Interactions coopératives 3D distantes en environnements virtuels : gestion des problèmes réseau

Chadi Zammar

N° d'ordre: 05-01

THÈSE

Présentée

devant l'Institut National des Sciences Appliquées de Rennes

pour obtenir

le grade de : Docteur de l'Institut National des Sciences Appliquées de Rennes
Mention Informatique

par

Chadi El Zammar

Équipe d'accueil : SIAMES - IRISA
École Doctorale : Matisse
Titre de la thèse :

Interactions coopératives 3D distantes en environnements virtuels : gestion des problèmes réseau

soutenue le 25 Janvier 2005 devant la commission d'examen

M. :	Jean-Louis	Pazat	Président
MM. :	Sabine	Coquillart	Rapporteurs
	Jacques	Tisseau	
MM. :	Mireille	Ducassé	Examinateurs
	Thierry	Duval	
	Jean-Pierre	Jessel	
	Bruno	Arnaldi	

À mes parents

Remerciements

Je souhaite en premier lieu remercier Bruno Arnaldi pour m'avoir accueilli au sein de l'équipe SIAMES, pour son support et pour la confiance qu'il m'a accordée en acceptant de diriger cette thèse.

Il n'y a pas de superlatif assez grand pour exprimer ma profonde gratitude à Thierry Duval, qui a été pour moi un grand ami et un encadrant exceptionnel. Je lui suis très reconnaissant pour tous les conseils, toute l'aide, toute la confiance et pour tous les moyens qu'il a mis à ma disposition. Cette thèse n'aurait peut-être pas vu le jour sans son support permanent et inconditionnel.

Je tiens aussi à exprimer ma reconnaissance envers Sabine Coquillart et Jacques Tisseau qui m'ont fait l'honneur d'être mes rapporteurs. Je remercie aussi Jean-Louis Pazat, Jean-Pierre Jessel et Mireille Ducassé pour leur participation au Jury de cette thèse. Je remercie aussi le projet VTHD++ pour avoir financé cette thèse.

Je veux saluer tous ceux qui ont fait partie de l'équipe SIAMES pour la superbe ambiance de travail et l'amitié qui s'est établie entre nous. Un grand merci à David Margery pour son aide et son soutien sans limite.

Enfin, je souhaite remercier tous ce qui m'ont soutenu dans le plan moral et affectif pendant ces années : Marie-Claude, Geoffroy, Renaud, Pascal, Gaël, Louis, André, Martin, merci infiniment à vous tous pour m'avoir accompagné durant ce voyage.

Table des matières

Introduction

Un environnement virtuel appelé aussi monde virtuel ou univers virtuel est un monde alternatif généré par des ordinateurs qui a pour but de simuler un univers dit “virtuel”. Cet univers simulé peut être une imitation du monde réel, mais il peut être aussi un univers imaginaire qui peut être régi par des lois autres que celles qu'on trouve dans notre univers dit “réel”. Par extension, un environnement virtuel coopératif est un environnement virtuel qui est simulé sur plusieurs machines en même temps, ce qui permet à plusieurs utilisateurs, même dispersés géographiquement, de partager le même univers virtuel.

Ces environnements virtuels sont de plus en plus utilisés dans les exercices militaires, la formation dans le domaine industriel, les jeux multi-utilisateurs, le télé-enseignement, les conférences virtuelles, la télé-présence, la télé-médecine, ainsi que dans d'autres domaines.

Un environnement virtuel coopératif permet la coopération entre différents utilisateurs afin qu'ils accomplissent une tâche particulière. Dans le cas le plus typique et représentatif les utilisateurs se partagent le même espace en même temps pour réaliser une tâche coopérative.

Pour assurer un niveau de coopération acceptable entre les différents utilisateurs, il faut prendre en considération certains critères. Nous pouvons diviser ces critères selon deux catégories :

- la première est visuelle/psychologique et elle est souvent liée à l'utilisateur. Il s'agit de comprendre les interactions de l'autre utilisateur : que fait-il ? Que va-t-il faire ? Sur quoi interagit-il ? ... Pour comprendre tout cela il faut assurer que l'utilisateur ait un certain niveau de conscience de tout ce qui évolue dans l'environnement. Cette conscience est la connaissance de l'état de l'environnement qui est assurée à travers des métaphores visuelles significatives, d'où le facteur psycho-cognitif.
- La deuxième catégorie est technique et elle est liée au réseau. Pour garantir la mise à jour de l'évolution de l'environnement virtuel il faut assurer la transmission d'informations liées à cette évolution dans les meilleurs délais. L'assurance de la réception des mises-à-jour nécessite à son tour une qualité de service réseau jugée acceptable en terme de latence et de bande passante. Qu'arrive-t-il si cette qualité de service est interrompue à un moment

donné ? Quel est l'impact d'un problème au niveau réseau sur la coopération ? Jusqu'à quel point la perte de données suite à un problème réseau peut-elle être tolérable ?

Cette thèse a pour objectif d'essayer de répondre à ces questions en mettant l'accent sur les conséquences indésirables d'un problème réseau sur les environnements virtuels coopératifs. Nous allons suivre le problème réseau dès le moment de son apparition au niveau de la couche de transport, et tout le long de son passage par les différentes couches qui composent les environnements virtuels distribués. Ces couches comportent : la couche de communication, le noyau du support d'exécution d'environnements virtuels, et enfin la couche applicative en relation directe avec l'utilisateur.

Sur chaque niveau nous étudions les effets provoqués par le problème réseau et nous proposons des solutions. Nos contributions suite à cette thèse sont les suivantes :

Chapitre 5 Conception et réalisation d'un algorithme de synchronisation tolérant au délai réseau et à la déconnexion d'un site au sein d'un environnement virtuel distribué. Le but est de permettre à une simulation distribuée de continuer même en présence de problèmes réseaux ou de déconnexions de sites participants.

Chapitre 6 Conception et réalisation d'un mécanisme de migration d'objets au niveau noyau d'un support d'exécution d'environnements virtuels. Le but est de pouvoir conserver l'accès aux objets critiques dans le cas d'un problème réseau.

Chapitre 7 Conception et réalisation d'un mécanisme d'ajout et de suppression dynamique de sites au sein d'un environnement virtuel distribué. Le but est de tirer un avantage supplémentaire du mécanisme de migration pour rendre l'accès à une simulation possible dynamiquement contrairement au schéma statique très contraignant.

Chapitre 8 Conception et réalisation d'un mécanisme d'information et de prise de conscience d'un problème réseau via des métaphores visuelles. Le but est d'informer l'utilisateur que ses objets virtuels ne sont pas accessibles, ou non-à-jour, suite à un problème réseau.

Avant d'expliciter les détails de nos contributions, nous estimons utile pour la compréhension globale du sujet d'introduire une bibliographie comportant :

Chapitre 1 Une introduction générale de la réalité virtuelle qui comporte sa définition et son évolution au fil des années écoulées.

Chapitre 2 La couche de communication et ses effets sur les plates-formes de développement d'environnements virtuels distribués, ainsi que l'étude de quelques protocoles dédiés aux besoins particuliers de la réalité virtuelle.

Chapitre 3 Quelques méthodes avancées utilisées dans certains supports d'exécution d'environnements virtuels distribués pour réduire le taux de communication entre les sites et atténuer par la suite les conséquences d'un problème réseau.

Chapitre 4 Les effets d'un problème réseau sur l'interaction et la coopération au sein d'un environnement virtuel distribué.

Cette thèse a été réalisée dans le cadre du projet RNRT VTHD++ [1].

[1] www.vthd.org

Première partie

État de l'art

Chapitre 1

Odyssée de la réalité virtuelle

1.1 Introduction

Imaginez vous en train de vous promener dans un centre commercial, vous regardez les vitrines, vous entrez dans le magasin de votre choix, vous faites un tour entre les rayons pour découvrir les dernières nouveautés. Vous pouvez même acheter le produit que vous avez choisi. Rien d'étonnant, sauf si vous êtes en train de le faire sans quitter votre domicile, une tasse de thé à la main !

Imaginez vous encore en train de placer ou déplacer les meubles dans votre maison, ou changer la couleur des murs dans votre salon, de rendre le séjour plus lumineux en élargissant la fenêtre. Ces travaux sont très coûteux ou parfois même impossibles à réaliser à cause des grosses modifications de structure de votre maison que cela peut imposer. Ne serait-il pas mieux de pouvoir le faire avant même que la maison ne soit construite ? Pour répondre à cette question, les architectes ont cherché à donner une représentation la plus réaliste possible de leurs projets : d'une part, pour mieux les faire comprendre au public ; d'autre part, pour prévenir les problèmes qui pourraient survenir lors de la construction. Grâce à la réalité virtuelle, ils peuvent maintenant réaliser des simulations de bâtiments avec un degré de réalisme jamais atteint jusqu'à présent. Nombreux sont les projets d'architecture et d'aménagement d'espaces intérieurs ou extérieurs qui commencent à utiliser un environnement virtuel en guise de maquette. Celle-ci devient un véritable espace visitable, que l'on peut évaluer et modifier avant la construction.

La formation de pilotes civils ou militaires est une tâche qui coûte cher et qui comporte des dangers pour la vie dans le cas d'une erreur humaine dûe à la faible expérience ou peut être à une défaillance quelconque. Beaucoup de pilotes auraient préféré acquérir leur expérience sans être confrontés à des dangers réels. Dans le cas d'un entrainement militaire, là aussi le danger est très présent. Les ingénieurs qui travaillent dans des endroits critiques tels que les centrales

nucléaires sont exposés aux radiations qui peuvent être cancérigènes. Tous ces dangers peuvent être évités en utilisant des simulateurs dédiés à l'apprentissage et l'entrainement, comme il en existe déjà pour la simulation de vols civils et militaires. De même, la NASA et l'Agence Spatiale Européenne (ESA) travaillent sur plusieurs projets de simulateurs pour la formation des spationautes qui embarqueront sur les stations orbitales *Freedom* ou *Columbus*. La réparation du téléscope spatial Hubble, réalisée par une équipe de la NASA avec le succès que l'on sait, a été minutieusement préparée à l'aide d'un tel simulateur. Nous avons aussi eu la preuve de l'efficacité des simulateurs lors de la Guerre du Golfe. Cette efficacité a été clairement mise en évidence, puisque les pilotes ont pu s'entrainer durant des mois sur des simulateurs avec des données qui provenaient directement des satellites espions américains. A leur arrivée sur le champ de bataille, ils en connaissaient déjà tous les recoins. La réalité virtuelle a été une des raisons de leur victoire.

Ces exemples et beaucoup d'autres que l'on trouve dans presque tous les domaines nous montrent que dans la vie réelle, le danger est lui aussi réel et peut être fatal dans certain cas. Changer la réalité est impossible, mais éviter ses dangers en utilisant les technologies récentes est faisable. Le but de la recherche en réalité virtuelle ne se limite pas seulement au fait d'éviter les dangers évoqués ci-dessus, mais il peut être orienté à des fins ludiques tel que les jeux vidéos ou le cinéma de synthèse.

Récemment, une nouvelle discipline thérapeutique basée sur les applications de la réalité virtuelle semble voir le jour. Selon les chercheurs, la réalité virtuelle pourrait avoir de nombreuses applications dans le traitement des phobies ou peurs maladives courantes. La thérapie basée sur la réalité virtuelle s'avère aussi efficace que la thérapie traditionnelle d'exposition in vivo pour surmonter la peur de l'avion, le vertige ou les phobies telles que la peur des araignées (arachnophobie). Le principe est simple : le patient est exposé à des environnements virtuels qui, petit à petit, le confrontent à l'objet de ses peurs.

Beaucoup d'autres domaines de la médecine sont intéressés par la réalité virtuelle, parmi lesquelles on peut citer : la téléchirurgie, la formation des médecins praticiens, la réhabilitation des handicapés, ...

Dans ce qui suit, nous allons discuter quelques définitions de la réalité virtuelle ainsi que son historique (paragraphe 1.2). Les différents types de configuration matérielles et logicielles de réalité virtuelle sont ensuite présentés (paragraphe 1.3). Enfin, nous allons mettre l'accent sur le lien qui existe entre les réseaux informatiques et la naissance de la réalité virtuelle distribuée (paragraphe 1.4).

1.2 La réalité virtuelle : définitions et historique

Réalité Virtuelle est un oxymoron, c'est-à-dire une expression formée de deux mots antinomiques qui laissent une certaine ambiguïté chez le lecteur. Le *réel*, en philosophie, c'est ce qui existe indépendamment de la pensée humaine, tandis que le *virtuel* évoque quelque chose d'irréel, de fictif, qui n'appartiendrait pas à la vraie réalité. Ce terme se rapporte à l'idée que cette *réalité* dite *virtuelle* est quelque chose qu'on peut expérimenter comme toute autre chose réelle sauf qu'elle n'est pas réelle en elle-même. Si nous regardons un film ou si nous lisons un livre, nous pouvons avoir une expérience réelle d'émotions, de joie, de tristesse, nous pouvons vivre l'histoire bien que cette histoire n'ait jamais existé. Cette réalité, vécue par le téléspectateur pendant un film, peut être appelée une "réalité virtuelle". De ce point de vue, l'expérience de la réalité virtuelle remonte dans le temps et n'est pas du tout quelque chose de nouveau. Toute expérience mystique, religieuse ou même paranoïaque, est une expérience de réalité virtuelle même si cette expérience a été nommée autrement.

De nos jours, le terme de réalité virtuelle désigne principalement une technologie très spécifique qui est devenue célèbre et médiatisée dans les années 90. Pendant cette période, nous avons commencé à entendre parler de ce terme qui a été parfois mal utilisé ou placé en dehors de son contexte, et ceci est lié à la mauvaise compréhension des gens de ce que la réalité virtuelle veut dire et de ses limites. C'est à cette époque aussi que plusieurs terminologies sont apparues pour désigner cette nouvelle technologie comme *Environnement Virtuel* qui est un terme adopté surtout par les académiques et les chercheurs *en raison de l'exagération et des espérances peu réalistes associées* [Gig93] (*because of the hype and the associated unrealistic expectations*) tandis que les industriels avaient tendance à utiliser le terme *Réalité Virtuelle*. Parallèlement, le terme *Réalité Artificielle* était populaire au Japon. D'autres terminologies ont été utilisées aussi, comme *Expérience Synthétique*, *Monde Virtuel*, *Monde Artificiel*, toutes veulent dire la même chose.

Expliquer la réalité virtuelle est une tâche complexe d'autant qu'il n'y a pas une seule définition mais plusieurs définitions qui évoluent au fil des temps en fonction des différentes disciplines impliquées par cette nouvelle technologie.

En 1993 la NASA a défini la réalité virtuelle ainsi [NJ93] :

La réalité virtuelle est l'expérience humaine de percevoir et d'interagir, via des "capteurs" et "effecteurs", avec des objets simulés dans un environnement synthétique comme si ces objets étaient réels.

Virtual reality is the human experience of perceiving and interacting through sensors and effecters with a synthetic (simulated) environment, and with simulated objects in it, as if they were real.

Le livre *Silicon Mirage* [AB92] définit la *Réalité Virtuelle* comme étant *un*

moyen pour les humains de visualiser, manipuler et interagir avec des ordinateurs et des données très complexes.

Virtual reality is a way for humans to visualize, manipulate and interact with computers and extremely complex data.

Les deux définitions précédentes, ainsi que des dizaines d'autres que l'on peut trouver, bien qu'elles aient quelques différences apparentes, sont pratiquement équivalentes. Toutes veulent dire que la réalité virtuelle est une expérience interactive dans le sens où l'utilisateur est capable de changer les événements et pas seulement être un spectateur, et immersive (donne le sentiment de la présence) au sein d'un monde virtuel autonome simulé par des ordinateurs. C'est de cette manière que Zeltzer [Zel92] a compris la réalité virtuelle en la schématisant par un cube qui montre ses composants de base, comme illustré dans la figure 1.1.

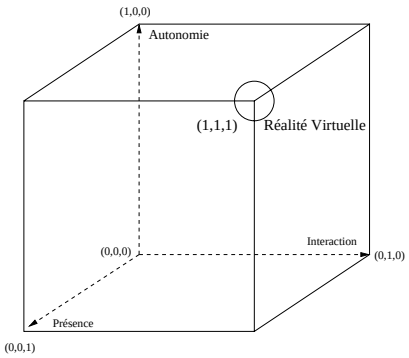

FIG. 1.1 – Cube de Zeltzer : Autonomie, Interaction et Présence en Réalité Virtuelle

Burdea [Bur93] a remplacé l'autonomie de Zeltzer par l'*imagination* et la présence par l'immersion dans sa représentation de la réalité virtuelle car pour lui les applications de la réalité virtuelle utilisent la créativité humaine qui, elle, est basée sur l'imagination, ce que l'on peut voir dan la figure 1.2.

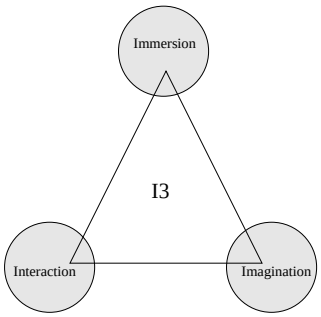

FIG. 1.2 – Les Trois I de la Réalité Virtuelle de Burdea

Tisseau a placé le concept d'autonomie au cœur de la problématique de recherche dans ses réflexions épistémologiques sur la réalité virtuelle [Tis01] : *Notre démarche s'appuie ainsi sur un principe d'autonomie selon lequel l'autonomisation des modèles numériques qui constituent les univers virtuels est indispensable à la réalité de ces univers. L'autonomisation des modèles conduit à les doter de moyens sensorimoteurs, de moyens de communication et de moyens de coordination perceptions-actions. Cette autonomie par conviction s'avère être une autonomie par essence pour les modèles d'organismes, une autonomie par nécessité pour les modèles de mécanismes, et une autonomie par ignorance pour les modèles de systèmes complexes caractérisés par une grande diversité de composants, de structures et d'interactions. L'utilisateur humain est représenté au sein de l'univers virtuel par un avatar, modèle parmi les modèles, dont il contrôle les coordinations perceptions-actions. Il est en relation avec son avatar par l'intermédiaire d'un langage et d'interfaces comportementales adaptées qui rendent possible la triple médiation des sens, de l'action et du langage.*

Le traité de la réalité virtuelle [ABC+03] fournit une définition plus technique : *la réalité virtuelle est un domaine scientifique et technique exploitant l'informatique et des interfaces comportementales en vue de simuler dans un monde virtuel le comportement d'entités 3D, qui sont en interaction en temps réel entre elles et avec un ou des utilisateurs en immersion pseudo-naturelle par l'intermédiaire de canaux sensori-moteurs.*

Deux autres termes sont souvent associés à la réalité virtuelle : *Téléprésence* et *Cyberspace*. S'ils sont étroitement couplés à la réalité virtuelle, ils ont pourtant une signification différente :

Téléprésence est une sorte de réalité virtuelle spécifique qui simule un environnement réel mais distant. Le mot téléprésence se réfère d'ailleurs à l'expérience sensorielle de sa propre présence dans un espace lointain.

Cyberspace est un terme qui a été inventé et défini par William Gibson en 1983 [Gib83]. Gibson est un auteur de science fiction né à Vancouver au Canada. Neuromancer [Gib83] est peut être son roman le plus célèbre, dans lequel Gibson a une vision d'un futur proche très noir, où il a utilisé le terme *Cyberspace* pour se rapporter à une matrice de données électroniques contrôlée par des sociétés puissantes. De nos jours ce terme est plutôt associé aux systèmes de jeux et à l'Internet.

1.3 Configuration matérielles et logicielles de réalité virtuelle

Un système de réalité virtuelle est un ensemble de matériels et de logiciels nécessaires à la création d'un monde virtuel. Un tel système doit fournir aux

participants la sensation de *présence* dans un endroit autre que celui où ils sont physiquement.

En 1962, Morton Heilig a conçu un des premiers systèmes dont le but était de fournir à l'utilisateur une sensation de présence. Heilig n'était pas un chercheur en réalité virtuelle mais un directeur de production de films cinématographiques qui faisait des documentaires dans les années 50. Son système appelé *Sensorama Simulator* [Rhe91] a été conçu comme une machine de jeu du type Arcade qui fournit la sensation de monter une moto à Brooklyn. *Sensorama* avait un système vidéo stéréoscopique (obtenu par des caméras accolées), et fournissait du mouvement, de la couleur, du son stéréo, des odeurs, du vent (à l'aide de petits ventilateurs placés près de la tête de l'utilisateur) et un siège qui vibrait. La figure Fig. 1.3 montre une photographie du *Sensorama* prise en 1962. La différence principale entre le système de Heilig et les systèmes récents est le fait que le *sensorama* n'était pas interactif.

FIG. 1.3 – Sensorama Simulator par Heilig, 1962

Plus tard, en 1970, Myron Krueger a conçu un système fournissant un degré assez élevé d'interaction, ce qui manquait le plus dans le *Sensorama*. Ce système baptisé *Videoplace* [Kru91] crée un environnement qui répond aux mouvements des participants. En effet, *Videoplace* est une forme de réalité projetée. Krueger a utilisé un grand écran en face de l'utilisateur. Sur cet écran, une ombre de l'utilisateur est affichée comme le montre la figure 1.4. Il était également possible d'afficher plusieurs personnes sur le même écran. Un utilisateur pouvait toucher

les silhouettes représentant d'autres utilisateurs ainsi que manipuler les objets graphiques et animés qui apparaissent sur l'écran donnant l'illusion d'une forme de vie artificielle. *Videoplace* diffère de la plupart des systèmes actuels par le fait que l'utilisateur se voit dans le monde virtuel par une vue objective *"third person point of view"*, alors que de nos jours la plupart des systèmes présentent une vision subjective *"first person point of view"*.

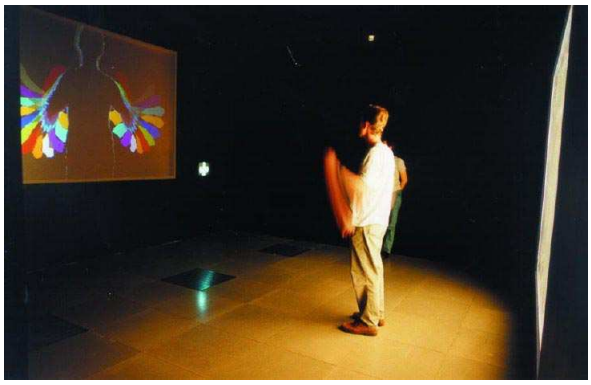

FIG. 1.4 – Videoplace de Krueger, 1970

En 1984, des scientifiques de la NASA ont développé le premier visiocasque *HMD (Head Mounted Display)* commercial qui a ouvert la voie à la commercialisation de produits dédiés à la réalité virtuelle. Un visiocasque est un casque de vision stéréoscopique (c'est-à-dire de vision en relief). Un visiocasque comporte donc deux écrans, placés très près des yeux afin de couvrir le champ de vision le plus large possible, et sur lesquels s'affichent les images. Cette simulation du mode de vision réel contribue à donner l'impression au porteur du casque d'être "dans l'environnement virtuel".

