliage d'une surface NURBS

Dans ce scénario, un des participants se rendait à la bibliothèque à fichiers pour amener la surface à déplier dans la scène, une NURBS simple comportant une vingtaine de points de contrôle. Les collaborateurs se sont ensuite donnés rendez-vous plus loin et se sont placés autour de la surface. Ils devaient ainsi communiquer leurs intentions constamment quand ils voulaient déplacer la surface et avertir les autres quand ils voulaient s'emparer de tel ou tel point de contrôle. Ils devaient aussi coordonner leur vision de la surface car chacun pouvait voir la NURBS d'un point de vue différent (un des collaborateurs s'était même placé au-dessus des autres). Ce scénario a été très concluant, les collaborateurs ayant réussi à déplier la surface en ramenant tous les points de contrôle dans le même plan.

4.1.3 Résultats des tests usagers

Nous avons obtenu un retour sur les tests usagers de deux façons : quantitativement avec un formulaire et qualitativement pendant une discussion faisant suite à la séance. Nous présentons ici ces résultats.

Résultats quantitatifs

Nous avons compilé les questionnaires de tous les participants et nous nous fions à la moyenne de leurs réponses pour dégager la tendance générale.

Interface immersive et sur poste de travail. Si on compare (voir figure 4.1) l'aisance d'interface entre l'environnement immersif et le poste de travail on remarque que les usagers ont été plus à l'aise à manipuler les objets dans la CAVE alors qu'ils

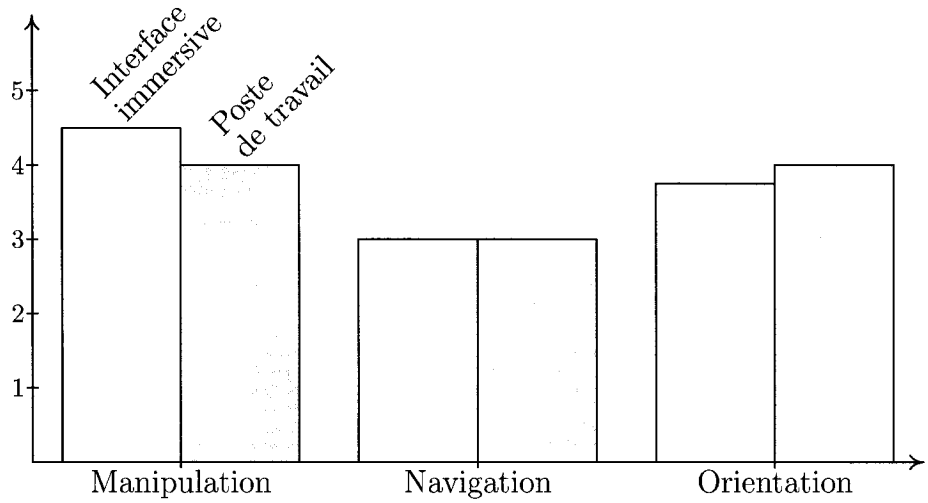


FIGURE 4.1 Résultats du questionnaire d'évaluation : interface

s'orientaient un peu plus facilement sur poste de travail. En revanche ils n'ont pas trouvé la navigation très aisée, autant dans la CAVE que sur poste de travail.

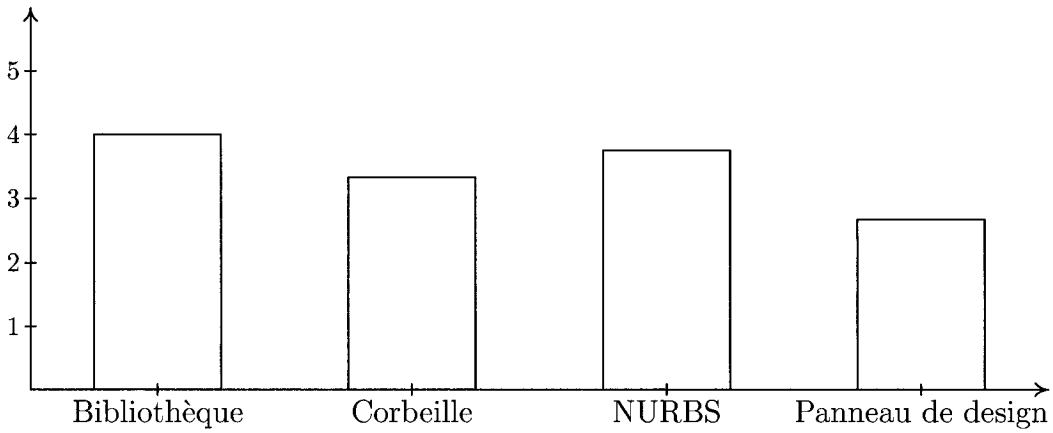


FIGURE 4.2 Résultats du questionnaire d'évaluation : interactions

Interaction avec les objets. Nous retenons surtout que le panneau de design était difficile à manipuler (voir figure 4.2). Les usagers ont trouvé l'interaction avec la bibliothèque et la manipulation d'une NURBS intuitives mais pas tant la corbeille.