Les militaires, à leur tour, ont été très intéressés par les avancées de ce domaine pour le développement des simulateurs dédiés à l'entrainement au vol. Le problème était que les simulateurs existants à l'époque coûtaient très cher et avaient besoin d'un matériel conçu pour un modèle spécifique d'avion. Lorsque ce modèle devenait obsolète le simulateur le devenait également. Ceci a poussé les chercheurs à s'orienter vers le développement des plates-formes logicielles génériques.

Dans les dernières années plusieurs plates-formes logicielles de réalité virtuelle ont vu le jour. Parmi les principales on peut citer : NPSNET [MZP+94], MASSIVE [GB95], DIVE [CH93], RING [Fun95], OpenMASK [MAC+02].

1.3.1 Types de systèmes de réalité virtuelle

Il est un peu difficile de classifier tous les systèmes de réalité virtuelle, mais la plupart des configurations entrent dans trois catégories principales. Chaque ca-

tégorie peut être classée selon le sentiment d'immersion qu'elle offre, ou le degré de présence qu'elle peut fournir. Plus le système est immersif, plus le degré de concentration de l'utilisateur sur la tâche qu'il réalise est important. L'immersion ou la présence sont généralement le produit de plusieurs paramètres comme l'interactivité, la complexité de l'image, la vue stéréoscopique, ou encore le champ visuel...

1.3.1.1 Systèmes non-immersifs

Comme son nom l'indique, un système de réalité virtuelle non-immersif est un système qui n'offre rien de particulier pour renforcer visuellement le sentiment de présence dans l'environnement virtuel qu'il génère. Un tel système peut utiliser par exemple le bureau électronique (*Desktop*) classique d'un ordinateur et l'environnement virtuel est vu à travers une fenêtre en utilisant la résolution d'un écran standard. Ces systèmes sont appelés aussi WOW "*Window On a World*" [Isd93]. L'interaction avec un tel environnement peut se faire par des moyens conventionnels tels que des claviers, des souris ou même en utilisant des dispositifs 3D un peu plus élaborés comme les *SpaceBalls* ou les gants *DataGlove*.

L'avantage d'un système non-immersif est qu'il n'exige pas de matériels spécifiques, par conséquent un tel système peut être implanté sur un ordinateur normal.

1.3.1.2 Systèmes semi-immersifs

Un système semi-immersif comporte normalement une unité de calcul graphique performante en terme à la fois de résolution d'image et de fréquence de production, qui sera couplée avec un grand écran et un système de projection. Ces systèmes augmentent la sensation d'immersion d'un utilisateur en offrant un champ visuel large. Il est très important de calibrer la géométrie de l'image projetée par rapport à la forme de l'écran pour éviter les déformations de l'image. Les systèmes semi-immersifs sont très utiles pour les applications éducatives car ils permettent une expérience multi-utilisateurs simultanée qui n'est pas envisageable dans le cas d'un système HMD. Le *Reality Center* développé par *Silicon Graphics Inc* est un exemple des systèmes semi-immersifs (Fig. 1.5).

1.3.1.3 Systèmes entièrement immersifs

L'expérience de la réalité virtuelle la plus directe est fournie par les systèmes entièrement immersifs. Ce genre de système exige souvent l'utilisation des HMDs (*head mounted display*) pour améliorer la sensation d'immersion. Les systèmes entièrement immersifs donnent une sensation de présence dans l'environnement

FIG. 1.5 – Reality Center

virtuel qui ne peut pas être égalé par d'autres approches. Le *Cave* est une variation des systèmes entièrement immersifs obtenue en utilisant des projections multiples d'images 3D dans un cube composé d'écrans de visualisation qui entourent complètement l'utilisateur (Fig. 1.6).

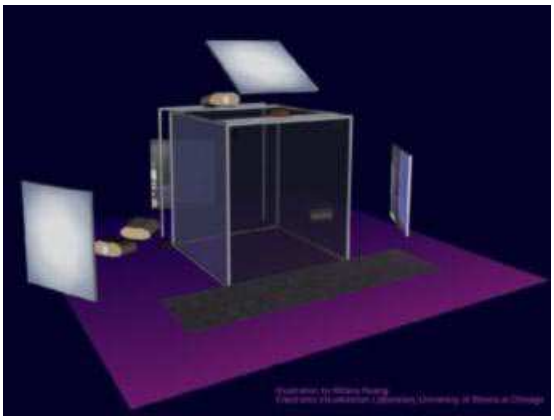

FIG. 1.6 – Cave

Le *Holodeck* [1] (ou *Holographic Environment Simulator*) utilisé dans la série télévisée *Star Trek: The Next Generation* est une bonne extrapolation future de systèmes entièrement immersifs.

1.3.1.4 Bilan

Le tableau 1.1 offre un bilan rapide des différentes performances qualitatives offertes par les différentes configurations matérielles d'environnements virtuels.

[1] Une technologie qui peut recréer virtuellement une copie conforme de la réalité

Systèmes/Critères	Présence	Résolution	Type d'immersion	Champ visuel
Station de travail simple	basse	élevée	non-immersif	faible (50^0)
Station de travail + périphériques d'interaction 3D	moyenne/élevée	élevée	dépend des périphériques	faible (50^0)
Reality center	moyenne/élevée	élevée	semi-immersif	moyen (150^0)
Cave	élevée	élevée	immersif	élevé
HMD	élevée	moyenne	immersif	moyen (150^0)

TAB. 1.1 – Performance qualitative des différents systèmes de réalité virtuelle

1.4 Passage du concept d'une machine isolée au concept de groupe avec l'arrivée des réseaux

En parallèle à l'évolution des systèmes de RV, une autre évolution ou même une "révolution" était en train de naître dans les laboratoires de l'ARPA (*U.S Defense Department's Advanced Research Projects Agency*). L'ARPA a été mise en place pour renforcer les développements scientifiques susceptibles d'être utilisés à des fins militaires, ce qui s'explique par le contexte de Guerre Froide et par le lancement du premier satellite par l'URSS, *Spoutnik*, en 1957. Puis, dans les années 1970, l'ARPA continua ses recherches dans l'étude des protocoles de transfert de données entre des réseaux d'ordinateurs, réseaux qui pouvaient être de natures différentes. Le système fut nommé ARPANET (c'est-à-dire le réseau de l'ARPA) (Fig. 1.7). En 1972, ARPANET comportait déjà une quarantaine de lieux ou institutions branchés sur le réseau. Il constituait donc le premier réseau informatique et aussi la base originelle d'*Internet*. On y faisait circuler du courrier électronique et on pouvait aussi se connecter à distance par une procédure appelée *rlogin*, contraction de *remote login* (connexion à distance). Le nom d'*Internet* (qui élargissait l'ARPANET à l'*Inter-networking*) fut alors adopté et développé entre les différentes universités américaines.

Le concept de réseau a révolutionné l'ordinateur et le monde des communications. La réalité virtuelle, à son tour, a su profiter de ce nouveau concept. Les chercheurs en réalité virtuelle ont commencé dans les années 70 à étudier la possibilité de concevoir des environnements virtuels distribués sur plusieurs machines et interconnectés par un réseau. Plus tard, le succès de quelques uns de ces environnements virtuels distribués a conduit à la création des simulateurs distribués dédiés à l'entrainement. Dans les annés 80, avec l'apparition de la technologie des microprocesseurs et du WAN (*Wide Area Network*) à grande vitesse, il est devenu possible de développer des réseaux de simulateurs à grande échelle. Une des premières implémentations réussies d'un tel réseau d'environnement virtuel temps réel a été SIMNET [CDG+93]. SIMNET (*Simulator Networking*) était un

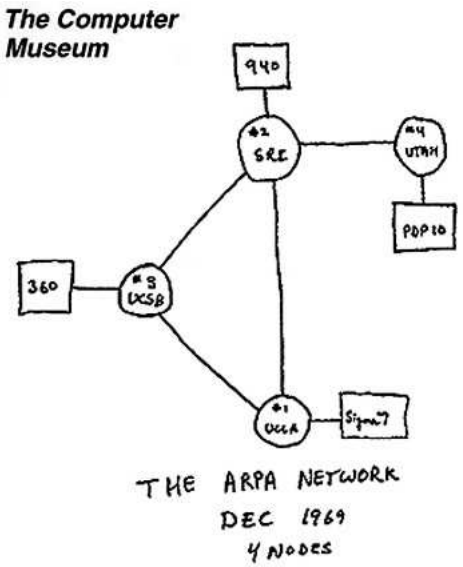

FIG. 1.7 – Premier réseau ARPA

projet d'entraînement militaire de simulation de champ de bataille. SIMNET a été développé et implémenté entre 1983 et 1990 par l'ARPA en collaboration avec *Bolt Beranek and Newman Inc* et *Perceptronics Inc.* Plus de détails concernant ce système seront discutés dans le chapitre suivant dédié aux systèmes de réalité virtuelle distribués.

La "coopération" est une nouvelle notion qui a été introduite grâce aux réseaux et systèmes de réalité virtuelle distribués. Cette notion veut dire que plusieurs utilisateurs peuvent agir simultanément au sein du même environnement virtuel. La coopération permet aux utilisateurs de réaliser des tâches qui nécessitent des manœuvres à plusieurs. De plus, la notion de coopération participe à rendre les environnements virtuels plus utiles et surtout plus réalistes.

1.5 Synthèse

L'odyssée de la réalité virtuelle n'est pas encore terminée. Dans cette odyssée, imagination et réalité se croisent pour former une existence où la réalité n'est autre que ce que l'on voit et l'on ressent, et le virtuel n'est autre que notre conscience que ce monde n'existe pas, même si ses conséquences sont bien réelles.

La réalité virtuelle est non seulement une technologie mais aussi une aventure humaine. Au fil des années le développement des environnements virtuels est devenu un domaine qui a attiré beaucoup de chercheurs. La construction d'un environnement virtuel est devenue un objectif pour les organisations militaires,

les scientifiques, ainsi que pour la commercialisation des jeux vidéos.

Les environnements virtuels sont devenus variés et s'étendent des applications mono-utilisateur, où un seul participant se connecte sur une seule machine, jusqu'aux applications multi-utilisateurs où plusieurs participants partagent en parallèle la même simulation. Des simulateurs dédiés à l'entrainement civil ou militaire deviennent de plus en plus primordiaux pour former des pilotes, des astronautes, des médecins, ou encore des ingénieurs.

La réalité virtuelle ajoute une nouvelle dimension à la culture humaine. L'influence de cette nouvelle technologie est en train de toucher et changer presque tous les autres domaines. Enfin, tant que l'imagination humaine fait partie de cette odyssée, la réalité virtuelle ne connaîtra aucune limite.

Chapitre 2

Réseaux et systèmes distribués

2.1 Introduction

La conception d'un environnement virtuel nécessite des considérations et des besoins particuliers. L'introduction de notions comme la coopération et l'interaction dans les systèmes multi-utilisateurs augmente la complexité de conception et de réalisation d'un tel environnement. La raison est que différents programmes sur différentes machines doivent être gérés ensemble (interactions inter-processus) et les participants doivent communiquer périodiquement durant la simulation tout en interagissant avec l'environnement virtuel (interactions homme-machine).

Les environnements virtuels distribués ont besoin d'être servis par une couche de transport qui satisfait simultanément plusieurs critères. Cette couche de transport doit pouvoir :

- communiquer les changements d'états entre les différents sites le plus vite possible : chaque utilisateur doit pouvoir interagir à n'importe quel moment sur n'importe quel objet au sein de l'environnement virtuel. Ces interactions doivent être diffusées à tous les utilisateurs dans les délais les plus brefs. Sinon, plus le délai de communication et de mise-à-jour est important, plus les incohérences augmentent entre les perceptions de l'environnement qu'ont les différents utilisateurs.
- assurer une bande passante la plus large possible : un environnement virtuel distribué doit supporter plusieurs utilisateurs ainsi qu'un nombre considérable d'objets faisant partie d'une simulation.
- supporter la communication de données complexes qui peuvent être des modèles graphiques, des sons enregistrés, des comportements et des descriptions de nouveaux types d'objets.

Les couches de communication actuellement utilisées dans les applications commerciales sont divisées en deux catégories basiques : les systèmes à serveur central CS (*Central Server*), et les systèmes adhérant au standard DIS (*Distributed In-*

teractive simulation).

Les systèmes CS sont surtout utilisés pour les applications à base de MUD[1], forum de discussion, et les jeux multi-utilisateurs. Les systèmes DIS ont eu leur succès dans les simulateurs d'entraînement militaire.

Ces différentes architectures de communication utilisent des protocoles de transport classiques comme *TCP* et *UDP*, mais aussi des protocoles dédiés, conçus spécialement pour répondre à des besoins spécifiques.

Dans ce chapitre, nous allons détailler quelques protocoles dédiés aux environnements virtuels distribués en commencant par un rappel des protocoles classiques de base. En effet, les protocoles classiques ne sont pas forcément suffisants pour répondre aux nouveaux besoins de communications dans les systèmes de développement d'environnements virtuels distribués. Dans ce qui suit, nous détaillons le besoin qui a poussé à définir de nouveaux protocoles et architectures réseaux dédiés aux environnements virtuels.

2.2 Protocoles classiques de communication

Un protocole réseau est l'ensemble des règles qui fixent le déroulement d'une communication entre deux applications. Un protocole peut être défini comme un format de données que les utilisateurs s'échangent. Il définit aussi la façon dont la communication doit se faire (initialisation, progression, terminaison). Le livre *Analyse structurée des réseaux* [KR03] définit un protocole ainsi : *Un protocole définit le format et l'ordre des messages échangés entre deux entités interlocutrices ou plus, ainsi que les actions générées au moment de la transmission ou réception d'un message ou de tout autre événement.*

Les protocoles sont un peu comme les langages de programmation, de sorte qu'on trouve des niveaux qui s'étendent du plus haut au plus bas selon les besoins. Par exemple dans le réseau *Internet*, les programmes d'application utilisateur définissent des protocoles de haut niveau comme le transfert de fichiers (*FTP*) et le courrier électronique (*SMTP*). Les protocoles de bas niveau gèrent l'utilisation des matériels de connexions.

De nos jours, il y a des milliers de protocoles de réseau différents en service. La section 2.2.1 discute les protocoles communs de transports puis la section 2.2.2 présente les méthodes de transmission de données.

[1]MUD (Multiple User Dimension/Dungeon) est un logiciel où chaque utilisateur peut se connecter et explorer l'environnement. Chaque participant sera représenté par un avatar.

2.2.1 Les protocoles de transport

Un protocole de couche transport fournit une communication de type logique entre des processus d'applications exploités sur des serveurs différents. Une communication logique veut dire que, du point de vue de l'application, les serveurs sont en connexion directe, même s'ils sont distants ou séparés par de nombreux routeurs. Les processus d'application utilisent les connexions logiques établies par la couche Transport pour échanger des messages, sans se préoccuper des caractéristiques de l'infrastructure physique utilisée pour les véhiculer.

2.2.1.1 *Internet Protocol* "IP"

Dans l'environnement Internet, nous sommes concernés par un ensemble de couches de protocoles souvent nommés architecture TCP/IP. La figure 2.1 illustre la hiérarchie conceptuelle de cette architecture.

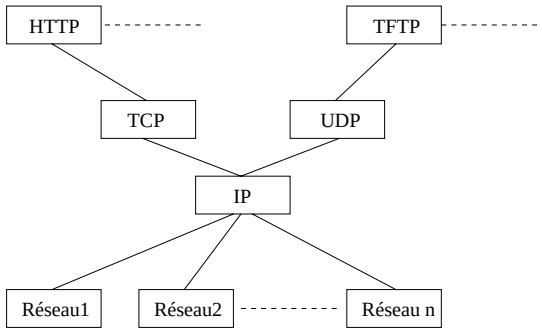

FIG. 2.1 – Protocole IP

L'IP est le protocole qui a la charge d'interconnecter les multiples technologies de réseau (réseau1, réseau2, ..) de façon à obtenir un seul réseau logique. La plupart des machines connectés à *Internet* utilisent le protocole IP pour communiquer. IP est un protocole de bas niveau utilisé par les routeurs pour assurer la transmission des paquets entre la machine source et celle de destination. IP comporte un mécanisme de segmentation et réassemblage SAR (*Segmentation And Reassembly*). Ce mécanisme découpe les gros paquets en petits fragments lors de la transmission sur un canal qui ne peut pas supporter les grands paquets, et puis c'est le même mécanisme qui se charge du réassemblage plus tard. L'entête IP contient un champ TTL (*Time To Live*) qui spécifie le nombre maximal de *sauts* (*hops*) pour atteindre la destination. A chaque fois qu'un paquet est transféré par un routeur, le champ TTL est décrémenté, s'il atteint zéro, le packet est ignoré. Ce mécanisme empêche qu'un paquet soit routé indéfiniment sur le réseau.

Deux versions différentes d'IP coexistent. La version la plus répandue est la version 4, connue sous le nom d'IPv4 (RFC 791). La version la plus récente est

la version 6 (RFC 2373 et RFC 2460) qui remplacera progressivement la version 4, elle est connue sous le nom d'IPv6.

Normalement, les applications n'utilisent pas IP directement. Elles emploient un des protocoles qui sont décrits au dessus d'IP. IP est la base de deux protocoles principaux et très célèbres : TCP (*Transmission Control Protocol*) et UDP (*User Datagram Protocol*). Ces deux protocoles offrent des canaux logiques pour les programmes d'application utilisateur. Ils offrent aussi d'autres services parmi lesquels des services d'acquittements et de retransmission.

2.2.1.2 *Transmission Control Protocol* "TCP"

TCP est un protocole de transport qui offre un service fiable en utilisant un système d'acquittement et de retransmission. TCP est défini dans les documents RFC 793, 1122, 1323, 2018 et 2581. TCP a prouvé son utilité pour plusieurs catégories d'applications qui ont besoin de garantir la réception des paquets par ordre et sans perte. Une connexion suivant TCP commence par un appel envoyé par le client vers le serveur désigné. Au cours de l'établissement de la connexion TCP, les deux processus communiquants génèrent de nombreuses variables d'état TCP relatives à la connexion. Ceci met en jeu des tampons de réception et d'envoi, des paramètres de contrôle de congestion et des numéros de séquence et d'accusé de réception. Ces indications d'états sont indispensables à la bonne marche du service de transfert de données fiable de TCP. C'est seulement après l'établissement de la connexion entre les deux bouts que le transfert de données commence. En outre, TCP vérifie l'intégrité des données reçues en utilisant un "*checksum*" contenu dans l'entête de chaque paquet transmis.

TCP permet aussi à une application de détecter si son correspondant est déconnecté. Une connexion TCP assure un transfert de données en mode *duplex*. Les données échangées entre deux processus A et B peuvent circuler simultanément dans les deux sens, soit de A vers B, soit de B vers A. Une connexion TCP est également dite de *point-à-point* (*Unicast*), c'est-à-dire reliant un seul expéditeur à un seul destinataire, par contre TCP est incompatible avec le *Multicasting*.

Malheureusement, la fiabilité a un coût. Le protocole TCP doit transmettre plus d'informations en transmettant des acquittements, détectant des corruptions, puis en retransmettant des données. D'ailleurs, le récepteur est obligé de recevoir les paquets dans l'ordre de leur envoi par l'émetteur. Ceci veut dire que TCP peut arbitrairement retarder des paquets pour préserver l'ordre de transmission.

2.2.1.3 *User Datagram Protocol* "UDP"

Le protocole de transport le plus simple possible est celui qui étend le service de communication entre deux hôtes de façon à devenir un service de communication entre deux processus. De cette façon le protocole permet à plusieurs processus

sur chaque machine de partager le réseau. UDP, conformément à la RFC 768, est un exemple de ce type de protocoles. Il réalise le strict minimum de ce qui peut être exigé d'un protocole de transport.

Le protocole UDP n'implémente pas de contrôle de flots ni de fiabilité, il s'agit d'un simple démultiplexeur de messages émis par l'application d'un certain processus. Il assure que les messages sont corrects en calculant un "*checksum*" (le checksum est optionnel dans l'Internet courant mais obligatoire avec IPv6). Notons aussi que UDP n'établit pas une connexion entre l'envoyant et le recevant des paquets émis. Chaque paquet envoyé peut prendre un chemin différent en allant vers sa destination. Ni l'ordre des paquets, ni même la garantie de leur réception ne sont assurés par UDP. Mis à part ces services modestes UDP n'apporte absolument rien de plus que le protocole IP.

Bien que UDP puisse sembler à première vue un protocole faible qui n'assure aucune garantie, il a quand même plusieurs avantages. Le premier avantage est sa simplicité. Les paquets UDP ne contiennent aucune information qui garantit la fiabilité comme avec TCP, ce qui entraine un temps de traitement plus court pour l'envoyant et le recevant. Un autre avantage est le fait que UDP ne maintient pas l'illusion de "flots de données". Ceci veut dire que les paquets peuvent être transmis dès qu'ils seront envoyés par l'application au lieu d'être empilés derrière d'autres paquets qui attendent eux aussi leur envoi. De même, les paquets peuvent être livrés à l'application dès leur arrivée à destination au lieu d'être retardés dans le cas d'une perte de données. Enfin, comme il y a plusieurs systèmes d'exploitation qui imposent des limites sur le nombre de connexions TCP/IP autorisées, UDP/IP semble logiquement être plus approprié pour les systèmes de développement d'environnements virtuels distribués à grande échelle.

Cependant, UDP comporte un inconvénient qui peut rendre son utilisation un peu délicate dans certains systèmes. Lorsqu'une *socket* reçoit des données sur un port UDP donné, elle peut reçevoir n'importe quel paquet destiné à ce port en provenance de n'importe quel site même si ce dernier ne fait pas partie de l'application. Cet inconvénient peut entrainer des dangers surtout pour les applications qui ne savent pas traiter des données imprévues.

Le tableau 2.1 présente une comparaison rapide des deux protocoles TCP et UDP.

2.2.2 Méthodes de transmission de données

Il y a trois méthodes basiques pour transmettre les données sur un réseau : *Unicast*, *Broadcast* et *Multicast*. Les protocoles de couche Transport étudiés dans la section 2.2.1 ont pour fonction l'expédition de paquets entre un expéditeur et un destinataire uniques. Ces protocoles sont connus sous le nom de protocoles de transmission en point-à-point ou *Unicast*.

TCP	UDP
effectue un établissement de communication	pas d'établissement de communication
maintient des indications d'état de connexion	pas d'état de connexion
présente un en-tête de paquets de 20 octets	faible volume des en-têtes de paquets (8 octets)
système de contrôle de congestion	pas de système de contrôle de congestion
transmission point-à-point fiable	transmission non fiable "*best-effort*"

TAB. 2.1 – Les protocoles TCP et UDP

Or, certaines applications nécessitent l'expédition de paquets entre un ou plusieurs expéditeurs et un groupe de destinataires. Ce type de transmission est appelé transmission multidestinataires qui peut être du *Broadcast* ou du *Multicast* (Fig.2.2).

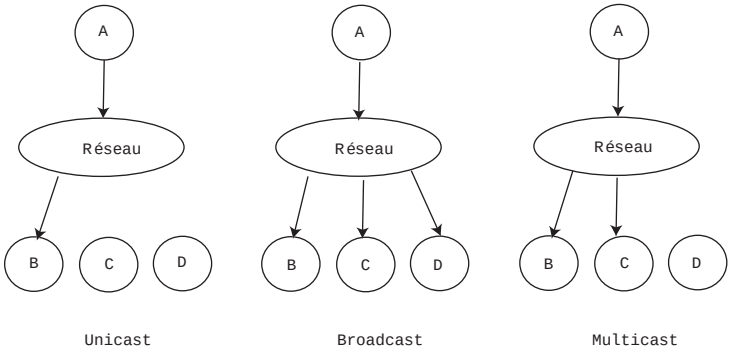

FIG. 2.2 – Exemples d'*Unicast*, de *Broadcast* et de *Multicast*

2.2.2.1 *Unicast*

Unicast est le terme employé pour décrire la communication d'une information qui est envoyée d'un point (processus ou machine) A à un autre point B. D'où le terme point-à-point qui est parfois associé à *Unicast* dans la littérature. La transmission *Unicast*, dans laquelle un paquet est envoyé d'une source unique à une destination unique, est toujours la forme prédominante de transmission sur les réseaux locaux LAN (*Local Area Network*) ainsi que sur *Internet*. Par exemple *HTTP*, *smtp*, *ftp* et *telnet* qui implémentent le protocole de transport TCP utilisent la méthode de transmission *Unicast*.

2.2.2.2 *Broadcast*

Broadcast est le terme employé pour décrire la distribution d'une information envoyée d'un point à tous les autres points d'un réseau. Dans ce cas-ci il y a juste

un expéditeur, mais l'information est envoyée à plusieurs récepteurs.

Un protocole *Broadcast* permet alors à un paquet d'être livré à tous les destinataires sur le réseau en une seule transmission, contrairement à *Unicast* qui nécessite autant de transmissions que de destinataires. Cette approche est plus utile pour les environnements virtuels distribués que celle d'*Unicast*. L'inconvénient du protocole *IP Broadcast* est qu'il ne permet pas l'envoi à un sous groupe d'utilisateurs. Il exige que tous les hôtes examinent le paquet même si ce dernier ne leur est pas destiné. Cela provoque des dégradations de performance pour les hôtes qui réalisent ce test à cause de l'interruption des opérations qui étaient en train de se dérouler au niveau du système d'exploitation.

2.2.2.3 *Multicast*

Multicast est le terme employé pour décrire la communication d'une information qui est envoyée d'un point à un ensemble d'autres points. MTP (*Multicast transport protocol*) est le premier protocol *Multicast* qui soit apparu (RFC 1301). Les autres protocoles *Multicast* s'appuient en grande partie sur celui-ci. Ce protocole n'a pas pour vocation d'être rapide, il est surtout fait pour ne pas saturer le réseau.

Le protocole de communication *Multicast* offre des services du type un-à-plusieurs et plusieurs-à-plusieurs pour des applications comme la téléconférence et les simulations distribuées qui ont besoin de communiquer avec plusieurs sites simultanément. Par exemple, une téléconférence utilisant le *Multicast* permet à un site d'envoyer des séquences audio et vidéo à une liste d'utilisateurs.

L'idée dans *Multicast* est d'envoyer le message à une adresse *IP Multicast* spécifique[2], ce message ne sera livré qu'aux abonnés explicites à cette adresse. Cette technique facilite la construction des environnements virtuels à grande échelle en permettant l'échange de différents types de données en utilisant différentes adresses *Multicast*.

Certaines versions du protocole *Multicast* sont fiables, d'autres non. Parmi les versions fiables on peut citer LRMP (*Light-weight Reliable* Multicast *Protocol*) [3] conçu et implémenté à l'INRIA Rocquencourt par Tie Liao et SRM (*Scalable Reliable* Multicast) réalisé par Sally Floyd et Al. [FJL+96].

Cependant, *Multicast* souffre de quelques limitations lors de son utilisation pour des environnements distribués sur *Internet*. Ces limitations sont souvent dues au fait que certains routeurs anciens ne sont toujours pas capables de gérer des souscriptions *Multicast*. Actuellement, les routeurs qui en sont capables communiquent directement entre eux en canalisant les données en dehors des anciens

[2]L'adresse IP doit être compris entre 224.0.0.0 et 239.255.255.255

[3]Internet draft: http://webcanal.inria.fr/lrmp

routeurs. Ce réseau virtuel des routeurs *Multicast* s'appelle le MBONE (*Multicast Backbone*) [4].

2.3 Communication

Une des parties les plus importantes d'une plate-forme de développement d'environnements virtuels distribués est celle qui a la charge de distribuer l'environnement virtuel sur les différents sites. Cette partie doit garantir la communication entre les sites en gardant toujours un certain niveau de fiabilité. Plusieurs facteurs peuvent affecter la communication entre les sites, comme la bande passante, la latence et la fiabilité de la transmission.

2.3.1 Bande passante

La bande passante est la capacité maximale de débit sur une liaison donnée. La bande passante joue un rôle majeur dans la communication inter-sites car c'est elle qui détermine la taille et la richesse d'un environnement virtuel distribué. Plus le nombre des participants s'accroit, plus on a besoin d'une bande passante importante. Ceci est lié à différentes raisons. Tout d'abord, chaque nouvel utilisateur doit recevoir l'état initial de l'environnement ainsi que ses mises-à-jour. Il doit pouvoir aussi introduire ses modifications de l'environnement partagé à distribuer chez les autres participants.

En fait, les systèmes de développement d'environnements virtuels distribués nécessitent un débit important pour pouvoir supporter les utilisateurs multiples et les échanges vidéos et audios. Dans les réseaux LAN ce problème est résolu grâce au très haut débit offert par les nouvelles technologies, mais par contre le nombre de participants est limité. Quant aux réseaux WAN le débit est plus limité que celui offert par les LAN mais ils permettent un nombre d'utilisateurs beaucoup plus important.

2.3.2 Latence

La latence est un facteur qui affecte directement la communication dans les environnements virtuels distribués. La latence peut dégrader la communication en retardant la réception des messages émis sur le réseau (Fig. 2.3). Si un environnement distribué doit imiter le monde réel, il doit agir en temps réel dans les limites de la perception humaine. En d'autres termes une plate-forme de développement d'environnements virtuels distribués doit délivrer des paquets avec une

[4]Frequently Asked Questions on the Multicast Backbone (Casner, S.) : venrera.isi.edu:/mbone/faq.txt

latence minimale (inférieure à 100ms) et doit générer une structure graphique 3D à une fréquence allant de 30 à 60 Hz pour garantir l'illusion de la réalité [Wlo95].

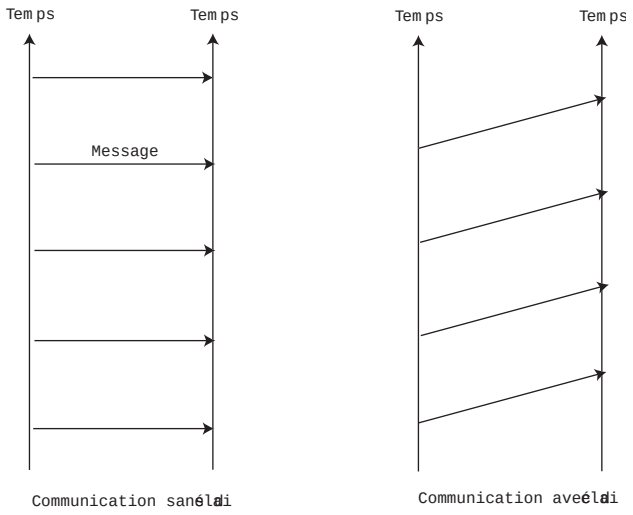

FIG. 2.3 – Communication sans et avec latence

La variation de la latence en fonction du temps est aussi un facteur qui perturbe le fonctionnement des application interactives. Sawler a noté dans [Saw91] qu'une latence variable au cours du temps est aussi importante que la latence elle même, particulièrement dans les tâches qui exigent un niveau de synchronisation élevé. Le problème de latence est devenu un défi pour les systèmes qui utilisent les réseaux WAN à cause de l'affectation de la transmission par plusieurs types de retard : des longs chemins, des "switchs", des routeurs, traitement de paquets, files d'attentes... D'après [Tan90], les plus importants de ces retards sont :

- le temps de traitement à chaque nœud : le temps de traitement est le temps requis pour l'examen de l'en-tête d'un paquet et la détermination de son destinataire. Le temps de traitement peut être déterminé par le temps requis pour la vérification des erreurs dans le paquet lors de son transfert.
- le temps d'attente : le temps d'attente correspond au temps nécessaire pour faire parvenir le paquet jusqu'à la liaison qui l'acheminera vers le prochain routeur. Le temps d'attente d'un paquet dépend du nombre de paquets qui le précèdent dans le routeur.
- le temps de transmission : le temps de transmission correspond au temps requis pour transmettre tous les bits du paquet sur la liaison.
- le temps de propagation : le temps de propagation est le temps nécessaire au trajet entre l'origine de la liaison et le routeur destination. Un bit se propage à la vitesse de propagation d'une liaison, qui est déterminée par son support physique.

En plus des différents retards sur le réseau, un paquet peut aussi être perdu. En effet, les files d'attente sont d'une capacité limitée. Un paquet peut tout à fait rencontrer une file d'attente complète et fermée à son arrivée dans le routeur. En l'absence d'espace libre pour stocker un paquet, le routeur l'abandonne et le paquet est dit perdu.

La latence peut être minimisée en utilisant des liens spéciaux qui utilisent des protocoles tels que *Reservation Protocol* [Dee93], des améliorations dans la technologie des routeurs, l'utilisation des interfaces rapides. Cependant elle ne peut pas être éliminée complètement.

Dans le contexte d'une plate-forme de développement d'environnements virtuels, la continuité d'une scène et le contrôle de sa fluidité ne sont plus possibles si la latence devient importante. Par conséquent, les interactions deviennent de plus en plus difficiles à réaliser. Plus de détails sur ce problème sont présentés dans le chapitre 4.

2.3.3 Fiabilité

Un système est dit fiable s'il peut s'assurer que les données envoyées sont reçues correctement. Cela peut parfois nécessiter un réenvoi des données.

Pour garantir la fiabilité, l'architecture des réseaux dans les couches basses utilise des acquittements. Cette technique implique malheureusement un grand délai dans le cas d'un réseau WAN ou d'un système distribué à grande échelle. De plus, quelques protocoles de transport comme TCP par exemple utilisent des mécanismes de contrôle d'encombrement qui ne conviennent pas pour le trafic en temps réel, parce que ces techniques diminuent le taux de transfert des paquets dans le cas de détection d'encombrement. Le protocole de *Multicast* fiable n'est pas très utilisable pour les grands groupes qui utilisent le même environnement virtuel à cause du nombre des acquittements et des retransmissions nécessaires afin de garantir que tous les utilisateurs aient reçu les mêmes données.

Les problèmes posés suite à la fiabilité ne veulent pas dire que les chercheurs n'essaient pas de développer un service de *Multicast* fiable qui puisse le mieux répondre aux exigences des systèmes de développement d'environnements virtuels. Par exemple, la communication dans DIVE (*Distributed Interactive Virtual Environment*) [CH93], développé à l'institut suédois de l'informatique, est assurée par ISIS, un système développé par Ken Birman qui utilise un *Multicast* fiable. Brian Whetten et Simon Kaplan ont développé RMP (*Reliable* Multicast *Protocol*) qui est aussi un protocole de *Multicast* fiable basé sur l'utilisation des acquittements négatifs. L'inconvénient de cette méthode est l'explosion exponentielle du nombre d'acquittements négatifs qui provoque l'encombrement du réseau. Cela arrive lorsque tous les récepteurs d'un groupe envoient simultanément un acquittement négatif qui, ajouté à l'encombrement partiel du réseau, a pour conséquence

la perte de plus de paquets. Ceci entraine encore plus d'acquittements négatifs sont envoyés, ce qui mène à un encombrement total du réseau.

On peut conclure que le choix d'un seul protocole *Multicast* (fiable ou non) ne satisfait pas tous les besoins d'un environnement virtuel distribué. Un tel environnement nécessite l'utilisation des deux protocoles, l'un fiable et l'autre non fiable, à utiliser selon les besoins pour avoir un compromis acceptable.

2.4 Distribution de données

Parmi les décisions à prendre au cours de la conception d'un environnement virtuel distribué, la plus critique est de déterminer où l'on doit mettre les données concernant l'état de l'environnement virtuel et de ses objets. Il y a plusieurs façons de distribuer les données entre les différents sites faisant partie du même environnement virtuel. Nous les détaillons dans la suite.

2.4.1 Monde homogène répliqué

Certaines plates-formes de développement d'environnements virtuels distribués initialisent l'état de chaque site participant à une simulation virtuelle avec une même base de données. Cette base contient des informations concernant le terrain, le modèle géométrique, les textures, et le comportement de tout ce qui est représenté dans ce monde. Ce qui est communiqué entre les utilisateurs de l'environnement virtuel est l'état des objets qui ont changé de valeurs, comme par exemple des véhicules en mouvement ou des événements comme une collision entre deux objets. L'avantage de cette approche est que le volume des messages envoyés est faible, mais l'inconvénient est que la taille de la base de données sur tous les sites croît considérablement si le contenu de l'environnement virtuel est important. Cette méthode est utilisée dans SIMNET [CDG+93, MT95].

2.4.2 Monde centralisé partagé

D'autres systèmes comme VISTEL (*Virtual Space Teleconferencing System*) utilisent une seule base de données partagée par les sites impliqués dans une simulation distribuée [OKKT93]. VISTEL est un système de téléconférence qui affiche des modèles 3D pour la représentation de la conférence offerte à chaque participant. Les changements de forme des modèles sont envoyés au serveur central en utilisant des messages et ensuite ils sont redistribués aux différents sites (Fig. 2.4). Un seul utilisateur à la fois peut modifier la base de données. L'avantage est d'abord la simplicité de l'approche, puis la garantie d'un environnement cohérent entre les différents sites. L'inconvénient est la possibilité d'un goulot d'étrangle-

ment réseau sur le serveur central car ce dernier doit envoyer les mises-à-jour à tous les sites à la fois.

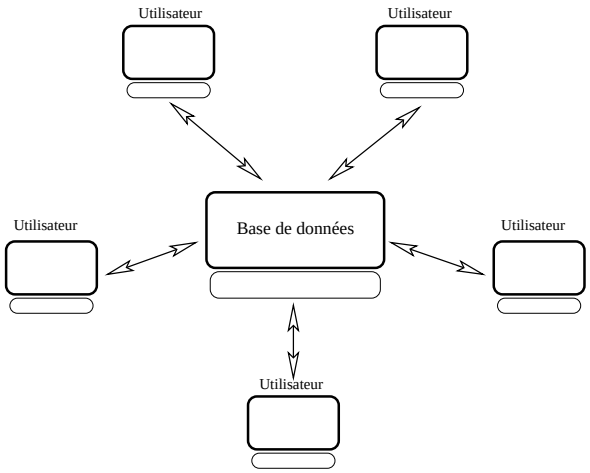

FIG. 2.4 – Base de données centralisée

2.4.3 Monde distribué

Plusieurs plates-formes logicielles d'environnements virtuels distribués utilisent cette architecture, comme DIVE où la base de données est entièrement distribuée sur tous les sites qui participent à une simulation. L'avantage de l'architecture distribuée est son extensibilité. En effet cette approche évite les inconvénients des deux méthodes précédentes. Par contre, l'inconvénient de cette approche est le coût de communication élevé pour garantir la fiabilité. Deux techniques sont utilisées pour la communication entre les différentes bases de données : point-à-point ou client-serveur.

La communication point-à-point exige l'établissement d'une connexion ou d'une voie de communication de chaque nœud vers tous les autres dans le réseau. Par exemple pour N nœuds on a $N(N-1)$ connexions virtuelles (Fig. 2.5).

Une autre technique consiste à utiliser une variante du modèle client-serveur. De cette façon la base de données est partitionnée entre les différents clients et la communication se fait via un serveur central. Par exemple dans le système BrickNet, à chaque fois qu'une entité bouge dans l'environnement virtuel, sa base de données est mise à jour par un ORB (*Object Request Broquer*) sur un serveur qui sait quel client maintient cette partie de l'environnement virtuel [Sin94]. Cependant, des études [Par94, SB89] ont noté que l'utilisation de RPC (*Remote Procedure Call*) dans les architectures client/serveur convient mal aux réseaux haut débit. La raison est l'attente d'une réponse pour chaque message envoyé

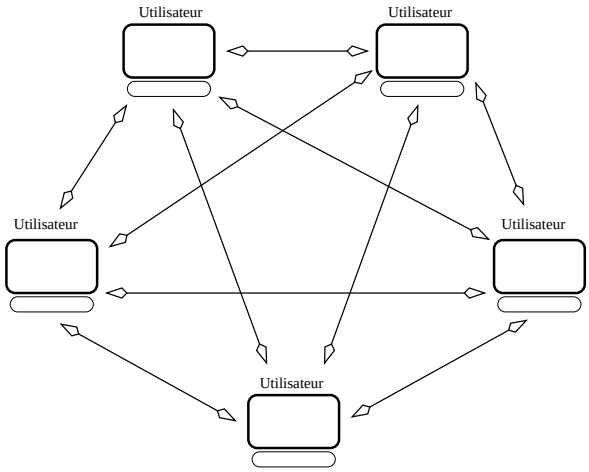

FIG. 2.5 – Communication point-à-point

(principe d'invocation synchrone), si donc le délai augmente, RPC devient coûteux surtout dans le cas d'interaction en environnement virtuel.

2.4.4 Synthèse

Le modèle de données utilisé dans un environnement virtuel distribué peut être répliqué ou partagé (Fig. 2.6). Avec un modèle répliqué, les participants initialisent l'environnement à partir d'une base de données commune, ensuite ils font transiter les mises-à-jour sur le réseau. Les mises-à-jour sont normalement diffusées en utilisant le *Broadcast* ou le *Multicast*. L'environnement est plus ou moins cohérent selon le mécanisme de fiabilité et de détection de perte utilisé. De plus, même si un mécanisme de transport fiable est utilisé, les participants peuvent subir différentes latences qui causent par la suite quelques inconsistances. Avec le modèle partagé, l'état de l'environnement virtuel demeure dans une ou plusieurs bases de données. Dans l'approche centralisée, toutes les communications passent par la base de données centrale. Cette approche n'est évidemment pas convenable pour des environnements distribués qui comportent beaucoup de participants à cause du goulot d'étranglement potentiel au niveau réseau sur le serveur central. L'approche utilisant une base de données distribuée est plus extensible en terme de participants par contre elle est soumise au même problème de cohérence que dans le cas d'un modèle répliqué.

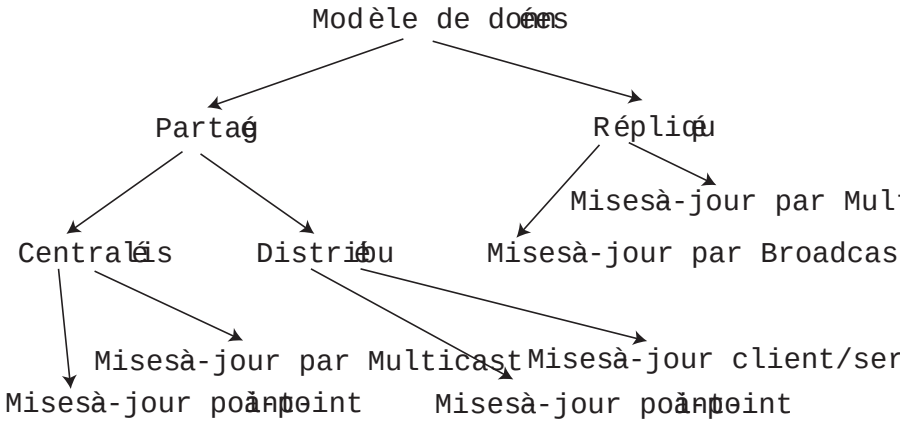

FIG. 2.6 – Modèles de données et types de mises-à-jour associées

2.5 *Dead-Reckoning*

La famille des techniques dites de *dead-reckoning* sert essentiellement à synchroniser l'état des objets sur différents sites. Par exemple, au lieu d'envoyer et de recevoir des messages de synchronisation sur le réseau pour décrire le mouvement d'un objet, une formule de prédiction de sa trajectoire sera utilisée [SZ99, Zyd99]. La formule est maintenue sur chaque site qui participe à la simulation. Les objets ne doivent envoyer des messages que lorsque les autres simulateurs du réseau ne peuvent plus prédire fidèlement leur état. Tous les simulateurs gèrent des copies fantômes de leurs objets de façon à comparer leur état courant et la prédiction qui peut en être faite sur les autres sites grâce aux dernières informations transmises (vitesse et position par exemple).

Prenons par exemple le jeu "*space dogfight*" cité par Gossweiler dans [GLK94]. N joueurs partagent un environnement virtuel distribué, et où chaque joueur est représenté par une navette qu'il contrôle (Fig. 2.7). Pour réduire la communication, un joueur X envoie la position de sa navette et la vitesse de son déplacement aux autres joueurs. Les autres utilisent la vitesse pour extrapoler la prochaine position de X.

Selon cette approche, chaque participant stocke une copie de son propre modèle, nommée modèle *fantôme* auquel il applique l'extrapolation. Si la différence entre la position réelle et celle calculée pour le fantôme est supérieure à un certain maximum prédéfini, le joueur X envoie sa position ainsi que sa vitesse aux autres joueurs afin qu'ils puissent corriger leurs copies de X. Notons que X n'envoie des messages sur le réseau que si la différence de position entre la valeur réelle et celle extrapolée est importante.

La méthode la plus simple pour prédire la position d'un objet fantôme est basée sur les dérivées polynomiales. La dérivée est prise en compte dans l'équation

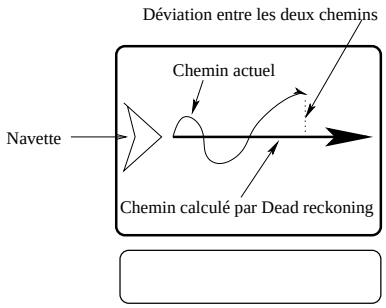

FIG. 2.7 – Prédiction par *dead-reckoning*

du premier ordre :

$$X(t+dt) = X(t) + V(t).dt$$

et l'accélération apparait dans les équations du second ordre :

$$V(t+dt) = V(t) + A(t).(dt)$$

$$X(t+dt) = X(t) + V(t).dt + A(t).(dt)2$$

Au lieu d'utiliser un schéma de prédiction fixe, les utilisateurs peuvent dynamiquement choisir entre les équations du premier ou second ordre en se basant sur l'historique du fantôme. L'équation du premier ordre est choisie si l'accélération est négligeable ou si l'information sur l'accélération change très fréquement. Un tel schéma est appelé *Hybrid Polynomial Prediction*. D'autres équations d'ordre plus élevé existent, mais elles sont rarement utilisées.