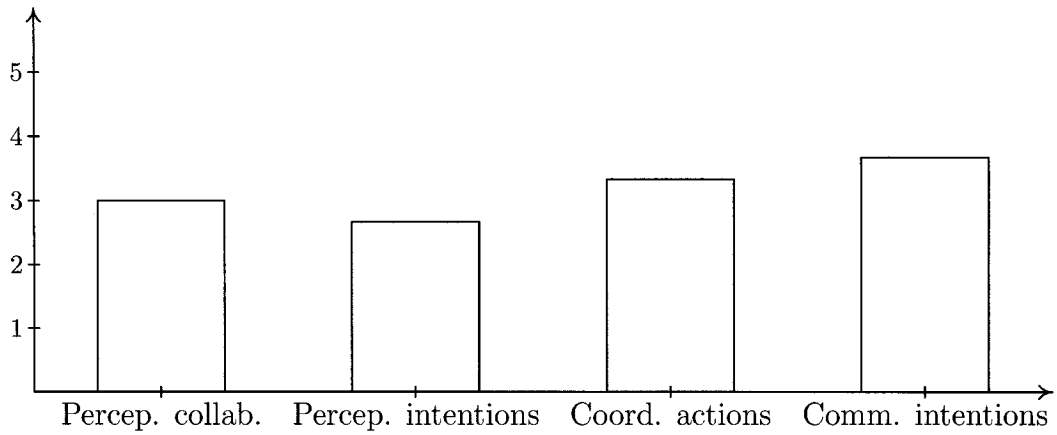


FIGURE 4.3 Résultats du questionnaire d'évaluation : collaboration

Collaboration. Il est intéressant de remarquer ici (figure 4.3) que les usagers semblent trouver plus facile de communiquer leurs intentions que de percevoir celles des autres. La perception visuelle des collaborateurs ne semble pas très aisée mais les usagers témoignent que la coordination de leurs actions reste assez bonne.

Résultats qualitatifs

Au début de la séance, lorsque les participants exploraient encore le monde virtuel, il y avait peu de dialogues entre eux et chacun « errait » dans le monde sans communication. Mais une fois des objectifs précisés, à travers les scénarios, les participants se sont mis à parler beaucoup plus entre eux et cherchaient à collaborer.

À la fin de la séance, les participants ont émis nombre de commentaires au sujet de leur expérience avec l'ECDI. La plupart de leurs commentaires confirment et expliquent les résultats des questionnaires :

Interface immersive et sur poste de travail. En immersion, les usagers ont trouvé la navigation difficile surtout pour la rotation ; ils n'ont pas trouvé intuitif

d'avoir à faire un angle droit entre le bras de repérage et la direction de la tête pour pouvoir tourner. En général, ils auraient aimé ne pas avoir à se déplacer aussi souvent et ont émis le souhait qu'un mode de navigation centré sur l'objet (en cours d'édition) soit disponible. Ils ont aussi mentionné que le monde virtuel manquait de repères spatiaux et ont proposé que des points cardinaux soit inscrits afin, par exemple, de pouvoir nommer rapidement un lieu aux autres. En ce qui concerne les boîtes englobantes indiquant la sélection d'un objet, les usagers auraient préféré qu'elles soient alignées sur les axes locaux des objets plutôt que sur ceux du monde, ce qui les rendrait plus petites.

Interaction avec les objets. Les usagers ont trouvé qu'à la longue, il était peu pratique d'avoir à se déplacer jusqu'à la corbeille pour déposer l'objet qu'il voulaient effacer. Ils auraient préféré qu'un bouton ou une combinaison de boutons permette de détruire un objet. Ils ont trouvé que certains objets étaient trop gros (surtout dans la CAVE) et ont proposé que l'on puisse appliquer des mise à l'échelle aux objets. Ils ont trouvé le panneau de design particulièrement volumineux, ce qui explique les résultats du questionnaire à cet égard.

Collaboration. Les usagers ont été assez satisfaits de leur expérience de collaboration. Ils ont cependant mentionné que le *bogue* des boîtes englobantes mentionné plus haut les avait gêné de temps en temps. Ils ont aussi proposé que les points de contrôle des NURBS soient numérotés pour qu'ils puissent les nommer aux autres quand ils essayent de se coordonner.

4.2 Évaluation des choix architecturaux

Nous revenons brièvement ici sur nos choix architecturaux exprimés à la section 2.5.2 et les examinons à la lumière de nos tests de validation.