2.6 Protocoles dédiés

La plupart des travaux relatifs aux environnements virtuels distribués ont commencé par employer des formes de communication basées sur le *Broadcast* ou l'*Unicast*. Par exemple *MR ToolKit Peer Package*, qui est utilisé pour créer des applications de réalité virtuelle distribuée à travers l'*internet*, utilise *Unicast* pour les communications entre les applications. Cependant, ces schémas sont insuffisants pour des grands groupes. De plus, le *Broadcast*, qui est utilisé dans SIMNET et la plupart des implémentations DIS, ne convient pas dans le cas de plusieurs sites interconnectés via un réseau car cela va encombrer le réseau par des données non demandées.

Pour toutes ces raisons, les chercheurs ont proposé différentes idées concernant l'utilisation d'architectures hybrides incluant plusieurs protocoles, notamment le

Multicast pour la communication dans les environnements virtuels. L'utilisation du *Multicast* pour les mises à jour réduit les dégâts provoqués par le trafic des messages. Dans cette section, nous allons citer quelques architectures et protocoles dédiés à la communication dans les environnements virtuels partagés.

2.6.1 *Simulator Networking* : SIMNET

SIMNET est une plate-forme militaire de développement d'environnements virtuels distribués destinée à former des petites unités séparées (véhicules, hélicoptères, postes de commandes) pour combattre en équipe. Le but de SIMNET est de faciliter les phases d'entraînement et de formation à un coût bien au-dessous des dépenses occasionnées par l'utilisation de vrais véhicules et matériels militaires.

L'approche dans SIMNET est basée sur trois éléments clés [MT95] :

- **architecture objet/événement** : cette approche modélise le monde comme étant une collection d'objets, appelés aussi entités, qui interagissent entre eux par envoi d'événements. Les véhicules, les armes et les matériaux sont les objets, les messages transmis entre les différents objets sont les événements.
- **nœuds autonomes de simulation** : SIMNET utilise une base de données répliquée sur chaque site participant. De cette façon les sites gardent un certain niveau d'autonomie. Des nœuds peuvent joindre ou abandonner une simulation indépendamment d'autres nœuds. La communication entre les nœuds se fait par l'envoi d'événements. Pour minimiser les communications, les mises à jour des états sont transmises seulement quand des objets changent de comportements.
- **l'algorithme "*Dead reckoning*"** : qui utilise l'extrapolation pour prévoir l'état d'un objet distant.

Le succès de SIMNET a incité les chercheurs à utiliser cette expérience pour développer un standard pour les simulations distribués. Le résultat a été DIS (*Distributed Interactive Simulation*), accepté comme standard ANSI/IEEE en 1993 et révisé en 1995.

2.6.2 *Distributed Interactive Simulation* : DIS

DIS découle directement de SIMNET. DIS et SIMNET partagent la même philosophie et le même objectif. DIS a été conçu pour supporter les interactions des participants dans l'environnement virtuel commun [HL95, SZaF96].

Le document *DIS Vision* [DIS94] décrit la mission de DIS ainsi : *La mission de DIS est de définir une infrastructure qui peut lier des simulations de différents*

types à différentes localisations. Cette infrastructure doit créer un monde virtuel complexe et réaliste pour la simulation des activités hautement interactives.

DIS a les mêmes élements basiques que le format d'échange de SIMNET (architecture objet/événement, autonomie des nœuds de simulation et Dead Reckoning). Le DIS standard (IEEE 1278) est basé sur le concept d'autonomie d'applications simulées au sein d'un environnement virtuel. Ces applications sont distribuées géographiquement et interconnectées par un réseau. Le standard comprend deux parties : la première spécifie les protocoles d'applications, la deuxième spécifie les services de communications. Les protocoles d'applications définissent de messages formatés appelés PDU (*Protocol Data Units*). Les PDUs sont un ensemble de paquets de données qui communiquent des informations spécifiques entre les sites de simulation. Le DIS standard définit 27 PDUs pour la communication d'événements entre les sites. Seulement 4 PDUs sont dédiés à l'interaction des entités avec l'environnement virtuel. Les autres PDUs sont utilisés pour la transmission des différentes données et actions nécessaires à l'apparition et la disparition d'entités, l'utilisation d'armes, les explosions de charges, la détection de collisions et le réapprovisionnement logistique des unités... (Fig. 2.8) .

Protocol Data Units (PDUs)

DIS 1.0 PDUs	Additional PDUs Proposed for DIS 2.0	Additional PDUs Considered for DIS 3.0
Entity State		
Fire	Emission	C3I
Detonation	Laser	Dynamic Terrain
Service Request	Transmitter	Weather/Atmosphere
Resupply Offer	Signal	Fidelity Controls
Resupply Received	Simulation management	Transfer Control
Resupply Cancel	Create Entity	Aggregate/Deaggregate
Repair Complete	Remove Entity	Instrumentation.
Repair Response	Start/Resume	
Collision	Stop[/Freeze	
	Acknowledge	
	Action request	
	Action Response	
	Data Query	
	Set Data	
	Data	
	Event	
	Message	

SOURCE: Office of Technology Assessment, 1995.

FIG. 2.8 – Liste des PDUs DIS

Cependant, un système de développement d'environnements virtuels général ne peut pas fonctionner selon la façon décrite dans DIS, car DIS reste très

spécifique au contexte d'une simulation militaire. En dehors de ce contexte DIS présente des limitations et par exemple ne convient pas pour simuler des environnements urbains car ce protocole ne décrit pas le comportement d'objets "non-militaire" (un oiseau n'a pas les mêmes spécifications qu'un avion de chasse).

2.6.3 *High Level Architecture* : HLA

Le protocole DIS s'occupe d'une façon particulière des simulations d'entrainement militaire pour les exercices au niveau tactique. Une nouvelle génération a été proposée pour élargir l'utilité de DIS en servant aussi des communautés de modélisation et de simulation en dehors de la plate-forme originale, d'où la naissance du successeur de DIS nommé tout d'abord DIS++ puis finalement HLA.

HLA constitue une architecture de haut niveau pour les simulation distribuées [DFW98]. HLA est fondée sur l'hypothèse qu'aucune simulation ne peut satisfaire tous les besoins. L'objectif de cette architecture est de faciliter la réutilisation de simulateurs élémentaires, de faciliter l'interopérabilité entre simulateurs distribués et enfin de réduire les coûts de modélisation et de simulation. Le vocabulaire HLA définit par le terme "fédéré" chaque simulateur élémentaire, et par le terme "fédération", un ensemble de fédérés interopérants. L'architecture HLA est décrite par des spécifications et un ensemble de règles définissant les responsabilités des fédérés et de la fédération. Les concepts basiques sont les suivants :

- **infrastructure temps réel RTI (*Real-Time Infrastructure*)** : l'implémentation d'un système d'exploitation distribué comme une couche de base qui fournit une interface de service commune aux simulations HLA.
- ***Object Model Template* OMT** : des formats standardisés utilisés pour décrire les objets.
- **Modèle d'Objet Fédéré FOM (*Federate Object Model*)** : une identification des classes d'objets essentielles, des attributs d'objets et des interactions. FOM ne contient pas d'informations à propos des objets actuels dans la simulation, mais seulement à propos des classes d'objets possibles.
- **Modèle d'Objet de Simulation SOM (*Simulation Object Model*)** : une spécification des possibilités intrinsèques qu'un simulateur peut offrir aux fédérations. Chaque SOM est un ensemble d'attributs et de paramètres. Le SOM est comparable à une description de classe, par conséquent, plusieurs fédérés peuvent partager un SOM donné s'ils ont des spécifications communes. Par exemple, tous les simulateurs d'avions F16 peuvent partager le même SOM.

Le taux de communication et du trafic réseau généré en utilisant le modèle HLA peut être réduit en s'abonnant à la RTI. Ceci est réalisable en respectant les deux mécanismes :

1. **publication** : chaque simulation doit enregistrer les objets et les attributs

qu'elle représente.

2. **abonnement** : chaque simulation doit enregistrer les attributs et les interactions qu'elle exige pour exécuter une tâche.

Publication et abonnement sont dynamiques et peuvent être modifiés pendant une session en cours. Ils utilisent le *Multicast* pour réduire la bande passante nécessaire.

L'utilisation de HLA est profitable par rapport à DIS pour les points suivants :
- HLA supporte davantage de catégories de simulations ;
- HLA n'est pas seulement un protocole mais aussi une architecture ;
- HLA est plus étendue que DIS en raison de la réduction de la bande passante et de l'utilisation de *Multicast* là où DIS utilise le *Broadcast* ;
- HLA sépare l'application de la couche de communication.

2.6.4 *Virtual Reality Transfer Protocol* : VRTP

VRTP [BZWM97] est le fruit du travail de Michael Zyda et Al. (Naval Postgraduate School) en coopération avec Michael Macedonia (Computer Graphics Inc.).

VRTP est conçu pour être un support pour le VRML (Virtual Reality Modeling Langage) de la même façon que l'http est la base de l'html. En d'autres termes VRTP peut être vu comme une extension du protocole http dans le but de répondre aux besoins des environnements virtuels distribués. L'http n'est plus suffisant pour gérer des objets 3D interactifs. L'idée est d'ajouter de nouveaux concepts en combinant plusieurs protocoles à la fois (http, DIS, *Multicast*,..) afin de faire naître un nouveau protocole qui peut gérer efficacement les besoins d'un environnement virtuel distribué

L'approche utilisée dans VRTP est qu'un site participant à un environnement virtuel peut prendre le rôle d'un client, d'un serveur, ou d'un *peer* selon le besoin. Il peut se présenter comme un client lorsqu'il examine les bases de données d'autres mondes, comme un serveur en répondant aux requêtes d'autres utilisateurs, et comme un *peer* en participant à une simulation regroupant des groupes de nombreuses entités actives qui se parlent à travers des canaux *Multicast*. VRTP joue donc le rôle d'une combinaison optimisée de plusieurs procoles déjà existants mais fonctionnant chacun à part afin de mieux répondre aux besoins d'une simulation d'un environnement virtuel.

2.6.5 *Distributed Worlds Transfer Protocol* : DWTP

DWTP est un protocole d'application pour les environnements virtuels partagés sur *Internet* [Bro98]. Il est basé sur les protocoles standards comme TCP/IP

et UDP/IP. Il permet le transport de différents genres de données : événements, messages, fichiers et flots de données.

Les événements sont utilisés pour assurer la cohérence entre les différentes copies de l'environnement virtuel. Les messages sont des événements prédéfinis comme par exemple quitter ou rejoindre un environnement virtuel partagé. Les fichiers sont plus grands que les messages et les événements. Un fichier peut être une description de scène ou d'un avatar et il exige un transport fiable. Les flots de données sont utilisés pour transmettre un flux continu de données tel qu'un flux de type audio ou video. Ce type de données ne nécessite pas une transmission fiable.

DWTP offre une interface pour les types de données que je viens de citer. La notion de base est celle des *démons* et des *participants*. Les démons ont pour rôle d'offrir des services aux participants d'un environnement virtuel. Contrairement à d'autres protocoles, DWTP utilise plusieurs démons pour répondre aux besoins des utilisateurs :

- **démon de fiabilité** : pour détecter un échec de transmission (pour les protocoles non fiables (UDP)).
- **démon de redémarrage** : pour récupérer les données perdues en offrant des connexions *Unicast*.
- **démon du monde** : pour transmettre le contenu d'un environnement virtuel aux nouveaux participants.
- **démon *Unicast*** : pour réaliser une architecture étendue qui peut même contenir des participants qui ne disposent pas d'un canal *Multicast*.

2.6.5.1 Transfert de données

La fiabilité dans DWTP est assurée grâce à trois démons : le démon de fiabilité, le démon de redémarrage, et le démon du monde. DWTP découpe les données à transmettre en petits paquets. La taille du paquet est indéterminée mais il y a une taille maximale pour chaque paquet. Chaque paquet a un identificateur unique qui permet aux destinataires de rassembler le message complet. Chaque paquet contient aussi le nombre total des paquets à transmettre et un drapeau indiquant si un transfert fiable est obligatoire ou pas. Dans le cas d'un transfert fiable, le démon de la fiabilité est utilisé pour détecter des pertes de transmission. Le démon de fiabilité utilise des acquittements positifs à la réception des paquets. Quand un participant détecte une perte de paquet, il contacte le démon de redémarrage qui garde une copie des paquets transmis à un certain instant. Le paquet perdu sera alors transmis en utilisant l'*Unicast*.

2.6.5.2 Les participants *Unicast*

Les participants qui ne sont pas capables de communiquer avec les autres utilisateurs en utilisant le *Multicast* peuvent communiquer par une connexion *Unicast* (TCP/IP, UDP/IP) offerte par le démon *Unicast*. Ce démon assure l'échange de données entre les participants utilisant l'*Unicast* et d'autres utilisant le *Multicast*.

2.6.5.3 Création d'environnements virtuels avec DWTP

Les démons du protocole DWTP sont complètement indépendants du support d'exécution d'environnements virtuels distribués, mais ils doivent quand même communiquer avec l'application ou le serveur de l'application simulée. DWTP dispose de deux interfaces pour assurer cette communication : l'interface serveur et l'interface client. Les démons reçoivent les messages concernant un environnement virtuel via son interface serveur. L'interface client est utilisée pour qu'un utilisateur puisse se connecter sur les sites d'autres participants ou envoyer des requêtes de descriptions des autres utilisateurs.

2.6.6 *Real-Time Transport Protocol* : RTP

RTP [SCFJ96] est conçu pour le trafic temps réel sur Internet ainsi que sur Intra-net (RFC 1889). RTP inclu des services de détection de perte, d'adaptation du taux de transfert, de séquencement de données, d'identification du source, ainsi que d'autres informations basiques. RTP peut servir au transport de formats de fichiers courants tel que PCM, GSM et MP3 pour les données audio, et MPEG et H.263 pour les données vidéo.

RTP peut agir au dessus de n'importe quel genre de protocole réseau. C'est un protocole indépendant qui ne dépend d'aucune information en provenance des couches basses du modèle réseau.

Le protocole RTP est implémenté actuellement au dessus d'UDP mais il peut être utilisé au dessus de n'importe quel type de paquets réseau (ATM ou ISDN).

En effet, RTP comporte deux protocoles différents : le RTP qui transporte les données, et les RTCP (*Real-time Transfer Control Protocol*) qui échangent des méta-informations à propos de la session courante. Ces deux protocoles opèrent sur deux canaux différents pour assurer une transmission du type "*best-effort*". Ceci veut dire que les paquets peuvent être perdus sans être retransmis. Tous les membres sur le canal RTCP envoient des messages réguliers à propos de la quantité de données émises (s'il y en a), et des données reçues. Ces informations seront utilisées pour détecter la perte et demander par la suite à l'envoyant d'adapter le taux de sa transmission par rapport à la perte actuelle.

Ces dernières années RTP est devenu le protocole utilisé pour le transport d'audio et vidéo sur *Internet*. L'une des raisons de ce succès est sa flexibilité vu

qu'il peut être utilisé pour différents formats de paquets. Ce succès a encouragé des chercheurs à utiliser ce protocole pour d'autres genres d'applications qui ne sont pas forcément liées à la transmission d'audio ou vidéo. M. Mauve et Al. ont proposé dans [MHKE99] que RTP puisse servir de base pour un protocole de transport de médias interactifs comme les tableaux blancs partagés par exemple.

2.6.7 *Interactive Sharing Transfer Protocol* ISTP

ISTP [WAS97] a été développé dans le contexte spécial du système de développement d'environnement virtuel SPLINE [WAB+97]. Cependant, ISTP n'est pas particulièrement attaché à SPLINE et vice-versa, *i.e.* ISTP peut être utilisé par d'autres architectures de systèmes de développement d'environnements virtuels. SPLINE à son tour peut utiliser d'autres couches de transport.

Le but d'ISTP est de communiquer les informations concernant les objets dans un environnement virtuel partagé. L'environnement virtuel selon SPLINE peut contenir des objets divers qui ont des caractéristiques différentes (des objets de décors, des objets interactifs, de l'audio ...). Pour répondre aux besoins particuliers de chaque type d'objets, ISTP contient cinq sous-protocoles :

1. **sous-protocole de connexion** : utilisé pour établir des connexions TCP entre deux processus ISTP.
2. **sous-protocole de transmission d'état d'objets** : utilisé pour communiquer l'état d'objets d'un processus ISTP. De telles mises-à-jour peuvent être envoyées par une connexion point-à-point ou par UDP *Multicast*.
3. **sous-protocole de flots de données audio** : utilisé pour la communication de flots de données de type audio (basé sur RTP).
4. **sous-protocoles de communication des *locales*** [5] : ce sous-protocole est considéré comme le noyau d'ISTP, il permet le partage d'informations à propos des objets dans le *modèle du monde* [6] .
5. **sous-protocoles de communication basé sur le contenu** : pour communiquer des informations concernant les *beacons* [7] .

Les deux derniers protocoles sont basés sur les trois autres. ISTP ne fournit pas la possibilité de communiquer de flot de données de type vidéo.

ISTP est un protocole totalement distribué, chaque processus ISTP joue le rôle d'un client et d'un serveur à la fois. C'est un protocole hybride qui est construit au dessus d'autres protocoles : TCP, UDP, RTP, et HTTP.

[5] Voir 3.2.2 pour la définition des locales.
[6] Voir 3.2.2 pour la définition du modèle du monde.
[7] Voir 3.2.2 pour la définition des beacons.

2.7 Synthèse

La couche de communication joue un rôle capital dans le fonctionnement d'un système de développement d'environnement virtuel distribué. Il y a des systèmes qui intègrent leur propre couche de communication, d'autres qui utilisent des couches existantes. L'environnement virtuel a besoin de faire transiter différents types d'informations à des priorités différentes. Les protocoles classiques de transport ne peuvent plus satisfaire aux besoins croissants de la réalité virtuelle en terme de latence et de bande passante et les architectures classiques ne suffisent plus pour garantir le temps réel.

Dans ce chapitre nous avons discuté des environnements virtuels distribués dans le contexte de leur relation avec la couche de communication. Nous avons détaillé comment le modèle de distribution de données et le choix du protocole utilisé influencent la simulation de tels environnements. Seulement quelques nouveaux protocoles de communication dédiés à la réalité virtuelle ont été détaillés dans ce chapitre parce qu'il aurait été trop fastidieux de les présenter tous. Chaque protocole offre une combinaison d'avantages et d'inconvénients. Le choix d'un protocole dépend essentiellement des besoins particuliers de l'environnement virtuel à déployer.

Les environnements virtuels sont partiellement ou totalement répliqués sur chaque machine participante afin de réduire le coût de communication. Dans ce cas, seulement les mises-à-jour ou les changements d'états d'un objet répliqué seront envoyés sur le réseau à tous les participants. Les délais provoqués par la vitesse de transmission réseau ajoutés au temps de calcul entrainent que les mises-à-jour mettent du temps avant d'aboutir à leurs destinations. Ce délai brise à son tour la cohérence entre les différents réplicas distribués. Cette incohérence peut passer inaperçue dans le cas d'un réseau LAN parce que le délai est faible, mais dans le cas d'un WAN c'est difficile voire impossible d'en cacher les conséquences.

Si un problème de transmission de données surgit sur le réseau, ce problème va directement affecter le fonctionnement de l'interaction à distance. Les types de problèmes les plus fréquemment envisagés sont le délai de transmission et les coupures qui interviennent de temps en temps. Une coupure réseau entraine différentes sortes de perturbations pour une session de simulation d'un environnement virtuel distribué et partagé.

Au niveau interactivité, les interactions d'un utilisateur agissant sur des objets qui résident sur un site déconnecté ne seront pas vues par le reste du monde. À son tour, cet utilisateur ne pourra plus percevoir les interactions des autres.

Les conséquences des problèmes provoqués par la couche de communication sont dans certains cas inévitables. Les chercheurs ont étudié la possibilité de réduire les messages transmis sur le réseau pour réduire par la suite l'impact du réseau sur le comportement d'une plate-forme de développement d'environnements

virtuels distribués. Plusieurs méthodes ont été proposées et réalisées au sein de quelques plates-formes logicielles de développement d'environnements virtuels. Ces méthodes sont citées et étudiées dans le chapitre suivant.

Chapitre 3

Méthodes de réduction des communications

3.1 Introduction

Les plates-formes de développement d'environnements virtuels distribués doivent respecter de nombreuses contraintes pour pouvoir offrir la richesse et les fonctionnalités demandées par les utilisateurs. Il est facile d'imaginer un environnement virtuel comportant des milliers d'utilisateurs et offrant toutes sortes de services de façon à rendre cet environnement le plus réaliste possible. Sa réalisation, par contre, reste beaucoup plus compliquée. Avant qu'un tel environnement ne soit réalisé, un certain nombre de problèmes doivent être surmontés.

Tant que les ressources de calculs sont limitées, il est évident que certains problèmes vont surgir une fois que le nombre d'utilisateurs dépasse une certaine limite. L'infrastructure de communication, elle aussi, même dans les meilleurs des cas, pourrait engendrer quelques problèmes insurmontables techniquement (comme le délai et la limitation de la bande passante discutés dans le chapitre précédent). Quelques plates-formes de développement d'environnements virtuels distribués fournissent des mécanismes spéciaux pour atténuer les effets indésirables en provenance de la couche de communication ou de la limitation des ressources. Ces sytèmes deviennent hybrides suite à l'intégration de plusieurs fonctionnalités allant de la gestion de leur propre couche de communication (comme on l'a vu dans le chapitre précédent) jusqu'à l'introduction des notions de partitionnement de l'environnement virtuel en plusieurs groupes ou, en d'autres termes, en plusieurs zones d'intérêt. Ces systèmes complexes comportent de plus en plus de sous-systèmes dédiés à la gestion des différentes fonctionnalités. Dans ce chapitre nous allons détailler quelques uns de ces services complexes, qui nous semble être les plus importants, comme les "*zones d'intérêt*" et la "*migration*" au sein d'une plate-forme de développement d'environnements virtuels distribués.

3.2 Gestion des zones d'intérêt

La gestion des zones d'intérêts ou AOI (*Area Of Interest*) a pour objectif de réduire le nombre des destinataires d'un message. Au lieu de partager tous les éléments d'un environnement virtuel partagé, on réduit le partage à un sous-ensemble d'objets en tenant compte de l'intérêt de chaque site. Les objets partagés seront filtrés selon des critères spatiaux ou fonctionnels. Par exemple, dans un environnement virtuel un avatar peut n'être intéressé que par les objets localisés à sa proximité (critère spatial), ou peut être encore par des objets vérifiant certaines propriétés applicatives (critère fonctionnel).