Immersion. L'immersion était une caractéristique principale pour notre ECDI et cela nous a amené à rechercher un paradigme d'interface intuitif. Comme nous nous y attendions, l'immersion s'est avérée utile pour la manipulation des objets virtuels en 3D. Les usagers ont trouvé très naturel de se saisir des objets dans l'espace et de les déplacer.

Périphériques d'entrée. Le bras de repérage est adéquat à la manipulation. Par contre, pour la navigation, il serait souhaitable de proposer d'autres méthodes plus directes d'interaction. Par exemple, le bouton analogique du bras de repérage pourrait servir à se déplacer latéralement de côté et vers l'avant et l'arrière. Il faudrait alors, afin de pouvoir l'utiliser, investiguer sur la raison d'incompatibilité des pilotes.

Gestion de la cohérence. Notre mécanisme de synchronisation des changements par masques de bits s'avère efficace pour répliquer les modifications locales aux autres pairs. Le manque de verrouillage gêne un peu et il serait souhaitable d'y pallier. Nous ne voyons cependant pas de difficulté à le faire : dans notre architecture il suffirait d'ajouter au décorateur d'interaction `Grabable` un champ indiquant que l'objet est pris, auquel cas il serait impossible de s'en saisir à nouveau. De part la réplication automatique des changements des objets, ce champ serait dupliqué chez les autres collaborateurs, les empêchant de saisir l'objet avant qu'il ne soit lâché.

Gestion de l'attention. Notre choix s'avère aussi adéquat : étant donné le nombre restreint de collaborateurs, nous n'avons pas besoin de gestion avancée de l'attention. De plus, nous n'utilisons pas excessivement la bande passante car nous ne transmettons que les changements apportés aux objets. Nous pourrions cependant ajouter sans difficulté un mécanisme d'estimé de position (voir section 1.3.7) pour éviter l'envoi inutile de paquets pour les positions de l'avatar et du bras de repérage lorsque ceux-ci ne bougent pas ou peu.

Visualisation. La lecture du fichier de visualisation scientifique `VU/pirate` s'effectue sans problème et valide bien notre choix de l'analyseur lexico-syntaxique ANTLR. Maintenant que les grammaires sont écrites, il serait facile d'ajouter la visualisation d'autres entités que les NURBS.

Persistance des objets de monde virtuel. Notre ECDI offre une persistance limitée mais qui est très naturelle aux collaborateurs, ceux-ci pouvant joindre et quitter la session en cours à loisir. Cependant, il manque à notre prototype une persistance statique du monde virtuel bien que nous n'en avions pas besoin dans l'immédiat. Ce serait un ajout souhaitable pour sauvegarder des sessions de design, il suffirait de diriger la sérialisation des objets, déjà présente, vers un fichier binaire plutôt que le réseau.

Téléconférence. Bien que nous n'ayons pas utilisé de logiciel de téléconférence dans nos tests, nous sommes confiants que le logiciel testé, Skype, fonctionnerait bien en situation de collaboration distante.

4.3 Discussion

Nous avions plusieurs attentes face à notre prototype d'ECDI. Pour atteindre nos objectifs, il nous fallait nous assurer que notre environnement s'intègre bien dans les tâches de MOSAIC (en offrant le contrôle d'un processus distant et la visualisation des résultats), qu'il permette la collaboration et qu'il présente une interface intuitive. Les tests usagers nous révèlent avec satisfaction que nos objectifs ont été bien rencontrés.

4.3.1 Intégration avec les tâches de MOSAIC

Notre environnement permet non seulement la collaboration entre usagers mais aussi la collaboration avec des programmes extérieurs développés au sein d'autres tâches de MOSAIC. Nous avons également implémenté la lecture de fichiers de visualisation scientifique, en commençant par les surface NURBS. Ce format de fichier (`VU/pirate`) est largement utilisé au sein de MOSAIC et permettra, par la suite, de visualiser des entités autres que les surfaces NURBS. Le panneau de contrôle nous permettait de modifier les paramètres associés à un design et d'appeler un programme extérieur en lui fournissant ces paramètres pour visualiser ensuite le résultat de son calcul.

En cela, les objectifs d'intégration avec les tâches de MOSAIC sont atteints : le prototype de l'ECDI permet de visualiser des ailes d'avion et de dialoguer avec un programme qui les génère à partir de paramètres manipulables au sein de l'environnement.

4.3.2 Collaboration

Les usagers ont bel et bien collaboré, notamment pendant le troisième scénario. Il est apparu qu'ils « croyaient » véritablement au monde virtuel quand l'un d'eux a

demandé, à la cantonade, qui était le « bleu au fond » : il voyait en l'avatar la représentation d'un de ses collègues et cherchait à connaître son identité pour s'adresser à lui et collaborer à leur tâche commune. Ajoutons aussi que les usagers avaient véritablement investi leur espace virtuel. Ils pouvaient quitter (l'application) à tout moment et rejoindre quelque temps après la séance (qui avait continué sans eux). Il réapparaissaient, notifiaient aux autres leur nouvelle présence et s'intégraient aux activités en cours dans le monde virtuel.