Selon le principe des zones d'intérêt, un environnement virtuel est partitionné selon des *auras*. Une *aura* est une notion qui définit une sphère d'intérêt associée à un utilisateur [BBFG94]. Dans les sections suivantes, nous allons détailler cette notion d'*aura* ainsi que son implémentation au sein de la plate-forme de développement d'environnements virtuels MASSIVE. D'autres notions dérivées de celle de l'*aura* sont aussi détaillées ainsi que les plates-formes qui les implémentent.

3.2.1 Aura dans MASSIVE

MASSIVE (*Model, Architecture and System for Spatial Interaction in Virtual Environments*) [GB95] est un prototype d'implémentation du modèle spatial. Le but principal de MASSIVE est de supporter le plus grand nombre possible d'utilisateurs et de permettre à des systèmes hétérogènes d'interagir entre eux.

Différents environnements virtuels peuvent être interconnectés à travers MASSIVE. Chaque environnement peut être habité par plusieurs utilisateurs qui interagissent via différentes interfaces. L'interface graphique rend les objets visibles dans un espace 3D et permet aux utilisateurs de naviger dans l'espace avec six degrés de liberté. L'interface auditive donne la possibilité d'entendre d'autres objets et permet aussi d'avoir des conversations en temps réel. Enfin l'interface textuelle permet l'échange de textes.

Concernant la communication, contrairement aux systèmes utilisant l'approche base de données partagée, MASSIVE utilise l'approche *pair-à-pair*.

3.2.1.1 Modèle spatial d'interaction

Le modèle spatial utilise les propriétés de l'espace comme un médiateur pour l'interaction. De cette façon, des objets peuvent naviguer dans l'espace pour former des sous-groupes et gérer des conversations entre eux. Pour aboutir à cet objectif, il faut introduire quelques abstractions comme *espace*, *objet*, *conscience*, *focus*, et *nimbus* .

Le concept fondamental est l'espace lui même. L'espace est défini comme étant un espace "métrique" où l'on peut mesurer la position et l'orientation. L'espace

est habité par des objets qui peuvent représenter des gens, des informations, ou peut être une représentation d'autres ordinateurs.

Chaque interaction entre des objets se produit à travers un medium. Un medium représente un moyen de communication qui est typiquement auditif, graphique, ou texte.

3.2.1.2 Aura

Une des difficultés liées aux environnements virtuels distribués est de déterminer quels sont les objets qui sont capables d'interagir entre eux à un moment donné. La notion d'*aura* va permettre de résoudre ce problème.

L'*aura* est un sous-espace qui limite la présence d'un objet à l'intérieur d'un certain medium qui facilite l'interaction. Les objets portent leur *auras* avec eux quand ils bougent dans l'espace. Quand deux *auras* entrent en collision, l'interaction entre ces objets dans le medium devient possible. Quand de telles collisions se produisent, l'environnement effectue les actions nécessaires pour mettre ces objets en contact entre eux. Ainsi, l'*aura* est une technologie fondamentale pour permettre l'interaction. Chaque objet peut avoir plusieurs auras pour des modes d'interactions différents (visuel, auditif...). Un objet ne peut être vu que lorsque son *aura* graphique entre en collision avec l'*aura* graphique de celui qui désire le voir. De même il ne peut être entendu que lors de la collision des auras auditifs.

3.2.1.3 Focus, nimbus, et conscience

Les objets ont eux mêmes la charge de contrôler leur interactions. Cela se fait selon les niveaux de conscience entre les objets. Le degré de conscience entre deux objets ne doit pas être obligatoirement symétrique. Par exemple un objet A peut être conscient de l'existence d'un objet B sans que ce dernier ait conscience de l'existence de A. Le niveau de conscience entre les objets est manipulé via le "*focus*" et le "*nimbus*".

Le focus et le *nimbus* sont deux sous-espaces selon lesquels un objet choisit d'accentuer ou d'atténuer sa présence ou son attention. Plus précisément, si on considère deux objets A et B, plus l'objet B est placé dans le focus de A, plus A est conscient de l'existence de B. Par contre, plus B est dans le *nimbus* de A, plus il est conscient de l'existence de A.

La notion de *focus* est intuitive, mais celle de *nimbus* nécessite plus d'explication. Un *nimbus* est un sous-espace dans lequel un objet peut mettre quelques uns de ses aspects pour être vus par les autres. Ces aspects peuvent être sa présence, son identité, ses activités, ou bien une combinaison de tout cela. Donc le *nimbus* permet aux objets d'influencer d'autres objets pour être par exemple vus ou entendus.

3.2.2 Locale dans SPLINE

Spline (*Scalable Platform for Large Interactive Network Environments*) est une plate-forme logicielle offrant la possibilité d'implémenter des environnements virtuels coopératifs [WAB+97, BWA96]. L'architecture de Spline est centrée sur ce qu'on appelle un *modèle de l'environnement virtuel* qui joue le rôle d'un médiateur pour les interactions. Dans Spline les applications ne communiquent pas directement entre elles, mais à travers le modèle de l'environnement virtuel. Cette approche permet à une application d'être conçue sans se soucier des problèmes d'établissements de la communication.

Le modèle de l'environnement virtuel spécifie les objets qui existent cet environnement, leurs emplacements, leurs fonctions, et les sons qu'ils produisent. Le modèle de l'environnement virtuel ne contient pas l'historique des informations mais plutot une image de l'environnement virtuel à l'instant courant. Cette vision de l'environnement virtuel change au fur et à mesure que l'environnement virtuel change.

Les applications peuvent ajouter, supprimer ou modifier les objets existant dans le modèle de l'environnement virtuel. Pour éviter le conflit entre plusieurs écrivains, chaque objet dans l'environnement virtuel possède un processus propriétaire qui seul a le droit de le modifier. Le propriétaire d'un objet peut changer au cours du temps.

Spline n'implémente pas la persistance des objets à travers le temps. Un objet existe tant que l'application qui l'utilise est en cours. Pour avoir des objets persistants, une application doit fournir un processus persistant qui accepte d'être le propriétaire de l'objet en question.

3.2.2.1 Modèle de distribution de SPLINE

Dans Spline le modèle de l'environnement virtuel est répliqué, donc une copie de l'environnement virtuel réside sur chaque processus. Les modifications ou les changements sont communiqués entre les processus par l'envoi de messages.

Spline est conçu de façon à pouvoir supporter un très grand nombre d'utilisateurs (des milliers) qui interagissent en temps réel. Spline repose sur deux idées de base :

- l'égalité approximative des copies locales du modèle de l'environnement virtuel,
- la division du modèle de l'environnement virtuel en morceaux appelés *locales*. Une *locale* sera communiquée seulement aux groupes d'utilisateurs qui sont effectivement intéressés par elle et non pas à tous les utilisateurs de l'environnement virtuel.

Spline n'utilise pas de méthode de synchronisation forte entre les différents fragments de la base de données distribuée. Par contre Spline se soucie de la rapidité

d'interaction en utilisant seulement des égalités approximatives entre les différentes copies du modèle de l'environnement virtuel.

Les différents utilisateurs observent les choses avec une légère désynchronisation. Ce modèle de communication est nommé *le modèle relatif*. Les concepteurs de ce modèle le défendent en disant que c'est ce qui approche le mieux le monde réel. Si on entend une voix dans le monde réel par exemple, cela veut dire que cette voix a été émise il y a déjà un certain temps. Dans les différentes localisations, les gens n'entendent pas cette même voix au même instant. De la même façon, quand un processus dans Spline détecte un changement dans un modèle de l'environnement virtuel, cela veut dire que ce changement s'est produit dans un passé proche. Depuis combien du temps exactement ? Cela dépend de la distance reliant les deux processus qui communiquent.

3.2.2.2 Concept de *locales*

Le concept de *locales* est basé sur l'idée que dans un grand environnement virtuel, un utilisateur n'observe à un moment donné qu'une partie locale de cet environnement. Par la suite, chaque utilisateur a une vision relative à l'environnement. Les *locales* divisent donc un environnement virtuel en des morceaux qui peuvent être manipulés séparéments. Cette division est purement une question d'implémentation et elle est transparente pour l'utilisateur.

Chaque *locale* est associée à un canal séparé de communication *Multicast*. Chaque objet dans l'environnement doit être contenu dans une *locale*. Les limites d'une *locale* spécifient le volume 3D qu'elle occupe. Le volume d'une *locale* est choisi arbitrairement et cela change d'une *locale* à une autre. Par exemple, un immeuble virtuel peut être divisé en plusieures *locales* correspondant aux chambres, corridors, etc. Si un utilisateur peut voir, entendre ou passer d'une *locale* L1 à une autre *locale* L2, alors L2 est dans l'ensemble des voisins de L1.

Les *locales* sont créées dynamiquement par les processus comme tous les autres objets dans Spline. Une fois créée, une *locale* est maintenue et ne peut être modifiée que par son créateur. Les processus peuvent créer des objets et les placer dans n'importe quelle *locale*. Ces mêmes processus seront responsables de la mise à jour des objets qu'ils ont créés.

Des processus spéciaux nommés serveurs des *locales* sont utilisés pour maintenir un enregistrement de l'état de tous les objets dans une *locale* donnée. De cette façon si un processus est intéressé par une *locale* donnée, il peut obtenir rapidement des informations complètes concernant les objets contenus dans cette *locale* sans contacter individuellement tous les processus qui ont des objets dans cette même *locale*. A chaque fois qu'une nouvelle *locale* est créée, elle est assignée à un serveur de *locale*.

3.2.2.3 Communications dans Spline

Comme nous l'avons déjà signalé, Spline ne repose pas sur un concept d'architecture centralisée mais plutôt sur un concept de duplication de l'environnement. Pour minimiser la latence et éviter la surcharge d'un serveur central lors de traitements de données, la communication dans Spline est du type *pair-à-pair* au lieu de passer par un serveur central. Dans la version Spline 1.5 toutes les communications se déroulent en utilisant le mécanisme *pair-à-pair*. Spline 3.0 utilise un modèle de communication hybride où la méthode *point-à-point* est utilisée pour les communications client/serveur, et la communication serveur/serveur est du type *pair-à-pair* en utilisant *Multicast* (Fig. 3.1).

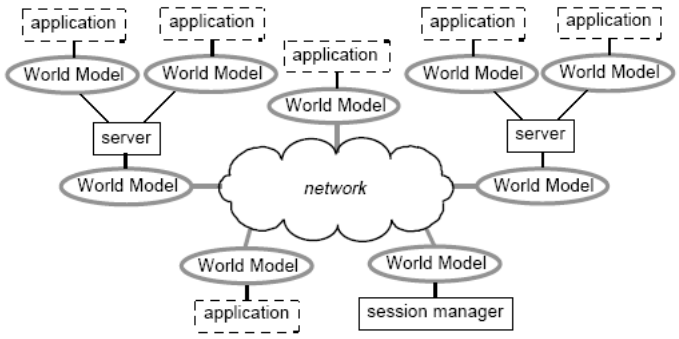

FIG. 3.1 – L'architecture hybride de Spline

Un ou plusieurs serveurs sont inclus dans une session Spline. Chaque serveur s'occupe d'un certain nombre d'utilisateurs. Quand un processus est intéressé par une nouvelle *locale*, il a besoin d'avoir des informations concernant les objets existant dans cette *locale*. Pour permettre à un processus de récupérer ces informations, les serveurs gardent l'état courant de chaque *locale*. Les différentes informations concernant les différentes *locales* sont réparties sur tous les serveurs existants pour ne pas trop charger un serveur particulier.

La copie du modèle de l'environnement virtuel ne contient que des informations concernant les objets contenus dans une certaine *locale*. Il faut donc un mécanisme explicite pour localiser des objets lointains dans l'environnement virtuel. Ceci est réalisé dans Spline à travers les serveurs qui jouent le rôle de serveurs de noms en associant des noms à des objets spéciaux nommés *beacons*. Un processus peut utiliser ce service de nommage pour localiser rapidement un *beacon*. De la même façon que pour les *locales*, la responsabilité du service de nommage est partagée à travers un groupe de serveurs.

La base de Spline est le module de communication inter-processus. Ce module se charge de maintenir une consistance approximative entre les copies de

modèle de l'environnement virtuel. Il s'occupe aussi de l'envoi des messages qui décrivent un certain changement dans l'environnement virtuel, et de la réception des messages en provenance d'autres processus Spline. L'interface réseau de Spline spécifie le format de ces messages.

Trois genres de messages sont envoyés par Spline qui correspondent à trois types de données dans le modèle de l'environnement virtuel : les petits objets à changement rapide, les grands objets qui changent lentement, et les flots de données. Les données les plus répandues au sein d'une simulation Spline sont les mises-à-jour de petits objets qui changent assez fréquemment. Par exemple, un objet qui représente une chose dans l'environnement virtuel (une chaise par exemple) nécessite seulement une description légère pour spécifier sa position et son orientation.

Les messages décrivant les changements dans les petits objets sont envoyés en utilisant le protocole UDP. Cela permet l'envoi rapide d'un message. L'objet doit être assez petit pour que sa description soit adaptée pour l'envoi en un seul message UDP.

Les modèles graphiques, les sons, et les comportements sont représentés en utilisant des objets volumineux. Ces objets sont identifiés par des URLs (*Universal Resource Locators*) et ils sont communiqués en utilisant le protocole WWW standard (*World Wide Web*). Des formats standards sont utilisés pour permettre le développement par des outils standards. Dans Spline 3.0 les formats sont VRML, WAVE, et Java.

Le dernier genre d'objets dans Spline est le flot de données comme les sons capturés par un microphone. Les données de ce genre sont communiquées en utilisant UDP.

3.2.3 Zone d'intérêt dans VELVET

Velvet (*a hybrid adaptive architecture for VEry Large Virtual EnvironmenTs*) est une architecture hybride pour les environnements virtuels partagés [OG02]. Cette architecture permet une adaptation en temps réel aux besoins d'un participant. A n'importe quel instant, un utilisateur peut augmenter ou réduire sa propre vision de l'environnement virtuel. Pour mieux comprendre l'objectif de Velvet, prenons l'exemple suivant démontrant la limitation des systèmes basés sur la notion des *locales* tels que Spline. L'idée de base dans Spline ou plus généralement dans les systèmes adoptant la notion des *locales* est que les participants sont en quelque sorte uniformément répartis dans l'environnement virtuel. Mais que se passe-t-il si tout le monde ou la majorité a décidé d'aller au même endroit ? Dans ce cas il y aura une concentration des participants dans la même *locale* et le nombre des messages communiqués sera trop important. D'où la proposition du concept de zone d'intérêt ajustable dynamiquement de Velvet.

3.2.3.1 Zone d'intérêt

L'idée derrière la zone d'intérêt AOI (*Area of Interest*) est qu'un avatar est capable de voir tout ce qui est localisé dans son AOI. L'AOI d'un avatar est indépendante de celle d'un autre avatar, ce qui permet à chaque site de gérer la dimension de son AOI *i.e.* de ce qu'on peut voir à un instant donné. L'AOI peut être réglée, c'est-à-dire redimensionnée, dynamiquement. Par exemple si à un moment donné la densité des objets qui entourent un avatar a augmenté, cet avatar peut diminuer son AOI de façon à ne s'intéresser qu'à un nombre réduit de ces objets. Seuls les objets qui sont à l'intérieur de l'AOI sont vus par l'avatar. Si le nombre de ces objets diminue, alors l'avatar peut augmenter de nouveau son AOI.

3.2.3.2 Frontière à double couches

Quand un objet traverse la frontière d'un AOI, il doit faire une opération de *check-in* (CI) qui consiste à avertir l'avatar de l'existence d'un nouvel objet dans son AOI. Inversement, une opération de *check-out* (CO) est appliquée lorsqu'un objet dépasse la zone d'AOI vers l'extérieur. Pour éviter de nombreuses opérations de CI/CO, Velvet définit deux frontières nommées AICI (*Area of Interest Check-In*) et AICO (*Area of Interest Check-Out*). De cette façon seuls les objets traversant AICI et qui ne seront pas déjà enregistrés feront l'opération de CI, et seuls ceux traversant AICO et étant enregistrés feront le CO. La distance séparant les deux frontières AICI et AICO est variable. La figure 3.2 montre un exemple d'utilisation de l'AICI et l'AICO. Les cas A, B et C montrent respectivement un AOI avec une seule bordure (distance zéro entre AICI et AICO), et deux distances différentes entre AICI et AICO. Nous pouvons remarquer que dans le cas A 10 opérations de CI/CO ont été effectuées, alors que dans le cas B 6 opérations sont effectuées et que dans le cas C où la distance entre AICI et AICO est la plus importante, on a seulement 3 opérations.

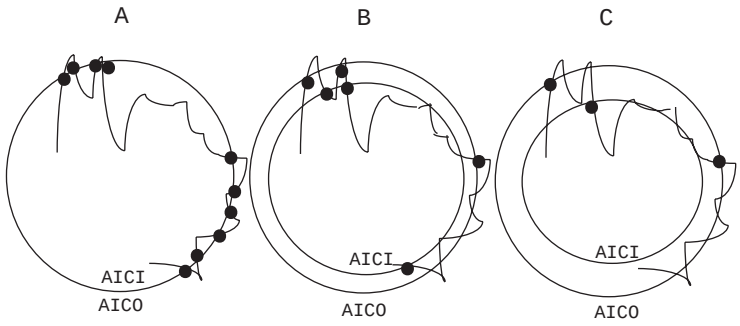

FIG. 3.2 – Frontière à double couches dans VELVET

3.2.3.3 Monde virtuel parallèle

La gestion des AOI dans VELVET n'est pas forcément basée sur des critères spatiaux mais aussi sur des critères fonctionnels qui sont prédéfinis par le gestionnaire d'AOI. De cette manière, chaque avatar possède son propre environnement virtuel parallèle pour ses propres intérêts. Ceci peut engendrer certains cas d'asymétries "autorisées" par VELVET. Par exemple, Un avatar A peut voir un avatar B sans que ce dernier ne le voie. Ceci se produit dans le cas où l'AOI de A est assez étendue pour que B en fasse partie, sans que B n'ait A dans son AOI (Fig. 3.3). Selon la terminologie VELVET ce cas s'appelle *degré de cécité* (*degree of blindness*).

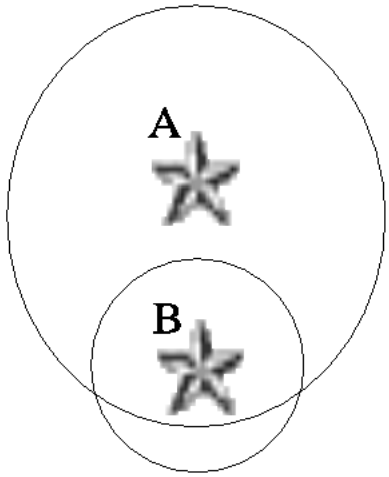

FIG. 3.3 – Degré de cécité dans VELVET

3.2.4 Visibilité des entités dans RING

RING est une plate-forme de développement d'environnements virtuels distribués, conçue et développée par *AT&T Laboratories* [Fun95]. L'idée de base dans la conception de RING est de développer un système de communication qui n'envoie pas systématiquement les messages de mise-à-jour à tous les utilisateurs à chaque fois qu'une entité change d'état. En effet, si on peut réduire le nombre de messages envoyés, on pourra autoriser un nombre d'utilisateurs beaucoup plus important au sein d'un même environnement virtuel.

L'approche dans RING se base sur le fait que les changements d'états doivent être communiqués seulement aux sites contenant des entités qui peuvent probablement percevoir ces changements *i.e.* les entités qui peuvent les voir. L'algo-

rithme de visibilité spatiale d'objet *object-space visibility* est utilisé pour calculer les régions d'influence pour chaque changement d'état. Les messages de mise-à-jour seront envoyés seulement au sous-ensemble des stations pour lesquelles la notification de ce changement est pertinente.

Pour mieux comprendre le fonctionnement de cet algorithme, regardons la figure 3.4. Bien que les entités A, B, C, et D se trouvent dans le même environnement virtuel, très peu d'interactions visuelles sont possibles à cause de la présence des murs ou des bâtiments. Dans cet exemple seule l'entité A peut voir l'entité B, donc un message de mise-à-jour doit être envoyé seulement à l'entité A pour chaque changement de position de B. Les autres entités n'ont pas besoin de distribuer leurs mises-à-jour car ils ne figurent dans le champ visuel d'aucune des entités du monde. A travers cet exemple nous pouvons déduire qu'il est possible de réduire le nombre de messages envoyés pour maintenir la cohérence entre des entités multiples surtout s'il s'agit d'un environnement avec des occlusions.

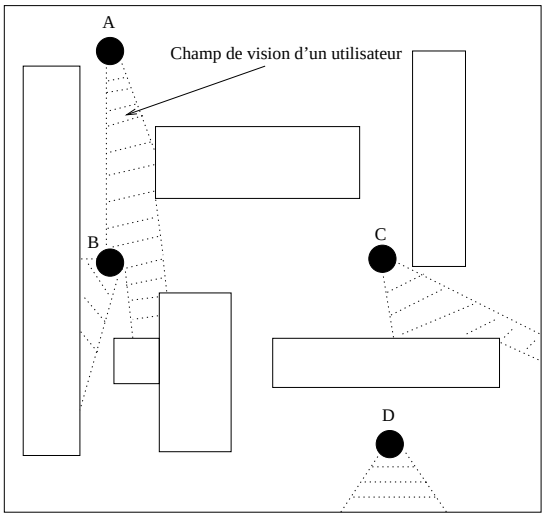

FIG. 3.4 – Visibilité des entités RING

3.2.4.1 L'architecture RING

RING représente l'environnement virtuel comme un ensemble d'entités indépendantes dont chacune a une description géométrique et un comportement. Quelques entités sont statiques (terrains, batiments, ...) et d'autres ont un comportement dynamique (robots, ...).

Chaque entité est gérée par exactement un client (*workstation*). En plus de la gestion des entités locales, chaque client garde la trace de quelques entités qui sont gérées par un autre client (entité distante). Il possède donc des informations concernant leurs représentations géométriques et leurs comportements. Quand un

client reçoit des messages de mise-à-jour concernant une entité gérée par un autre client, il actualise les données de cette entité. En attendant les mise-à-jour, les comportements de ces entités sont décrites localement sur chaque client.

La communication entre les clients est assurée par des serveurs. Un client n'envoie pas de messages directement à d'autres clients mais il les envoie aux serveurs qui à leur tour les acheminent à leur destinations (figure 3.5). Un serveur n'achemine les mises-à-jour qu'aux serveurs et clients gérants des entités qui peuvent probablement voir l'entité qui a changé. L'avantage de cette architecture client/serveur est que les serveurs peuvent traiter les données avant de les envoyer aux destinataires. Par exemple un serveur peut décider qu'un message particulier est utile seulement pour quelques clients précis, il envoie alors ce message seulement à ces clients au lieu de l'envoyer à tout le monde. Par exemple considérons les flots de messages entre les clients A, B, C, et D dans la figure 3.5. Si l'entité A est modifiée, le client A envoie un message au serveur X qui à son tour achemine le message vers le serveur Y et non pas Z car les entités C et D ne sont pas dans le champ visuel de A. Le serveur Y renvoie ce message au client B qui met à jour ses informations locales concernant l'entité A. De même si B bouge il doit envoyer des messages à A et C seulement ; c'est le serveur Y qui s'occupe de l'envoi du message aux serveurs X et Z qui s'occupent de l'envoyer à A et C.

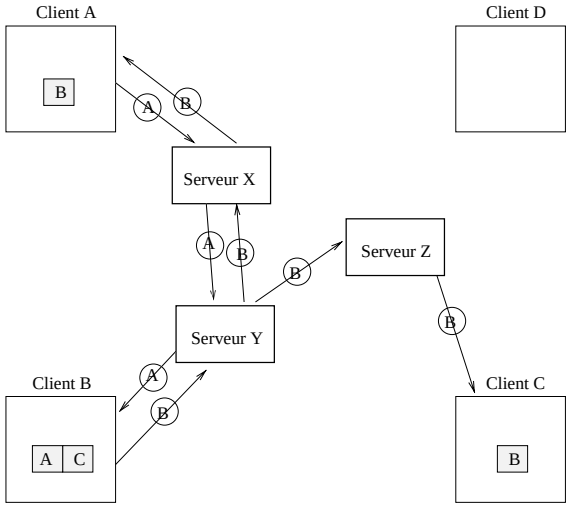

FIG. 3.5 – Les serveurs RING

L'avantage de l'architecture du système RING réside dans le fait que le besoin de performance demandé d'une machine client ne dépend pas du nombre total d'entités de l'environnement virtuel. En effet un client RING doit simuler et s'intéresser seulement aux entités qui sont visibles par lui d'autant plus que le champ visuel d'une entité tend à être limité dans les environnements virtuels contenant des occlusions denses.

Un autre avantage de RING est que la gestion des environnements virtuels est faite par les serveurs sans impliquer chaque client. Cela permet aussi d'utiliser des moyens de communication efficaces entre les serveurs qui ne sont pas forcément valables chez tous les clients. Par exemple les clients peuvent se connecter en utilisant des réseaux à faible bande passante, tandis que les serveurs communiquent entre eux via un réseau à très haut débit.

L'inconvénient de RING est qu'il introduit une latence supplémentaire quand les messages sont basculés via plusieurs serveurs au lieu d'être directement envoyés au destinataire. Ajoutons aussi le temps de calcul passé au sein des serveurs afin de décider où envoyer les messages.

3.2.5 Les hexagones NPSNET

NPSNET (*Naval Postgraduate School Networked vehicle simulator*) est un système de développement d'environnements virtuels distribués multi-utilisateurs. Ce système a été inauguré en 1990 et développé par le groupe de recherche NPSNET à Monterey, Californie, USA.

NPSNET est le premier système de développement d'environnements virtuels qui a été capable d'utiliser l'*Internet Multicast* en utilisant le protocole DIS. Il y a plusieurs versions de NPSNET. Les premières versions NPSNET I, II et III étaient basées sur une technologie Ethernet limitant ainsi le nombre d'utilisateurs. Des améliorations ont été faites pour faire naître une version nommé NPSNETStealth et plus tard NPSNET IV au dessus de DIS 2.0.3. La version la plus récente actuellement est NPSNET V.

NPSNET est capable de simuler des humanoïdes articulés, des vaisseaux marins, des avions et d'autres sortes de véhicules. Il permet aussi d'ajouter des nouveaux modèles d'entités comme des symboles, des terrains, ainsi que des modèles définis par l'utilisateur.

Le principe d'une simulation NPSNET est basé sur le paradigme d'objet et de fantôme. Dans ce paradigme, l'objet original est contrôlé sur sa station de travail, alors que sa représentation sur une autre machine est nommée fantôme. La position du fantôme est mise à jour en utilisant l'algorithme de "*dead-reckoning*".

NPSNET IV comporte quelques améliorations par rapport aux versions précédentes. Ces améliorations sont basées sur deux éléments clés :

- l'utilisation du *Multicast* au lieu du *Broadcast*.
- la gestion des zones d'intérêt.

NPSNET applique un partitionnement hexagonal de l'environnement virtuel. Les hexagones forment des groupes *Multicast* dynamiques. Si un objet ou un avatar se déplace d'un hexagone à un autre, trois hexagones sont ajoutés et trois autres sont supprimés selon le sens du déplacement. À un instant donné, chaque objet est concerné par seulement sept hexagones. Dans la figure 3.6 un objet se déplace

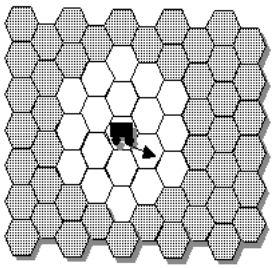

FIG. 3.6 – Partitionnemnet par hexagones dans NPSNET

d'un hexagone à un autre à proximité. Cet objet doit se désabonner des trois groupes Multicast dont il quitte et s'abonner à trois autres groupes Multicast (associés aux trois nouveaux hexagones voisins). En effet, on peut dire que la philosophie de *l'hexagone* est similaire à celle de *locale* dans SPLINE.

NPSNET V offre beaucoup plus de dynamicité que la version IV. Cette dernière version est basée sur Java et elle a la capacité d'ajouter dynamiquement d'autres systèmes ainsi que d'autres fonctionnalités en temps réel durant une simulation et ce d'une manière transparente à l'utilisateur.

3.2.6 Groupes *light-weight* dans DIVE

DIVE fournit une architecture pour implémenter et exploiter un environnement virtuel coopératif. Cette architecture utilise des solutions logicielles pour remédier aux problèmes et aux limitations posés par le réseau. Ces solutions doivent permettre un niveau d'interactivité assez élevé pour pouvoir afficher immédiatement les résultats des interactions d'un utilisateur sur les différents sites participant à une simulation.

DIVE utilise une base de données du type partagée-répartie. Tous les utilisateurs et les applications d'interactions opèrent à travers ce medium commun. Les applications DIVE opèrent d'une façon autonome dans l'environnement virtuel et ne communiquent pas directement entre elles. Cette technique permet de séparer les applications de l'interface réseau. De cette façon la programmation ne diffère pas en écrivant une application mono-utilisateur ou multi-utilisateur à travers le réseau.

Selon DIVE un environnement virtuel est une base de données hiérarchique d'entités. Une entité DIVE est comparable à un objet dans une approche orientée objet, bien que DIVE soit codé en C. En plus des informations graphiques, les entités peuvent contenir des données et des descriptions de comportement définies par l'utilisateur.

3.2.6.1 Base de données répliquée et active

L'architecture DIVE est basée sur une réplication active de la base de données (ou d'une partie de la base de données) pour qu'une copie réside chez le processus de chaque application. Ce modèle permet aux applications d'accéder à la base de données de l'environnement directement en accédant à leur mémoires locales.

Une application qui s'exécute sur un hôte spécifique est appelé *"pair"*. Typiquement, l'ajout des entités, le retrait et les modifications sont faits d'abord sur la copie locale, et ensuite ils sont distribués à tous les pairs par l'intermédiaire d'un message diffusé sur le réseau. Suite à l'utilisation de cette méthode, on dit que la base de données est active.

Conceptuellement, le programmeur pense à une base de données centrale "globale" qui réside quelque part sur le réseau bien qu'en fait elle soit répliquée sur chaque processus.

3.2.6.2 Communication suivant DIVE

La communication suivant DIVE est du type "*Multicast pair-à-pair*". De cette façon les processus ne sont pas obligés de communiquer à un serveur à chaque fois qu'une entité est modifiée contrairement au modèle classique "client-serveur".

Deux types de messages sont généralement envoyés dans DIVE :

- les messages de modifications de la base de données qui sont envoyés en utilisant un protocole *Multicast* fiable. La fiabilité est nécessaire pour assurer que tous les sites ont les mêmes copies de l'environnement.
- le flot de données continu (*data stream*), comme l'audio et la vidéo. Ce type de données est envoyé en utilisant un *Multicast* non-fiable.

Une application (*pair*) qui envoie des informations n'a pas besoin de stocker les messages jusqu'à ce que leurs arrivées soient confirmées chez tous les destinataires comme par exemple TCP. Si une application a détecté un message manquant, elle envoie une requête à l'adresse du groupe *Multicast* associé. En appliquant un algorithme d'estimation (*round trip time*) l'application la plus proche qui a la dernière mise à jour de la version demandée répond à la requête.

3.2.6.3 Les groupes *light-weight*

Pour remédier aux problèmes d'encombrement de réseau et de saturation de trafic, et pour offrir la possibilité à des centaines de participants de partager le même environnement virtuel, DIVE présente un mécanisme qui divise l'environnement en des sous-hiérarchies qui ne sont répliquées et utilisées que par les applications qui en ont besoin. Chaque sous-hiérarchie peut être associée à une voie de communication *Multicast* nommé "*light-weight*" (Fig. 3.7). Les processus

qui sont pas intéressés par ces sous-hiérarchies peuvent ignorer simplement cette branche de l'arbre des entités et faire ainsi diminuer le trafic sur le réseau.

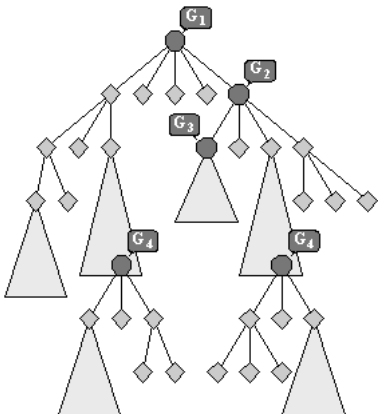

FIG. 3.7 – Les groupes *light-weight* dans DIVE

DIVE utilise la communication Multicast sur deux niveaux différents de la hiérarchie de l'environnement virtuel :

- l'entité qui est au sommet de la hiérarchie est associée à un groupe *Multicast* qui est utilisé par défaut pour la communication avec tout le monde.
- chaque entité dans la hiérarchie peut être associée à un groupe *Multicast* différent. Quand un message de modification concernant une entité est envoyé, l'algorithme remonte la hiérarchie en commencant par l'entité designée. Une fois un groupe *Multicast* trouvé, il sera utilisé comme un médium de communication. Si aucun groupe n'a été trouvé dans l'ascension, DIVE utilise le groupe par défaut qui est associé à la racine de l'arbre.

3.3 Migration et répartition des charges

La complexité croissante des environnements virtuels nécessite d'importantes ressources de calcul. L'idée d'utiliser toutes les ressources de toutes les machines disponibles sur un réseau s'appelle *équilibrage de charges (load balancing)* ou *partage de charges (load sharing)*.

En fait, l'étude des systèmes distribués est un domaine de l'informatique qui a beaucoup de choses à offrir à la réalité virtuelle. La répartition des charges est une des branches d'études des systèmes distribués qui a pour but de répartir le calcul sur tous les processeurs disponibles. Shyamal [Cho90] a défini le partage des charges comme étant une technique pour améliorer la performance d'un système distribué en migrant les tâches qui résident sur les nœuds les plus chargés vers les nœuds les moins chargés.

La notion du temps réel dans certains systèmes distribués rend la réalisation d'un mécanisme de répartition de charges plus compliquée. La raison est que la notion du temps réel exige que le temps alloué à la tâche de migration soit très limité. Dans une telle situation, si l'algorithme de distribution des charges n'est pas efficace, le résultat de la distribution n'est pas optimal.

La migration de processus peut aussi être très coûteuse en terme de communication. Ce coût croît dramatiquement si le processus à faire migrer était en état d'exécution lors du déclenchement de l'opération de migration. Ceci est dû aux informations additionnelles qu'il faut faire migrer avec le processus pour pouvoir le remettre en fonction dans le même état que lors de sa suspension. Donc en plus du coût élevé en terme de temps (les processus sont suspendus pendant la migration), la migration peut aussi entraîner un encombrement au niveau de la couche de communication. Ces limitations peuvent être acceptables pour certains systèmes distribués, mais pour les systèmes de développement d'environnements virtuels, des telles circonstances rendent le temps réel difficile à atteindre.

Dans les plates-formes de développement d'environnements virtuels distribués, la migration présente aussi d'autres intérêts comme l'ajout et le retrait dynamique de site pendant une simulation. Par exemple, l'ajout dynamique d'un site nécessite un mécanisme qui est capable de copier ou répliquer les objets de l'environnement virtuel sur la nouvelle machine. Le mécanisme de migration d'objets permet facilement d'atteindre cet objectif. De la même manière, le retrait dynamique d'un site nécessite un mécanisme de migration pour garder les objets simulés qui étaient initialement sur le site qui demande la déconnexion. La migration peut assurer alors le transport des objets vers d'autres sites avant la déconnexion du site en question.

3.3.1 Pourquoi la migration ?

Le but de la migration au sein d'un système distribué quelconque est étroitement lié au type d'application qui va s'en servir. En général on peut citer les buts suivants :

- **plus de capacité de calcul :** ce but est normalement atteint lorsque la migration est utilisée pour assurer la répartition de charges. Par exemple, dans le système d'exploitation distribué MOSIX [BLB95], quand un nœud est moins chargé il annonce sa disponibilité et il initialise la procédure de migration de processsus à partir du nœud le plus chargé.
- **exploitation locale des ressources :** la migration permet d'atteindre cet objectif dans le cas où il est plus pratique d'accéder aux ressources localement que d'y accéder à distance. Par exemple le transfert d'un processus d'une machine à une autre transforme l'accès distant en un accès local, ce qui peut entraîner une amélioration des performances dans certains cas.

- **partage de ressources :** la migration permet l'accès à des nœuds équipés d'un certain type spécifique de dispositifs ou d'une mémoire importante.
- **tolérance aux fautes :** la tolérance aux fautes peut être améliorée en migrant les objets et les processus d'un nœud partiellement défaillant vers un autre plus sûr. L'échec d'un nœud peut être dû à des raisons internes comme une défaillance quelconque ou des raisons externes comme une défaillance au niveau du réseau qui l'interconnecte à d'autres nœuds.

3.3.2 Migration de processus dans les systèmes distribués

Deux facteurs peuvent compliquer la réalisation d'un mécanisme de migration. Le premier est lié au système qui veut intégrer la migration. En effet, un système distribué qui est conçu d'une manière complexe ne facilitera pas l'intégration d'un mécanisme de migration. Le deuxième facteur est lié au niveau duquel le mécanisme de migration va opérer et à son degré de transparence.

La migration peut être classifiée selon le niveau auquel elle est appliquée. Elle peut être appliquée au niveau noyau d'un système, au niveau couche de service ou au niveau applicatif comme faisant partie de l'application elle même. L'implémentation de la migration sur des niveaux différents entraîne des performances différentes. De plus, on n'obtient pas le même niveau de transparence ni de réutilisabilité.

3.3.2.1 Migration au niveau noyau

Il y a eu beaucoup de réalisations de systèmes d'exploitation distribués comportant un mécanisme de migration de processus au niveau noyau. La plupart de ces implémentations a exigé un effort significatif et des modifications importantes sur le noyau [BS85, TLC85, VMB+02]. Cette complexité est dûe à l'architecture des systèmes d'exploitation distribués et leur incapacité de supporter les interactions complexes provoquées par la migration de processus. Notons aussi que la plupart de ces systèmes n'étaient pas prévus initialement pour supporter un tel mécanisme.

Comme exemple de système distribué supportant la migration de processus au niveau noyau on peut citer MOSIX et Kerrighed. Le système d'exploitation MOSIX supporte la migration de processus au sein d'un système à image unique[1] [BL85] ainsi que dans un environnement de type NOW (*Network Of Workstations*) [BLB95]. Le mécanisme de migration de processus est utilisé pour assurer l'équilibrage dynamique de charges. Kerrighed est un système d'exploitation distribué à image unique réalisé à l'IRISA [VLR+03]. Kerrighed vise à donner une vision d'une machine de type *SMP* (*Symetric MultiProcessing*).

[1] c'est-à-dire qu'il offre la vision d'une machine unique

3.3.2.2 Migration au niveau couche de service

La réalisation d'un mécanisme de migration au niveau couche de service est plus facile qu'une implémentation au niveau noyau système. L'inconvénient est la performance réduite et l'absence de transparence par rapport à une migration au niveau noyau.

On peut citer à titre d'exemple Condor [Lit87, LS92] qui est une couche logicielle supportant la migration des processus. La procédure de migration d'un processus consiste à générer un fichier de sauvegarde d'espace d'adressage mémoire associé à ce processus, le joindre à l'exécutable du processus et l'envoyer vers la machine cible. Les appels systèmes sont redirigés vers un processus "fantôme" sur la machine source. Condor ne supporte pas la migration des processus qui utilisent des signaux, des fichiers mappés en mémoire, des librairies partagées ou IPC. MPVM (*Migratory Parallel Virtual Machine*) [CCC+95] est un autre exemple de couches logicielles qui offrent une migration au niveau utilisateur. MPVM est une extension du PVM [BDG+93] qui supporte la migration de processus entre des machines homogènes. Les objectifs de MPVM sont la transparence, la compatibilité avec PVM et la portabilité.

3.3.2.3 Migration au niveau applicatif

La migration peut aussi être implémentée comme une partie d'une application. Une telle approche sacrifie la transparence et la réutilisabilité [Fre91, Sko95]. De plus, la migration doit être ajustée pour chaque nouvelle application. Néanmoins l'implémentation peut être simple à réaliser et optimisée spécialement pour l'application en question.

3.3.3 Migration d'objets dans les systèmes de développement d'environnements virtuels

Les Systèmes de développement d'environnements virtuels qui intègrent un mécanisme de migration sont rares. Aucun des systèmes étudiés dans 3.2 ne fait mention d'un tel service. Avocado [Tra99] est un autre système qui mentionne la possibilité de faire migrer ses objets mais sans fournir plus de détail sur l'implémentation de son système de migration. Huang et al. ont présenté dans [HDW03] une étude de migration d'avatars d'un serveur à un autre dans un environnement virtuel distribué, mais cette étude n'est pas généralisée pour tous les objets dans l'environnement : seuls les avatars peuvent migrer d'une machine à une autre. Parmi les systèmes qui offrent la migration d'objets nous allons détailler AVIARY et WAVES.

3.3.3.1 Migration d'objets dans AVIARY

AVIARY [SWH93, SW94, WHH+93] est un système réalisé à l'université de *Manchester*. Ce système autorise la cohabitation entre plusieurs environnements virtuels en même temps ce qui permet à plusieurs utilisateurs d'agir au sein d'un même environnement virtuel. L'idée d'avoir plusieurs environnements virtuels parallèles ressemble un peu à l'environnement de fenêtrage du bureau (*Desktop Windowing*). Dans cet environnement, un utilisateur peut avoir plusieurs programmes s'exécutant en parallèle dans des fenêtres séparées. Il suffit de déplacer le pointeur de la souris sur les différentes fenêtres pour les activer. De la même façon, les environnements virtuels peuvent cohabiter dans AVIARY. Le passage d'un environnement à un autre se fait à travers des portails. L'utilisateur accède simplement à ces portails qui existent dans l'environnement de la même façon qu'on traverse les portes dans un batiment.

AVIARY est conçu pour permettre l'utilisation de matériel parallèle ainsi que des systèmes distribués. Le noyau d'AVIARY est le système de communication qui permet à tous les objets de communiquer et qui permet aussi à un objet de migrer d'un processus à un autre. La création et destruction des objets ainsi que leur procédure de migration sont réalisés par le système de communication. Pour éviter le goulot d'étranglement dans le système de communication, son implémentation est complètement distribuée, par contre aucun détail technique n'a été présenté concernant comment la réalisation a été faite.

Le gestionnaire d'environnement virtuel VEM (*Virtual Environment Manager*) est un composant d'AVIARY qui offre des services systèmes comme l'association d'identificateur unique aux objets. Les services de VEM n'exigent pas de calculs ni de bande passante importante.

La base de données de l'environnement EDB (*Environement Data Base*) fournit une gestion spatiale d'objet, en particulier la détection de collision. EDB connaît la localisation de tous les objets dans l'environnement et il autorise les interactions avec ces objets.

Enfin, il faut rappeller qu'AVIARY est l'un des rares systèmes de développement d'environnements virtuels qui intègrent un sous-système de migration d'objets. Les concepteurs d'AVIARY ne mentionnent cependant pas comment la réalisation a été faite ni si un algorithme de répartition des charges est utilisé dans leur système.

3.3.3.2 Équilibrage de charges dans WAVES

WAVES (*WAterloo Virtual Environment System*) est un système de développement d'environnements virtuels présenté dans [Gau95, Kaz96]. WAVES est le successeur du système HYDRA présenté par Kazman dans [Kaz95]. Selon l'architecture WAVES, un environnement virtuel contient un certain nombre de gestion-

naires de messages (*Message Managers*). Les gestionnaires de messages jouent le rôle de médiateurs de communication entre les différents hôtes. Les hôtes sont des processus qui simulent un certain nombre d'objets virtuels nommés *objoïdes*. L'environnement virtuel simulé suivant WAVES est composé strictement d'objoïdes. Une description de l'architecture typique de WAVES est présentée dans la figure 3.8.

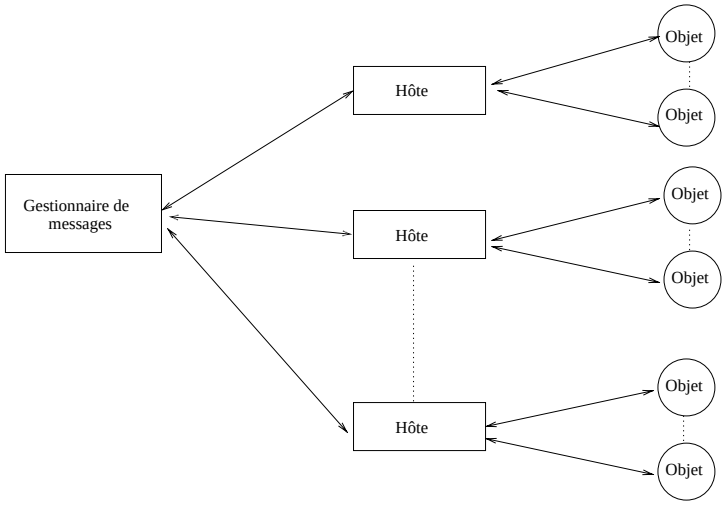

FIG. 3.8 – L'architecture WAVES

Dans un environnement virtuel distribué, chaque hôte simule un sous ensemble d'objets. Quelques un de ces objets sont natifs sur un hôte donné, d'autres sont des clones. La différence entre les objets natifs et les clones est typiquement le degré de fidélité de leur simulation. Sur un hôte les objets natifs sont simulés avec un degré de fidélité très élevé, tandis que les clones sont simulés avec une fidélité moins élevée. On peut comparer ce comportement à celui du modèle référentiel/miroir utilisé dans d'autres systèmes.

Gestionnaire de messages Le gestionnaire de messages assure la communication entre les hôtes et c'est à sa charge de savoir qui communique avec qui et quand. Le gestionnaire de messages permet un environnement virtuel flexible où les hôtes peuvent joindre et quitter l'environnement d'une façon dynamique. En plus d'établir la connexion entre les différents hôtes, le gestionnaire de messages peut filtrer les messages reçus par un hôte particulier. Ce filtrage est une sorte de gestion de zones d'intérêts qui a pour but de minimiser le taux de communication dans un environnement virtuel complexe.