Il reste que la première condition à la bonne collaboration est la bonne communication. Lorsque les usagers se saisissaient du même objet, ils se nuisaient l'un l'autre. Nous avions, pour un temps, considéré l'utilisation d'un jeton pour assurer que les objets du monde virtuel ne puissent être modifiés que par un usager à la fois mais il s'est avéré que cela entravait la collaboration par la nécessité de se passer le jeton sans arrêt. Nous pouvons faire un constat similaire avec la corbeille : la nécessité de se déplacer pour effacer un objet est un geste inutile qui détourne le focus de l'usager sur la tâche qu'il est en train d'accomplir. Malgré ces aspects qui mériteraient à terme d'être corrigés, les usagers ont pu collaborer à tes tâches communes au sein de l'ECDI en communiquant leurs intentions, en partageant les tâches et en coordonnant leurs actions.

4.3.3 Intuitivité de l'interface

Les usagers ont démontré que la manipulation des objets en immersion leur paraissait intuitive, surtout pour l'utilisation de la bibliothèque à fichiers et pour l'édition d'une surface NURBS. Notre paradigme se voulant simple s'est montré adéquat. Nous avons remarqué que le seul objet qui ne présentait pas une manipulation intuitive était le panneau de design. En effet, ce dernier dérogeait un peu à notre principe de simplicité avec des boutons radios à manipuler s'inspirant un peu trop d'un paradigme de

gadgets logiciels 2D. Il serait souhaitable de trouver pour ce panneau une alternative d'interface avec un vocabulaire de glisser-déposer, comme pour tous les autres objets.

En ce qui concerne la navigation, la possibilité de se déplacer autour d'un objet de la scène serait très souhaitable d'autant plus que ce serait assez trivial à implémenter dans notre architecture. En effet, comme nous l'avons vu à la section 3.3.4, il suffirait d'écrire une autre stratégie de navigation qui serait activée à la sélection d'un objet.

Par contre, nous nous rendons compte que notre interface, si elle est bien adaptée à l'environnement immersif, souffre quelque peu sur un poste de travail non-immersif. La simulation du bras de repérage nécessite des combinaisons clavier et souris qui, bien qu'elles s'acquièrent à l'usage, ne sont pas d'emblée naturelles. Une solution serait de redéfinir les combinaisons de commandes pour quelque chose de plus adapté comme par exemple ce qui se fait dans les jeux vidéo : le clavier pour les déplacements frontaux et latéraux et la souris pour le lacet et le tangage. Une autre solution serait de proposer un autre périphérique d'entrée, comme un manche à balai ou même une souris 3D.

4.4 Travaux futurs

Le champ d'application de notre travail est très ouvert et les avenues de recherche ne manquent pas. Déjà, au sein de MOSAIC les besoins sont grands pour une plateforme décentralisée où gérer nombre de processus de calcul (d'analyse et d'optimisation) et où visualiser les artéfacts qui en découlent. Nous avons choisi de ne visualiser que les surfaces NURBS mais l'extension de notre l'architecture à d'autres OAV (iso-courbes, iso-surfaces, champs de vecteurs, maillages...) ouvrirait la porte à une visualisation scientifique plus complète et permettrait des activités de design plus riches : simulations numériques, ajustement interactif d'un maillage, exploration de

solutions...

Les activités de revue de design fourniraient une opportunité intéressante à notre ECDI pour explorer le déploiement d'un environnement d'annotation des design successifs. Cette avenue serait également à poursuivre en y intégrant une gestion des configurations et des versions d'un produit et en étudiant les questions relatives à la cohabitation, dans un même environnement, du design d'un produit et de la gestion de son cycle de vie.

Du côté de la communication avec des programmes extérieurs, notre architecture gagnerait également à être ouverte et, plutôt qu'une solution ad hoc, pourrait accéder à plus d'abstraction dans l'échange de données, à un niveau plus sémantique, à travers des *Web Services*. On pourrait ainsi se référer à certains travaux ayant développé un effort ontologique dans des domaines d'intérêt (en mathématiques ou en aéronautique par exemple) et faire correspondre aux entités découpées les OAV et objets de contrôle de processus distants de l'ECDI. Il serait alors possible questionner directement les objets du monde virtuel à l'aide d'une sonde à propos de résultats qui n'étaient pas explicites dans les données. Cette abstraction sémantique permettrait également de faire collaborer l'ECDI avec des programmes sans définir explicitement les informations à échanger.

Enfin, l'interface en immersion est aussi un domaine très vivant en recherche. Il y a eu beaucoup de développements dans des paradigmes complexes qui cherchent à transposer les concepts de gadgets logiciels du 2D vers la 3D (nous pensons par exemple à la *3D Widget Library*[1]). D'autres recherches se tournent vers une linguistique gestuelle et cherchent des vocabulaires gestuels intuitifs et expressifs. Par exemple, l'utilisation du gant comme périphérique d'entrée permettrait d'avoir des interactions différentes selon la position de la main : pointer du doigt pour sélectionner, fermer le poing pour

[1] `http://graphics.cs.brown.edu/research/widgetlib`

saisir, agiter la main pour effacer... Il serait très intéressant d'explorer ces avenues en vue d'une application dans un environnement comme l'ECDI.

CONCLUSION

Ce travail de recherche nous a amené étudier les Environnements Virtuels de Collaboration dans la perspective d'un Environnement Collaboratif de Design en Immersion (ECDI). Nous avions ainsi comme objectifs de permettre une collaboration immersive avec une interface intuitive entre plusieurs usagers mais aussi d'offrir un environnement de visualisation et de design à l'aide du contrôle d'un processus distant.

Nous avons effectué une revue de littérature, présentée au chapitre 1, pour étudier la visualisation scientifique et les EVC, ce qui nous a permis d'en dégager les principales caractéristiques. Nous avons alors réuni les contraintes et besoins en termes d'intégration avec les autres tâches de MOSAIC, de collaboration immersive et non-immersive, d'interface intuitive et de besoins fonctionnels pour formuler nos hypothèses et proposer au chapitre 2 une architecture et un prototype d'ECDI.

Nous avons alors pu bâtir un prototype dont nous décrivons l'articulation au chapitre 3. Enfin, ce prototype a été validé au moyen de tests usagers présentés au chapitre 4. Nous avons ainsi vu comment notre architecture satisfait nos objectifs initiaux autant sur les plans de la collaboration et de l'interface que du dialogue avec des processus extérieurs.

Mais notre sujet de recherche reste très ouvert et de nombreuses avenues sont à explorer autant à moyen terme que de façon plus large dans l'émergente culture cybernétique du XXI^{e}siècle.

RÉFÉRENCES

ABRAM, G. ET TREINISH, L. (1995). An extended data-flow architecture for data analysis and visualization. *SIGGRAPH Computer Graphics*, **29**(2), 17–21.

APPLEGATE, L. M. (1991). Technology support for cooperative work : A framework for studying introduction and assimilation in organizations. *Journal of Organizational Computing*, **1**, 11–39.

BENFORD, S., BOWERS, J., FAHLÉN, L. E., ET GREENHALGH, C. (1994). Managing mutual awareness in collaborative virtual environments. *VRST '94 : Proceedings of the conference on Virtual reality software and technology*, 223–236. World Scientific Publishing Co., Inc.

BENFORD, S. ET FAHLÉN, L. E. (1993). A spatial model of interaction in large virtual environments. *Proceedings of the Third European Conference on CSCW.*

BENFORD, S., GREENHALGH, C., BROWN, C., WALKER, G., REGAN, T., MORPHETT, J., WYVER, J., ET REA, P. (1998). Experiments in inhabited TV. *CHI 98 conference summary on human factors in computing systems*, New York, NY, USA, 289–290. ACM Press.

BENFORD, S., GREENHALGH, C., RODDEN, T., ET PYCOCK, J. (2001). Collaborative virtual environments. *Communications of the ACM*, **44**(7), 79–85.

BRODLIE, K. W., DUCE, D., GALLOP, J., ET WOOD, J. D. (1998). Distributed cooperative visualization. deSousa, A. et Hopgood, F., éditeurs, *Eurographics '98 State of the Art Reports*, 27–60.

BUTLER, D. M., ALMOND, J. C., BERGERON, R. D., BRODLIE, K. W., ET HABER, R. B. (1993). Visualization reference models. *Proceedings of the 4^{th} conference on Visualization '93*, 337–342.

CARLSSON, C. ET HAGSAND, O. (1993). DIVE : a multi-user virtual reality system. *Virtual Reality Annual International Symposium*, 394–400. IEEE.

CHI, E. H. (2000). A taxonomy of visualization techniques using the data state reference model. *Proceedings of the IEEE Symposium on Information Vizualization 2000*, 69–75. IEEE Computer Society.

CHI, E. H. (2002). Expressiveness of the data flow and data state models in visualization systems. *Proceedings of the Advanced Visual Interfaces Conference*, 375–378. IEEE Computer Society.

CHI, E. H. ET RIEDL, J. (1998). An operator interaction framework for visualization systems. *Proceedings of the 1998 IEEE Symposium on Information Visualization*, 63–70. IEEE Computer Society.