En effet, un hôte WAVES peut à tout moment (et spécialement lors de son initialisation) envoyer un ensemble de spécifications de filtrage au gestionnaire de messages. Ces filtres peuvent être sémantiques ou spatiaux. Dans le cas où les

messages à transmettre entre les hôtes sont volumineux (une vidéo par exemple) et la minimalisation de la latence jugé cruciale, le gestionnaire de messages crée un canal de communication du type point-à-point entre les hôtes.

Notons que le gestionnaire de messages peut devenir un ensemble de gestionnaires distribués et interconnectés dans le cas d'un environnement virtuel comportant un nombre important de sites.

Migration d'objoïdes La fonctionnalité qui nous intéresse le plus dans WAVES est la migration. Un objet WAVES entame son existence sur un hôte particulier, mais il peut migrer vers d'autres hôtes pour des raisons d'efficacité. Un objet a aussi le droit de demander la résidence permanente sur son hôte parent initial dans le cas où cet hôte est localisé sur une machine qui a un équipement spécial et primordial pour l'objet en question.

La migration des objets WAVES entre les différents hôtes ne fait pas partie des responsabilités des hôtes eux mêmes. Un hôte n'est même pas conscient de l'existence des autres. La tâche de la migration est assurée par le gestionnaire de messages.

La procédure de migration commence par l'envoi d'un message à l'hôte source. Ce message lui indique qu'une migration est en cours et quel est l'objoïde affecté. Après l'envoi du message de migration, le gestionnaire de messages doit attendre une confirmation de la part de l'hôte source. En attendant, le gestionnaire de message traite tous les messages en provenance de l'hôte source. Si à ce stade l'hôte source disparaît, la migration est annulée. Autrement, la confirmation est reçue, et tous les messages en provenance de l'hôte source sont expédiés directement vers l'hôte destination, jusqu'à la réception d'une autre confirmation de la source qui indique que tous les attributs d'objoïdes ont été envoyés. Si l'hôte destination disparaît à ce stade de l'évolution, la migration est annulée. Après que tous les attributs sont envoyés à l'hôte destination, une confirmation de transfert est envoyée vers l'hôte source. Finalement, un message est envoyé vers l'hôte destination lui indiquant qu'il est devenu le propriétaire de l'objoïde.

Équilibrage de charges WAVES implémente un algorithme d'équilibrage qui a pour but de répartir la charge sur tous les hôtes de manière optimale. La charge d'un hôte est définie par la somme des distances de tous les objoïdes j sur cet hôte du centre de l'ensemble d'objoïdes divisé par la capacité de calcul de l'hôte.

La capacité de calcul de l'hôte est une constante K prédéfinie. Le centre de l'ensemble d'objoïdes est le "centre de gravité" des objoïdes simulés actuellement sur l'hôte. Ce centre est défini comme étant la moyenne des centres géométriques d'objoïdes divisée par la taille de chaque objoïde. Cela veut dire qu'un objoïde volumineux va attirer le centre de gravité vers lui.

Donc la charge d'un hôte peut être donné par la formule suivante :

$$\frac{\sum_{i=1}^{i=j} dist(objode_i, centre)}{K}$$

De cette manière, la charge d'un hôte est définie selon trois considérations : la capacité de calcul de l'hôte, le nombre d'objoïdes sur l'hôte et la distance qui sépare les objoïdes. Si un hôte possède un nombre important d'objoïdes étroitement liés, il peut les garder jusqu'à ce que sa charge atteigne un certain seuil.

3.4 Synthèse

Dans ce chapitre nous avons présenté deux services complexes offerts par quelques plates-formes logicielles de développement d'environnements virtuels distribués. Ces services sont la gestion des zones d'intérêt et la migration.

La gestion des zones d'intérêt sert à minimiser le taux de données traitées et échangées sur le réseau. Chaque site utilisateur ne traite que les informations qui lui sont utiles selon ses critères. La minimisation des calculs ainsi que du taux de transfert de données sur le réseau améliore la performance du support d'exécution des environnements virtuels distribués. Ce service est assez largement répandu dans les plates-formes de développement d'environnements virtuels distribués.

La migration est un sujet peu traité dans les plates-formes de développement d'environnements virtuels distribués. La raison est peut être la dégradation de performance que la migration peut provoquer au sein d'un système interactif. Cette dégradation est expliquée par le temps nécessaire pour faire migrer un objet d'un site à un autre ce qui entraîne la suspension momentané de l'objet en question. Par contre la migration peut être fort utile pour certaines catégories de systèmes comme par exemple les jeux coopératifs dynamiques. Dans ces systèmes un utilisateur peut se connecter ou se déconnecter à n'importe quel moment. Dans ce cas, la migration assure le transfert des données ou des objets d'un site vers un autre au gré des besoins.

Chapitre 4

Systèmes coopératifs

4.1 Introduction

Une simulation peut être vécue uniquement par un individu (systèmes mono-utilisateur), ou au contraire par un groupe (systèmes multi-utilisateurs). Les environnements virtuels multi-utilisateurs coopératifs permettent aux utilisateurs d'accomplir une tâche commune d'une manière coopérative. Toute action effectuée par l'un des utilisateurs peut modifier l'environnement virtuel et doit pouvoir être perceptible par les autres utilisateurs. Pour effectuer une coopération au sein d'un environnement virtuel distribué, les utilisateurs doivent avoir un minimum de connaissances. Ces connaissances sont des informations telles que : qui existe dans l'environnement ? Qui fait quoi ? Qui coopère avec qui ? Comment ? Quand ?

Pour répondre à ces questions, les travaux relatifs aux environnements virtuels coopératifs font appel à plusieurs disciplines : la psychologie, le Travail Coopératif Assisté par Ordinateur ou TCAO (*Computer-Supported Cooperative Work*), l'Interaction Homme Machine ou IHM (*Human Computer Interaction*), etc. Le facteur psychologique s'intéresse aux processus mentaux d'un individu, c'est-à-dire la façon dont la représentation de l'information est traitée par le cerveau. Les recherches sur le travail coopératif à l'aide de l'ordinateur tentent de déterminer comment l'informatique peut aider des groupes de gens à travailler ensemble sur un projet, pour élaborer un produit, pour prendre une décision, etc. [Gru94]. L'interaction homme-machine est une discipline qui quant à elle s'intéresse à la conception, l'évaluation et l'exécution des systèmes de calcul interactifs pour l'usage humain et à l'étude des phénomènes principaux qui les entourent.

Il y a beaucoup de manières pour plusieurs utilisateurs d'accomplir un travail d'une façon coopérative dans un environnement virtuel. Quelques exemples sont cités dans [LG93] comme la manipulation d'objets à distance et la modélisation coopérative.

Dans un environnement virtuel coopératif les utilisateurs doivent avoir la sen-

sation de présence des autres utilisateurs. Ensuite, ils doivent pouvoir interagir sur les objets (ou sur un certains types d'objets) existants dans l'environnement virtuel. Pour assurer la coopération entre les utilisateurs il faut assurer un niveau de compréhension mutuel ainsi qu'une bonne communication entre eux.

4.2 Conscience et Métaphore

Avant d'approfondir ce chapitre, nous allons définir les deux termes suivants : conscience et métaphore. Ces deux termes sont largement utilisés tout au long de ce chapitre et ont une signification particulière dans notre contexte.

4.2.1 Conscience

La conscience est par définition la connaissance intuitive, immédiate, que l'être humain possède de son existence, de ses facultés, de ses actes. Le concept de *conscience* est peut-être l'aspect le plus fondamental de l'interaction dans les environnements virtuels. Sans une certaine indication minimale des états d'autres collaborateurs dans l'environnement virtuel, il est extrêmement difficile d'établir une base pour n'importe quel genre d'interaction. La conscience est la perception directe, plus ou moins claire et précise, du monde qui nous entoure. C'est aussi la connaissance d'une situation ou d'un problème.

Gutwin et Greenberg [GG99] relèvent quatre caractéristiques inhérentes au concept de conscience :

1. la conscience est la connaissance de l'état d'un environnement délimité dans l'espace et le temps.
2. la conscience est constituée de connaissances qui doivent être mises à jour en fonction des transformations de l'environnement.
3. la conscience est entretenue par l'interaction des personnes avec l'environnement.
4. la conscience est presque toujours liée à une activité. La conscience n'est pas un but en soi mais elle participe à la résolution d'une tâche.

Greenberg et Al. [GGC96] remarquent aussi que la conscience est plus difficile à maintenir dans les environnements virtuels que dans les espaces de travail réels. Ils donnent trois raisons :

1. les environnements virtuels n'émulent pas la richesse des espaces de travail réels. L'information résultant de l'interaction des sujets dans un espace de travail réel est beaucoup plus riche que celle obtenue dans un environnement virtuel.

2. les interactions d'un utilisateur dans un environnement virtuel génèrent beaucoup moins d'informations que les actions d'un collaborateur dans l'espace de travail réel. Pour illustrer cette idée, ils citent l'exemple suivant : un collègue de bureau qui prend un document, le déchire et le jette à la poubelle crée relativement plus d'informations qu'un utilisateur qui sélectionne un fichier et appuie sur la touche "supprimer".
3. les auteurs adressent une critique discrète aux concepteurs d'environnements virtuels en soulignant la fréquente sous-exploitation du peu d'informations à disposition du système relatif à la conscience.

Dans un environnement virtuel la conscience peut être offerte par de nombreux moyens et par des médias divers plus ou moins réalistes. En effet, dans les environnements virtuels partagés, on essaye toujours de fournir des formes les plus "normales" possibles de conscience. La conscience peut être mutuelle ou à sens unique. Le désir d'un participant d'être perçu peut être augmenté ou limité, ce qui fait augmenter ou diminuer la probabilité que les autres le perçoivent dans l'environnement virtuel. Le modèle qui présente exactement ces concepts dans un environnement virtuel est le modèle spatial d'interaction [BBFG94, GB97].

Les outils permettant de maintenir la conscience au sein d'un environnement virtuel sont schématisés par des métaphores.

4.2.2 Métaphore

La métaphore (du grec *metaphora* qui signifie "transport" est selon Aristote "le transport à une chose d'un nom qui en désigne une autre" (Poétique). Cette figure rhétorique consiste à substituer à un mot un autre mot sous l'effet d'une comparaison qui reste implicite.

En réalité virtuelle, la métaphore est une représentation réaliste ou parfois symbolique d'une action donnée ou d'un état. Prenons par exemple les *emoticons* utilisés très souvent dans les environnements de discussions sur *Internet*. Un visage souriant est une métaphore qui représente l'état d'un utilisateur qui est content.

4.3 Perception de la présence des utilisateurs en environnements virtuels

La "présence" peut être le composant le plus élémentaire de la collaboration virtuelle. La présence est le sentiment d'"être là". Dans le contexte d'environnements virtuels partagés, la présence peut être vue comme une sorte d'incorporation de l'environnement virtuel. Ce genre de présence fournit le sens que d'autres

collaborateurs, et même d'autres objets, partagent l'environnement avec nous. Il faut être conscient de la présence des autres pour pouvoir collaborer avec eux.

4.3.1 Conscience de présence

Il n'est pas difficile de fournir des indications basiques de la présence des autres. Une simple liste du nom de chaque participant est suffisante pour indiquer que plusieurs utilisateurs partagent le même environnement. La description textuelle des participants est utilisée par exemple dans les MUDs ou les serveurs de discussion où les utilisateurs sont reconnus par leur nom ou par leur pseudo. Ces méthodes sont très peu coûteuses et satisfont au besoin de base, mais par contre elles sont très peu réalistes lorsqu'on les applique au sein d'un environnement virtuel partagé.

La télé-immersion et certains environnements virtuels placent l'utilisateur à l'"intérieur" de l'environnement. En conséquence, d'autres collaborateurs dans l'environnement peuvent être représentés par des incorporations sophistiquées qui sont propres à l'immersion. Les avatars, représentations graphiques des collaborateurs à distance, sont des exemples de telles incorporations qui établissent un sens fort de la présence. Les avatars peuvent être aussi simples que des images GIF des autres participants, ou ils peuvent être aussi détaillés que des images visuelles tridimensionnelles d'une personne réelle.

4.3.2 Métaphore de présence

Avatar est en effet un mot qui vient du Sanskrit *avatara* qui veut dire "la descente du ciel sur terre" ou "la descente du Dieu sur terre". Dans l'hindouisme, les *avataras* sont les incarnations du dieu Vishnu :

The Avatara, or incarnation of Godhead, descends from the kingdom of God for [creating and maintaining the] material manifestation. And the particular form of the Personality of Godhead who so descends is called an incarnation, or Avatara. Such incarnations are situated in the spiritual world, the kingdom of God. When They descend to the material creation, They assume the name Avatara. (Chaitanya-caritamrita 2.20.263 -264)

Dans le contexte des environnements virtuels, le mot avatar n'a rien de divin, mais sa définition s'étend à n'importe laquelle des représentations que peut avoir un utilisateur. Un avatar est une métaphore qui peut être employée pour simuler différentes notions de la présence. Par exemple, dans un environnement virtuel, nous pouvons simplement regarder un autre avatar et immédiatement déterminer beaucoup de choses qui, autrement, pourraient exiger des indicateurs complexes et artificiels dans un environnement de travail de groupe plus traditionnel. Si un utilisateur distant est éloigné de nous dans l'environnement virtuel, alors son

avatar, obéissant aux lois normales de l'espace, semble petit et difficile à voir clairement. Si l'utilisateur s'approche, ou si nous nous rapprochons de lui, des détails d'aspect de son avatar deviennent plus clairs. Quand l'avatar se tient directement devant nous, nous voyons une incorporation grandeur nature de l'utilisateur en question, presque comme si la personne était là avec nous. Cela est aussi vrai pour les objets dans l'environnement virtuel. Se tenir à la base d'un bâtiment virtuel n'a pas du tout le même effet que lire une description de la même scène, ou alors regarder une image de ce bâtiment.

Le bruit est également un composant important pour la présence. Dans une interaction en tête à tête entre deux personnes, nous communiquons en parlant. Ce que nous entendons est fortement couplé à ce que nous voyons ; les mains bougent, les lèvres se déplaçent, les yeux jettent un coup d'oeil autour de la salle. Toutes ces choses contribuent ensemble à un plus grand sentiment de l'espace partagé. Des travaux ont été faits pour combiner le bruit avec l'espace volumétrique et virtuel de sorte que les utilisateurs perçoivent correctement des bruits basés sur la distance, la direction, et l'amplitude.

En utilisant des capteurs de mouvements, des avatars peuvent faire des gestes simples comme fixer le regard sur quelqu'un ou faire bouger les mains [BBFG94, LDJ+97]. Dans une des premières implémentations de MASSIVE, Benford et al. [BBFG94] ont montré comment des changements visibles de l'apparence d'un avatar peuvent fournir des indications à l'utilisateur distant. Quand un utilisateur peut entendre ce qui était dit, les oreilles de son avatar se développent, et quand l'utilisateur peut communiquer verbalement, son avatar développe une bouche.

4.4 Interaction avec les objets

Le terme "interaction" peut signifier n'importe quel raccordement ou communication entre les objets. L'interaction peut être aussi simple que la seule vue d'un autre objet, ou elle peut être aussi complexe qu'une conversation en tête à tête ou une manipulation d'objets. La façon dont une interface de manipulation d'objets est conçue influence la performance d'un utilisateur d'environnement virtuel immersif [Min95]. La sélection d'objet et son positionnement sont parmi les interactions les plus classiques entre un utilisateur et un objet virtuel.

Si l'on se refère à la décomposition des principales interactions proposée par Chris Hand [Han97] qui sont : interaction avec un objet, navigation et contrôle d'application. Nous sommes clairement ici dans la partie interactions avec les objets. De ce point de vue, Chaque application spécifique possède un certain nombre de tâches à accomplir. Une tâche est une coordination logique de plusieurs actions. Il y a plusieurs genres d'actions élémentaires. Dans [Wut99] Wuthrich a introduit une classification de tâches basée sur le nombre d'actions atomiques à réaliser.

Ce qui nous intéresse dans cette approche est la liste des actions atomiques qui ont été identifiées :

- sélectionner : l'action de sélectionner un objet ou un ensemble d'objets sert à montrer une volonté d'interaction avec ces objets.
- positionner : cette action sert à déplacer les objets d'une position à une autre.
- déformer : cette dernière action permet de modifier la forme et la taille d'un objet.

Dans ce qui suit, nous allons citer quelques périphériques physiques d'interactions rencontrés dans les environnements virtuels, puis nous allons détailler quelques métaphores d'interactions.

4.4.1 Périphériques physiques d'interactions

Dans ce paragraphe nous allons citer quelques interfaces d'interactions et de manipulations d'objets au sein d'un environnement virtuel. Parmi les interfaces, certaines sont classiques comme la souris, le clavier ou le joystick. D'autres interfaces le sont moins et peuvent être dédiées aux interactions au sein d'un environnement virtuel. Faire une liste exhaustive de toutes les interfaces qui existent serait une tâche trop longue et fastidieuse. Nous n'en citerons que quelques unes à titre d'exemple.

- souris 3D : c'est un dispositif de désignation à six degrés de liberté qui sert à selectionner et contrôler des objets dans un environnement virtuel.
- Phantom : c'est un dispositif d'entrée haptique à six degré de liberté qui fournit la sensation de retour d'efforts (à trois degrés de libertés seulement). Phantom est développé par *SensAble Technologies Inc* [1] (Fig. 4.1). Le phantom est utilisé dans plusieurs applications 3D du type CFAO et dans des applications médicales. Les actions atomiques les mieux supportées par le phantom sont la sélection et la positionnement.
- Le pointeur laser *Laser Pointer Pro* [2] : il remplace le curseur de la souris comme dispositif d'interactions. Un faisceau laser est lancé du pointeur de l'utilisateur en ligne droite. Le premier objet intersecté par le faisceau est sélectionné pour une probable manipulation. Ceci a l'avantage de permettre des interactions à distance.
- The Bat : ce dispositif est une sorte de souris à six degrés de liberté présenté par Ware dans [WJ88, War90]. Les actions permises par le Bat sont la sélection et le positionnement.
- Interface de déformation : Murakami et al. ont présenté dans [MN94] une interface en forme de cube qui sert pour la déformation d'objets en envi-

[1] SensAble Technologies, Inc. : http://www.sensable.com
[2] http://accettura.com/products/laserpointerpro

FIG. 4.1 – Phantom

ronnement virtuel (Fig 4.2). Un utilisateur peut déformer un objet 3D en interagissant sur le cube d'une manière naturelle (presser, plier, etc.). Les concepteurs de ce cube disent qu'il peut être utile aussi pour la conception d'objets virtuels basiques.

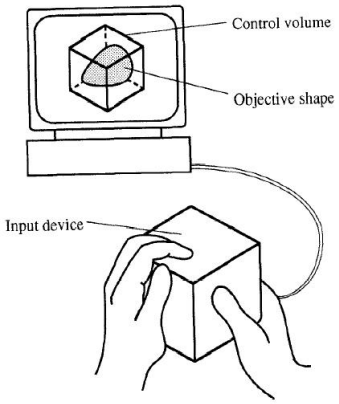

FIG. 4.2 – Cube de déformation

4.4.2 Conscience d'interaction

Les techniques de manipulation d'objets dans les environnements virtuels doivent rendre l'utilisateur totalement conscient de ses interactions. Les utilisateurs doivent savoir exactement ce qu'ils peuvent faire et comment ils peuvent faire. Ils doivent aussi être conscients de ce qu'ils ne peuvent pas faire.

Poupyrev et al. ont classifié les techniques d'interactions en deux catégories : les techniques exocentriques (objectives) et les techniques égocentriques (subjec-

tives) [PWBI98]. Selon l'interaction exocentrique (nommée aussi point de vue de l'œil du Dieu) l'utilisateur interagit avec l'environnement virtuel "de l'extérieur". Prenons par exemple la technique nommé monde-en-miniature *World-In-Miniature* [SCP95]. Selon cette technique on manipule les objets en interagissant sur leurs modèles de représentations en miniature.

Les techniques les plus communes d'interactions dans les environnements virtuels sont celles dites égocentriques. L'interaction est dite égocentrique lorsque l'utilisateur agit de l'intérieur de l'environnement. Dans le paragraphe suivant, nous discutons les métaphores utilisées par cette catégorie de techniques d'interactions.

4.4.3 Métaphores d'interactions

Deux métaphores basiques d'interactions égocentriques comptent parmi les plus utilisées : la main virtuelle et le pointeur virtuel 3D [Min95, BH97, PFC+97].

4.4.3.1 Main virtuelle

La métaphore de la main virtuelle est une représentation symbolique de la main humaine. Un utilisateur peut utiliser la main virtuelle pour sélectionner "toucher" et positionner les objets virtuels. On peut trouver deux variantes de la métaphore de la main virtuelle.

- La main virtuelle "classique" : la position et l'orientation de l'utilisateur de la main virtuelle sont contrôlées par un capteur attaché à sa main réelle. Pour sélectionner un objet avec la main virtuelle, l'utilisateur doit intersecter l'objet avec la main virtuelle et appuyer sur un bouton de sélection ou effectuer un mouvement particulier de la main. La figure 4.3 montre un exemple de la métaphore d'une main virtuelle.
- La technique *Go-Go* : cette technique utilise la même métaphore d'une main virtuelle sauf que cette main est supportée par un bras qui peut croître dynamiquement pour toucher des objets lointains [PBZI96].

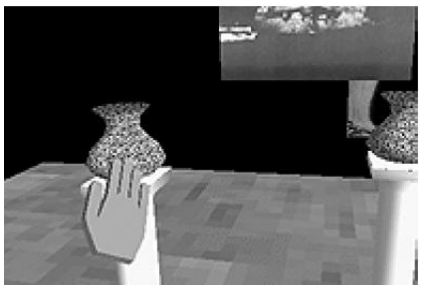

FIG. 4.3 – Métaphore de la main virtuelle

4.4.3.2 Pointeur virtuel 3D

La métaphore d'un pointeur virtuel permet à un utilisateur de sélectionner et positionner un objet en pointant sur lui. Quand le vecteur émanant du pointeur virtuel intersecte l'objet virtuel, ce dernier est sélectionné et prêt pour la manipulation [Min95]. Différentes variantes de cette technique existent aussi. Ces variantes se différencient par la définition de la direction du pointeur, sa forme, et les méthodes utilisées pour rendre la sélection d'un objet la moins ambigüe possible (surtout si les objets sont très proches les uns des autres). Dans les cas les plus simple, la direction du pointeur virtuel est définie par l'orientation de la main virtuelle et le pointeur est un rayon laser. On peut aussi trouver des cas plus complexes comme par exemple le pointeur flexible proposé par Olwal et Feiner dans [OF03] (4.4).