CRUZ-NEIRA, C., BIERBAUM, A., HARTLING, P., JUST, C., ET MEINERT, K. (2002). VR Juggler – An Open Source Platform for Virtual Reality Applications. *40th AIAA Aerospace Sciences Meeting and Exhibit.*

CRUZ-NEIRA, C., SANDIN, D. J., ET DEFANTI, T. A. (1993). Surround-screen projection-based virtual reality : The design and implementation of the CAVE. *Proceedings of SIGGRAPH '93*, 135–142. ACM, ACM Press.

FOULSER, D. (1995). Iris explorer : a framework for investigation. *SIGGRAPH Computer Graphics*, **29**(2), 13–16.

FRÉCON, E. (2004). DIVE : communication architecture and programming model. *IEEE Communications Magazine*, **42**(4), 34–40.

FRÉCON, E., SMITH, G., STEED, A., STENIUS, M., ET STÅHL, O. (2001). An overview of the COVEN platform. *Presence : Teleoperators & Virtual Environments*, **10**(1), 109–127.

FRÉCON, E. ET STENIUS, M. (1998). DIVE : a scaleable network architecture for distributed virtual environments. *Distributed Systems Engineering*, **5**(3), 91–100.

GAMMA, E., HELM, R., JOHNSON, R., ET VLISSIDES, J. (1995). *Design patterns : elements of reusable object-oriented software.* Addison Wesley Professional.

GREENHALGH, C. (1996). Dynamic, embodied multicast groups in MASSIVE-2. Technical report, University of Nottingham – Department of Computer Science.

GREENHALGH, C. ET BENFORD, S. (1995). MASSIVE : a collaborative virtual environment for teleconferencing. *ACM Transaction on Computer-Human Interaction*, **2**(3), 239–261.

GREENHALGH, C., PURBRICK, J., ET SNOWDON, D. (2000). Inside MASSIVE-3 : flexible support for data consistency and world structuring. *Proceedings of the third international conference on collaborative virtual environments*, New York, NY, USA, 119–127. ACM Press.

GUTWIN, C., BENFORD, S., DYCK, J., FRASER, M., VAGHI, I., ET GREENHALGH, C. (2004). Revealing delay in collaborative environments. *Proceedings of the 2004 conference on Human factors in computing systems*, 503–510. ACM Press.

HABER, R. B. ET MCNABB, D. A. (1990). Visualization idioms : A conceptual model for scientific visualization systems. B.Shriver, G. et L.J.Rosenblum, éditeurs, *Visualization in Scientific Computing*, 74–93. IEEE Computer Society Press.

HARRISON, S. ET DOURISH, P. (1996). Re-place-ing space : the roles of place and space in collaborative systems. *Proceedings of the 1996 ACM conference on Computer Supported Cooperative Work*, 67–76. ACM Press.

HIBBARD, B. (2000). Confessions of a visualization skeptic. *SIGGRAPH Computer Graphics*, **34**(3), 11–13.

JOHANSON, M. (1998). Designing an environment for distributed real-time collaboration. *Proceedings of the IEEE Workshop on Networked Appliances.*

KELSO, J., SATTERFIELD, S. G., ARSENAULT, L. E., KETCHAN, P. M., ET KRIZ, R. D. (2003). Diverse : a framework for building extensible and reconfigurable device-independent virtual environments and distributed asynchronous simulations. *Presence : Teleoperators and Virtual Environments*, **12**(1), 19–36.

LEIGH, J., JOHNSON, A. E., BROWN, M., SANDIN, D. J., ET DEFANTI, T. A. (1999a). Visualization in teleimmersive environments. *Computer*, **32**(12), 66–73.

LEIGH, J., JOHNSON, A. E., ET DEFANTI, T. A. (1997). CAVERN : A distributed architecture for supporting scalable persistence and interoperability in collaborative virtual environments. *Virtual Reality : Research, Development and Applications*, **2.2**, 217–237.

LEIGH, J., JOHNSON, A. E., DEFANTI, T. A., ET BROWN, M. D. (1999b). A review of tele-immersive applications in the CAVE research network. *Virtual Reality*, 180–187.

MACEDONIA, M. R., BRUTZMAN, D. P., ZYDA, M. J., PRATT, D. R., BARHAM, P. T., FALBY, J., ET LOCKE, J. (1995). NPSNET : a multi-player 3D virtual environment over the internet. *Si3d '95 : proceedings of the 1995 symposium on interactive 3D graphics*, New York, NY, USA, 93–ff. ACM Press.

MARCHAK, F. M., CLEVELAND, W. S., ROGOWITZ, B. E., ET WICKENS, C. D. (1993). The psychology of visualization. *Proceedings of the 4^{th} conference on Visualization '93*, 351–354.

MORTENSEN, J., VINAYAGAMOORTHY, V., SLATER, M., STEED, A., LOK, B., ET WHITTON, M. C. (2002). Collaboration in tele-immersive environments. *Proceedings of the workshop on Virtual environments 2002*, 93–101. Eurographics Association.

OZELL, B. ET CAMARERO, R. (1995). A linguistics approach to visualization. Santo, H. P., éditeur, *Compugraphics '95*, 192–200.

OZELL, B. ET CAMARERO, R. (1996). A configurable visualization environment. *Proceedings of the 34^{th} Aerospace Sciences Meeting and Exhibit.*

OZELL, B., PIC, C., ET CAMARERO, R. (2000). Topics in collaborative visualization environments. *8^{e}conférence annuelle de la Société canadienne de CFD*, volume 1, Hôtel du Parc, Montréal, Québec, Canada, 495–502.

PANG, A. (1994). Spray rendering. *IEEE Computer Graphics and Applications*, **14**(5), 57–63.

PANG, A. ET WITTENBRINK, C. M. (1995). Spray rendering as a modular visualization environment. *SIGGRAPH Computer Graphics*, **29**(2), 33–36.

PANG, A. ET WITTENBRINK, C. M. (1997). Collaborative 3D visualization with CSpray. *IEEE Computer Graphics and Applications*, **17**(2), 32–41.

PIC, C. ET OZELL, B. (2000). VU, a configurable scientific visualization program. *Bulletin de la société canadienne de CFD*, number 12, 13–16.

SAWANT, N., SCHARVER, C., LEIGH, J., JOHNSON, A., REINHART, G., CREEL, E., BATCHU, S., BAILEY, S., ET GROSSMAN, R. (2000). The Tele-Immersive Data Explorer : a distributed architecture for collaborative interactive visualization of large data-sets. *4^{th} International Immersive Projection Technology Workshop.*

SCHMIDT, D. C. (2000). *Pattern-Oriented Software Architecture. Volume 2. Patterns for Concurrent and Networked Objects.* John Wiley & Sons.

SMITH, R. C., PAWLICKI, R. R., LEIGH, J., ET BROWN, D. A. (2000). Collaborative VisualEyes. *Proceedings of the Fourth International Immersive Projection Technology Workshop.*

STEED, A., FRÉCON, E., AVATARE, A., PEMBERTON, D., ET SMITH, G. (1999). The london travel demonstrator. *VRST '99 : Proceedings of the ACM symposium on Virtual reality software and technology*, 50–57. ACM Press.

STEED, A., MORTENSEN, J., ET FRÉCON, E. (2001). Spelunking : Experiences using the DIVE system on CAVE-like platforms. Fröhlich, B., Deisinger, J., et Bullinger, H.-J., éditeurs, *Immersive Projection Technology and Virtual Environments 2001*, 153–174. Springer.

TÖRLIND, P., STENIUS, M., JOHANSSON, M., ET JEPPSSON, P. (1999). Collaborative environments for distributed engineering. *CSCWD'99 - Computer Supported Cooperative Work in Design.*

UPSON, C., FAULHABER, T., KAMINS, D., LAIDLAW, D. H., SCHLEGEL, D., VROOM, J., GURWITZ, R., ET VAN DAM, A. (1989). The application visualization system : A computational environment for scientific visualization. *IEEE Computer Graphics and Applications*, **9**(4), 30–42.

WHELESS, G., LASCARA, C., VALLE-LEVINSON, A., BRUTZMAN, D., SHERMAN, W., HIBBARD, W., ET PAUL, B. (1996). Virtual Chesapeake Bay : Interacting with a coupled physical/biological model. *IEEE Computer Graphics and Applications*, **16**(4), 52–57.

WHELESS, G. H., LASCARA, C. M., LEIGH, J., KAPOOR, A., JOHNSON, A. E., ET DEFANTI, T. E. (1998). CAVE6D : A tool for collaborative immersive visualization of environmental data. *IEEE Visualization.*

WOOD, J. D., WRIGHT, H., ET BRODLIE, K. W. (1995a). CSCV – Computer supported collaborative visualization. *Proceedings of BCS Displays Group International Conference on Visualization and Modelling.*

WOOD, J. D., WRIGHT, H., ET BRODLIE, K. W. (1995b). Improving visualization through collaboration. Research report 95.17, School of Computer Studies, University of Leeds. Presented at the 6th Eurographics Workshop on Visualisation in Scientific Computing, Chia, Italy, 3–5 May 1995.

ANNEXE I

QUESTIONNAIRE D'ÉVALUATION

Un Environnement Collaboratif de Design en Immersion

Tests Usagers

Questionnaire d'évaluation

Veuillez répondre aux questions par un chiffre de 1 à 5 selon que l'énoncé est représentatif de votre expérience avec l'ECDI :

1. Pas du tout représentatif
2. Pas vraiment représentatif
3. Assez représentatif
4. Bien représentatif
5. Absolument représentatif

A. Interface Immersive (dans la CAVE)

Je n'ai pas eu de difficulté à manipuler les objets	
J'arrive à me déplacer facilement	
J'arrive à m'orienter facilement	

B. Interface sur poste de travail

Je n'ai pas eu de difficulté à manipuler les objets	
J'arrive à me déplacer facilement	
J'arrive à m'orienter facilement	

C. Interaction avec les objets

J'ai facilement pu utiliser la bibliothèque	
J'ai facilement compris comment utiliser la corbeille	
J'ai facilement pu manipuler une NURBS	
J'ai facilement pu manipuler le panneau de paramètres	

D. Collaboration

Je perçois facilement où sont mes collaborateurs	
Je perçois facilement les intentions des mes collaborateurs	
J'arrive à coordonner mes actions avec mes collaborateurs	
J'arrive à communiquer mes intentions à mes collaborateurs	

ANNEXE II

SAISIES D'ÉCRAN DU PROTOTYPE

Nous présentons ici une série de saisies d'écran du prototype. Il est à noter que celles-ci ont été effectuées en mode simulation. Ainsi, apparaît une représentation supplémentaire du bras de repérage (en vert sur les images). Dans l'environnement immersif seul le cylindre semi-transparent apparaît à l'endroit où l'usager pointe son bas de repérage.

FIGURE II.1 Deux usagers en train de sélectionner un fichier de la bibliothèque.

FIGURE II.2 Les usagers travaillent à éditer l'aile qu'ils viennent de charger. Les points de contrôle de la NURBS apparaîssent comme des cubes lorsque l'aile est sélectionnée.

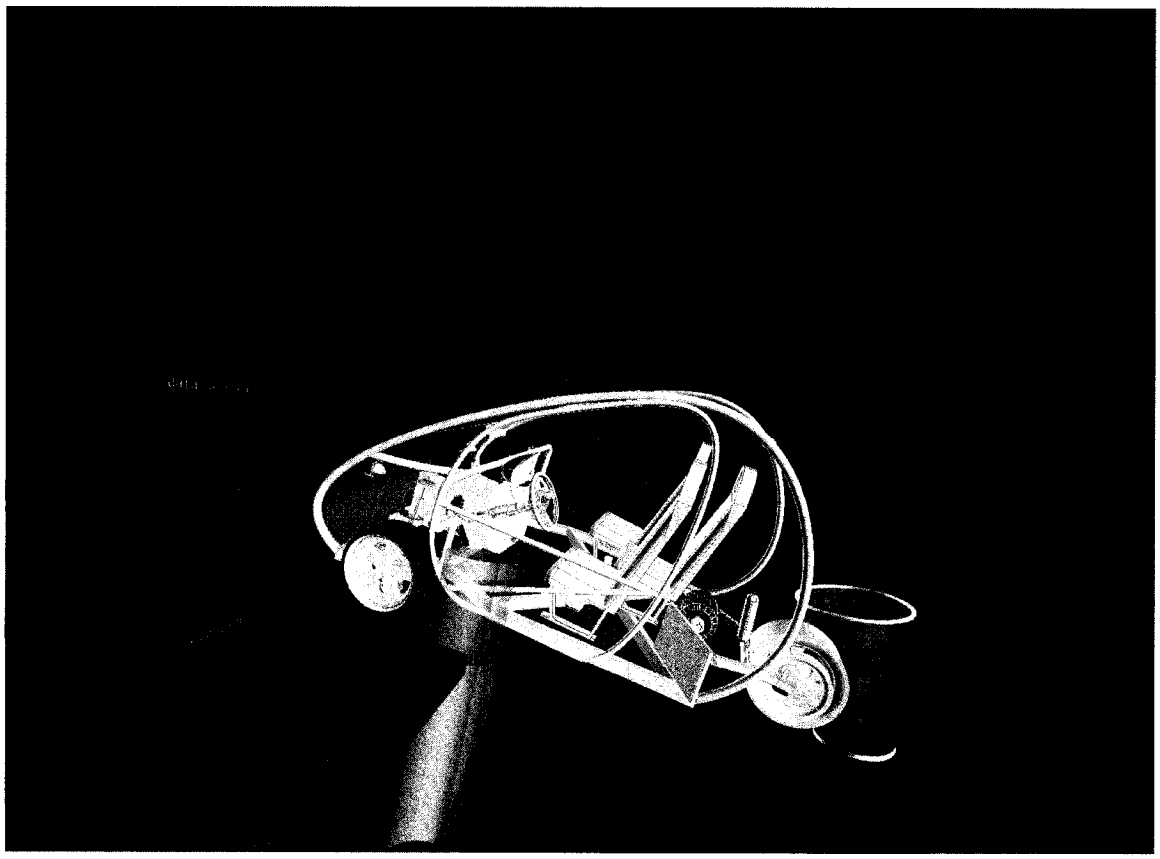


FIGURE II.3 Les usagers effectuent ici une revue de design d'un véhicule électrique.