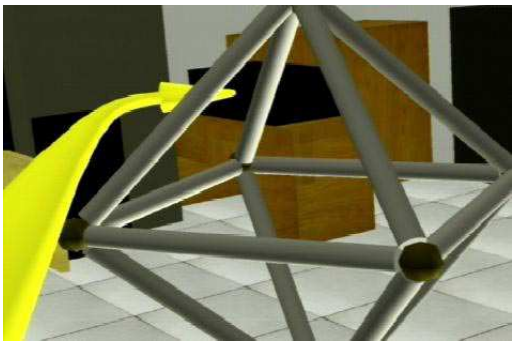

FIG. 4.4 – Pointeur flexible

4.5 Coopération entre plusieurs participants

Commençons d'abord par expliquer la nuance entre deux termes qui sont souvent utilisés conjointement. Ces deux termes sont : coopération et collaboration. La coopération est le fait de participer à une tâche ou un travail, tandis que la collaboration est le fait d'agir conjointement avec quelqu'un. W. Broll a présenté dans [Bro95] l'exemple figuré dans 4.5 pour montrer la différence entre deux utilisateurs qui coopèrent (en haut) et ceux qui collaborent (en bas). Dans le cas d'une coopération, les utilisateurs peuvent interagir avec l'objet virtuel (une table) chacun à son tour, Tandis que dans le cas d'une collaboration, les utilisateurs peuvent interagir avec la table simultanément. Un environnement virtuel collaboratif est forcément coopératif, l'inverse n'est pas vrai.

Pour Roschelle et Teasley [RT95] la collaboration est le processus qui consiste, d'une part à résoudre un problème à plusieurs, et d'autre part à construire et maintenir une représentation commune du problème.

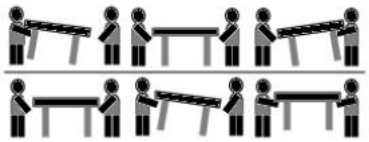

FIG. 4.5 – Coopération vs Collaboration

Collaboration is a coordinated, synchronous activity that is the result of a continued attempt to construct and maintain a shared conception of a problem.

Au sein d'un environnement virtuel, la collaboration impose plus de contraintes que la coopération. Par exemple, pour assurer la collaboration, le support d'exécution d'un environnement virtuel distribué doit offrir des services comme l'accès concurrent à un objet, ou la gestion des interactions distribuées.

La coopération peut avoir plusieurs buts : enseignement, conception, direction, surveillance, etc. Dans un environnement virtuel coopératif, l'objectif d'un utilisateur est d'accomplir quelque chose de spécifique tout en coopérant avec d'autres utilisateurs. Comme par exemple effectuer un travail, apprendre quelque chose, ou acheter un produit. Les utilisateurs n'ont pas forcément le même niveau de savoir et d'expérience. Par exemple, des visiteurs d'un musée n'ont pas les mêmes connaissances que leur guide qui doit être un utilisateur expérimenté du musée.

Dans un environnement coopératif, il est peut être important que tous les utilisateurs partagent le même sentiment d'un état commun limité dans le temps et l'espace. L'intensité avec laquelle ce sentiment est donné à tous les utilisateurs détermine le niveau de conscience de l'état de l'environnement [GG99]. Un niveau élevé de conscience de l'état de l'environnement virtuel conduit à une meilleure coopération entre les utilisateurs. Une conséquence de la conscience est la possibilité d'apprendre indirectement à partir des activités d'autres utilisateurs. Cela peut conduire à la possibilité d'utiliser des formes de communications indirectes ("suis moi", "prends ça" ...) pour la coordination des différentes actions entre les utilisateurs [BP01].

Le problème du choix de la bonne forme de métaphore donnant la conscience aux utilisateurs est souvent dépendant de l'application. Les informations liées à la coopération ne sont pas toutes nécessaires pour accomplir une tâche dans un environnement donné [BP01].

Pinho et al. ont présenté dans [PBF02] un modèle qui supporte des techniques de manipulations collaboratives dans des environnements virtuels. Leur modèle permet deux classifications de techniques de manipulations coopératives : homogène et hétérogène. La classe de technique homogène permet à deux utilisateurs d'utiliser la même technique d'interaction. Dans leur expérience, deux utilisateurs doivent placer un objet dans une boîte. L'un d'entre eux s'occupe de la position

de l'objet, tandis que l'autre s'occupe de son orientation. Les deux utilisateurs utilisent la technique de la main virtuelle pour interagir avec l'objet. La classe de technique hétérogène permet à deux utilisateurs d'utiliser des techniques différentes d'interaction. Dans ce cas l'un des utilisateurs utilise la technique de la main virtuelle, l'autre utilise la technique du pointeur virtuel. Tous les deux interagissent sur le même objet pour le placer dans sa boîte.

Le modèle de Pinho et al. est un exemple parmi d'autres qui offre la coopération dans un environnement virtuel. Presque tous les systèmes de développement d'environnements virtuels cités dans les chapitres précédents supportent la coopération entre les utilisateurs. Parmi les plates-formes de réalité virtuelle Françaises qui offrent la coopération on trouve : OpenMASK [MAC+02, DL02], CAVALCADE [TBG+00], SPIN3D [DDS+99].

4.6 Incertitudes liées à un problème réseau

Les problèmes qui surgissent au niveau réseau endommagent les applications de réalité virtuelle distribuée. Parmi les problèmes inévitables, le plus fréquent est peut être la latence. Dans un environnement virtuel distribué au dessus d'un réseau du type WAN, la latence peut être importante et imprévisible. Ainsi, la plupart de ceux qui ont déjà testé des logiciels distribués sur *Internet* ont probablement éprouvé l'effet de la latence surtout lorsqu'il s'agit de l'affichage temps réel des utilisateurs sur l'écran. Par exemple, plusieurs systèmes de développement d'environnements virtuels distribués fournissent à l'exécution une représentation de chaque utilisateur. Dans le cas de latence, la représentation d'un utilisateur peut sembler se figer pendant qu'il se déplace. La latence est aussi très apparente dans le cas d'un travail coopératif fortement synchrone (*closely-coupled*) et nécessite des mises-à-jour continues concernant les activités d'autres utilisateurs dans la scène [SSL96]

Les travaux de recherche concernant les stratégies qui traitent le délai ou la latence ont tout d'abord été faites au début au sein des communautés réseau et multimédia. Ces travaux ont été orientés vers la réduction du délai au niveau de la couche réseau [BFT99, HOK97]. Par exemple, beaucoup de travaux ont déjà examiné l'effet de la latence dans le cas de la transmission multimedia comme du flux audio ou vidéo [Kar96, RJ01]. Plus tard la communauté de la réalité virtuelle distribuée s'est intéressée à ce domaine. Toutefois, les recherches menées dans ce domaine sont beaucoup moins nombreuses que celles qui ont été faites par les autres communautés. Les chercheurs en réalité virtuelle ont orienté leur recherches vers la coordination d'actions dans les systèmes collaboratifs dans le cas d'un délai [Gut01, PK99]. Ces recherches ont pu démontrer l'importance de montrer l'existence du délai comme nous allons le détailler dans le paragraphe

4.6.1.

Le délai peut avoir des effets négatifs sur la collaboration dans un environnement virtuel distribué et partagé par plusieurs utilisateurs. Ces effets négatifs se manifestent dans la mauvaise compréhension de la situation partagée [GBD+04]. La réponse du système à l'action d'un utilisateur peut être retardée à cause du délai. Dans ce cas l'utilisateur va croire que son action était sans effet ou qu'elle a échoué.

4.6.1 Conscience de l'incertitude

Vaghi et al. [VGB99] ont réalisé une expérimentation très intéressante qui montre comment le délai affecte le comportement des utilisateurs d'un environnement virtuel distribué. Elle se présente sous la forme d'un jeu virtuel entre deux participants qui jouent avec une balle. L'un d'entre eux interagit à partir d'un ordinateur connecté au réseau sans délai ajouté artificiellement. L'autre interagit à partir d'un site qui est soumis à un délai ajouté artificiellement d'une manière croissante. Comme le délai croît avec le temps, la perception du jeu du deuxième utilisateur commence à produire des irrégularités croissantes. Vaghi et al. ont remarqué que les utilisateurs réagissent différemment vis-à-vis des incohérences provoquées par le délai. Les utilisateurs les plus habiles ont pu attribuer l'incohérence remarquée au délai réseau. Ces utilisateurs ont par la suite adopté des stratégies pour faire face à ce problème, comme par exemple attendre le plus longtemps possible avant de frapper la balle et essayer de diminuer le pas du jeu. Vaghi et al. ont aussi reporté que ces utilisateurs ont même essayé d'évaluer le taux du délai. Cependant, d'autres utilisateurs n'ont pas su remarquer la présence du délai. Ils ont formé d'autres hypothèses pour expliquer le problème comme par exemple croire que les mécanismes du jeu ne fonctionnent pas correctement. D'autres sujets ont pu identifier l'existence du délai mais ils n'ont pas pu comprendre ses conséquences.

Cette expérience a bien montré l'importance pour un utilisateur d'être conscient du problème du délai. Une mauvaise compréhension peut entrainer qu'un utilisateur perde confiance dans le système et perde aussi son habilité à interagir au sein d'un tel système. Pour qu'un utilisateur soit conscient de l'existence du délai, il faut attirer son attention sur ce phénomène lors de son occurrence.

4.6.2 Métaphore de l'incertitude

Fraser et al. ont décidé de montrer explicitement le délai en utilisant quelques métaphores visuelles présentées dans [FGV+00]. En montrant le délai, ils estiment que les utilisateurs ne vont plus voir cela comme un problème mais plutôt comme un phénomène auquel il faut s'adapter. Ils ont appelé ce phénomène : le phéno-

mène du délai induit (*delay induced phenomena*). Ils ont implémenté un système qui calcule une estimation de la différence maximale entre la position d'un avatar associé à un utilisateur et celle où un autre utilisateur peut le percevoir. Cette disparité est basée sur deux facteurs : la mesure du délai entre les utilisateurs et la vitesse et direction du mouvement de l'avatar. Par exemple, si un avatar se déplace à une vitesse de 5 m/s en présence du délai de 420 millisecondes, le resultat est une sphère de 2.1 mètres de rayon. Cette sphère englobe l'avatar et représente toutes les positions incertaines possibles (Fig. 4.6). En plus de la sphère, un widget 3D au dessus de l'avatar est affiché indiquant le niveau du délai pour l'utilisateur associé.

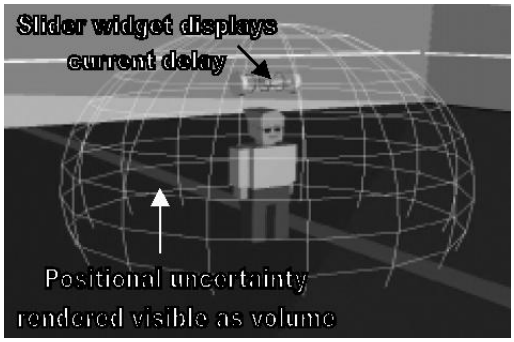

FIG. 4.6 – La sphère représente toutes les positions possibles

Une autre étude plus récente menée par Gutwin et al. [GBD+04] a présenté d'autres types de métaphores : les décorateurs qui sont une sorte d'ornement visuel ajouté à la représentation d'un objet. Deux familles essentielles de décorateurs ont été discutées dans cette étude : les décorateurs qui montrent l'existence du délai sur le réseau ainsi que sa valeur, et les décorateurs qui montrent l'état passé ou futur d'un objet.

Les décorateurs montrant l'existence du délai sur le réseau peuvent utiliser plusieurs techniques comme par exemple l'utilisation des différentes couleurs selon la valeur du délai. Les décorateurs montrant le passé d'un objet utilisent un trait montrant l'acheminement de l'objet dans le passé comme le montre la figure 4.7. De la même manière, les positions futures d'un objet peuvent être schématisées par un trait pointillé comme nous pouvons voir dans la figure 4.8.

L'inconvénient des décorateurs est qu'ils ne montrent que la position passée ou future d'un objet. Dans un environnement virtuel complexe l'état d'un objet ne peut pas être réduit à sa position dans l'espace. Par exemple la modification de l'état d'un objet suite à une interaction n'est pas prise en compte dans l'étude de Gutwin et al.

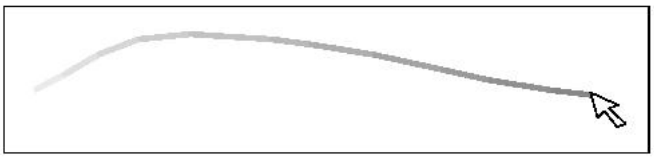

FIG. 4.7 – Un trait montrant le chemin passé d'un pointeur

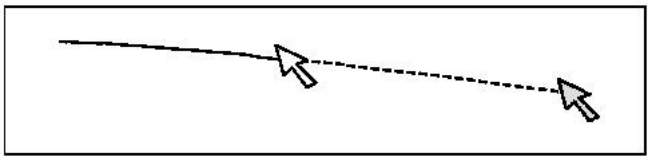

FIG. 4.8 – Un trait pointillé montrant l'éventuel chemin futur d'un pointeur

4.7 Synthèse

Dans ce chapitre nous avons présenté les différentes manières dont les utilisateurs peuvent "être présent", interagir, et coopérer au sein des environnements virtuels. À ce niveau de la réalité virtuelle la psychologie humaine se mêle avec la technologie pour assurer aux utilisateurs un certain niveau de conscience des changements d'état de l'environnement virtuel. Nous avons pu remarquer que le problème des effets non souhaitables provoqués par une perturbation au niveau réseau est parmi les problèmes les moins étudiés au sein de la communauté de la réalité virtuelle. Ce problème, bien qu'il agisse directement sur la performance d'une simulation distribuée, a également été longtemps délaissé par les chercheurs de la communauté réseau. Ces derniers ont fait des études permettant de réduire le délai au niveau réseau mais ne sont pas parvenus à s'en débarrasser entièrement. Nous avons présenté quelques études faites par la communauté de réalité virtuelle concernant ces problèmes, elles sont considérées comme un premier pas en vue de les résoudre.

Deuxième partie

Contributions

Chapitre 5

Gestion de la synchronisation en cas de problème réseau

5.1 Introduction

Les supports d'exécution d'environnements virtuels distribués, comme nous avons pu le constater dans les chapitres prédédents, comportent une infrastructure de communication. Cette infrastructure assure la communication entre les entités et les utilisateurs qui sont dispersés géographiquement et interconnectés par un réseau. La qualité de service offerte par le réseau d'interconnexion agit directement sur la performance des systèmes d'environnements virtuels distribués. Par exemple pour diffuser les changements d'états d'un objet qui a été manipulé par un utilisateur, la couche de communication associée à l'environnement virtuel distribué doit communiquer ces changements aux autres sites via le réseau. Un système distribué doit être conçu de manière à garantir que tous ses éléments opèrent d'une façon déterministe. Un tel système doit garantir que les données soient disponibles et livrées à tous les nœuds participants.

Le temps est un facteur critique pour certains environnements virtuels distribués qui exigent une forte synchronisation entre sites distants. Il est fondamental de communiquer les données et les mises-à-jour le plus rapidement possible. Un mécanisme de synchronisation est nécessaire pour coordonner et orchestrer les différents processus.

Malheureusement, les réseaux ont quelques limitations (reroutage, latence, bande passante) que nous avons discutées dans les chapitres précédents. De toute manière, comme Gutwin et al. l'ont signalé dans [GBD+04], tant qu'il y aura une distance physique entre deux nœuds, le problème de la latence ne sera jamais entièrement résolu. À cause de cela, aucun environnement virtuel distribué à architecture non centralisée ne sera jamais capable de faire partager une expérience parfaitement identique à tous les utilisateurs en même temps. En revanche, le

rôle d'un système d'environnement virtuel est de minimiser autant que possible les effets des problèmes réseaux, notamment la latence.

Dans le monde des systèmes distribués il y a deux modèles principaux de synchronisation de données : le modèle centralisé et le modèle distribué. Le modèle centralisé privilégie la communication et la synchronisation par une mémoire commune partagée alors que le modèle distribué privilégie la communication et la synchronisation par passage de messages. Dans nos travaux nous nous sommes intéressé au modèle distribué par passage de messages pour garantir la cohérence.

Les modèles de distribution classiques considèrent que la transmission est fiable et ne prennent pas en considération le cas d'un problème au niveau réseau. Pourtant, une forte latence ou une déconnexion peuvent endommager la synchronisation entre les sites. C'est pourquoi dans ce chapitre nous présentons notre modèle de synchronisation, qui peut gérer la perte momentanée (ou de longue durée) de communication avec un ou plusieurs sites distants, ainsi que sa mise en œuvre.

5.2 Pourquoi une synchronisation tolérante ?

5.2.1 Synchronisation par passage de message

Il peut y avoir plusieurs raisons de vouloir utiliser des environnements virtuels distribués. Tout d'abord, cela permet d'obtenir des mondes animés riches en distribuant le calcul à travers un nombre important de machines. Cela permet aussi une coopération à plusieurs dans un environnement virtuel partagé. Dans tous les cas, le même environnement virtuel va être partagé en totalité ou en partie par plusieurs machines.

Un environnement virtuel est souvent peuplé de plusieurs objets virtuels. Ces objets sont assignés à un ou plusieurs processus qui constituent l'environnement virtuel tout entier. Dans un tel cas, les objets doivent être répliqués sur les différentes machines participantes. Une solution commune pourrait être de contrôler chaque processus à l'aide d'un contrôleur dont le rôle serait de maintenir et garantir la communication avec d'autres processus. Mais en distribuant les processus sur les différentes machines un autre problème se pose : les objets virtuels doivent être distribués à leur tour. L'algorithme de distribution des objets sur les sites doit garantir le même état et la même vision de ces objets par les utilisateurs : les objets doivent toujours avoir la même visibilité à partir des différents sites utilisateurs. Cela veut dire que, si cela s'avère nécessaire, tous les utilisateurs doivent perçevoir le même environnement virtuel en même temps selon le principe WYSIWIS [1].

[1] What You See Is What I See

Ce principe est normalement garanti en dupliquant les objets à travers les différents sites sur lesquels l'environnement virtuel est distribué. Plusieurs techniques et modèles de duplications d'objets existent mais le principe est toujours la même. Les duplicas d'un objet sont classiquement appelés *fantômes*, *miroirs*, ou *proxies* selon la littérature alors que l'objet original est en général appelé *référentiel*. Normalement les miroirs sont reliés à leurs objets originaux pour recevoir les mises-à-jour, soit d'une manière périodique, soit à chaque fois que les valeurs changent. Les contrôleurs doivent donc synchroniser les différents processus, et aussi les différents objets virtuels dupliqués sur ces processus.

5.2.2 Les effets d'un problème réseau sur la synchronisation

Le facteur réseau affecte directement, comme nous avons pu le constater dans les chapitres précédents, le bon fonctionnement des environnements virtuels distribués. Dans ce chapitre nous nous intéressons à l'effet d'un problème réseau sur la synchronisation au niveau noyau du support d'exécution d'un environnement virtuel distribué. Nous rappelons que le type de problème réseau le plus fréquemment envisagé est le délai et les petites déconnexions. Or, si le délai de transmission d'un site croît pendant une simulation distribuée, les conséquences peuvent être catastrophiques.

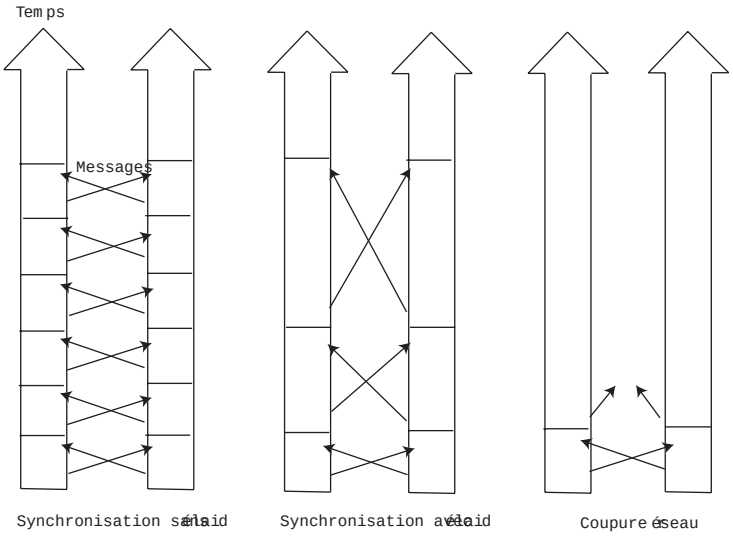

FIG. 5.1 – Synchronisation sans et avec délai

Les problèmes qui surgissent sur le réseau peuvent entraîner différentes perturbations dans le noyau du support d'exécution des environnements virtuels distribués. Ces perturbations noyau peuvent aller jusqu'au blocage total du mécanisme de synchronisation au niveau des contrôleurs. Ceci se produit lorsque

les messages de synchronisation sont retardés ou même perdus. Dans ce cas, les sites doivent attendre plus de temps que prévu, ce qui bloque la simulation en attendant l'arrivée des messages comme le montre la figure 5.1.

5.3 Conception d'un mécanisme de synchronisation tolérant aux problèmes réseau

Quel que soit le modèle de distribution de données adopté, nous ne pouvons pas éviter une communication entre les sites partageant un même environnement virtuel. Cette communication devient une nécessité primordiale dans le cas où les utilisateurs interagissent au sein de l'environnement. Une des méthodes communes de synchronisation dans les systèmes distribués est l'utilisation de messages périodiques de synchronisation. Cette méthode garantit une synchronisation forte entre tous les sites.

Pour qu'une application fonctionne bien, normalement elle exige un temps de réponse très réduit. Les troubles et le délai au niveau réseau empêchent les événements et les mises-à-jour d'être livrés à temps. Par conséquent, le temps réel ne peut pas être atteint. Dans le contexte de la simulation d'un environnement virtuel distribué, on doit alors choisir l'un des deux scénarios suivants :

1. paralyser la simulation sur tous les sites en attendant l'arrivée d'un message de synchronisation, ce qui peut provoquer un blocage définitif.
2. laisser continuer la simulation même en présence d'un fort délai ou déconnexion.

Le premier choix est nuisible pour la simulation surtout si le délai est trop important ou si une déconnexion se produit. Le deuxième choix a pour conséquence de diviser l'environnement virtuel en plusieurs environnements parallèles. Ceci est dû au fait que plusieurs utilisateurs pourraient interagir en même temps sur les mêmes objets sans que leurs modifications de l'environnement virtuel ne soient diffusées à cause du délai ou de la déconnexion. Ce deuxième scénario est donc problématique également.

Notre proposition est en fait une combinaison des deux scénarios précédents. Nous choisissons de laisser continuer la simulation après un problème réseau en n'immobilisant que les parties de l'environnement virtuel qui ont un état incertain à cause de la non réception des mises-à-jour. Cette partie non-certaine de l'environnement virtuel est bloquée pour des considérations de cohérence et dans le but de minimiser la divergence entre les différents points de vues des utilisateurs. Notre philosophie est donc de permettre à une simulation distribuée de continuer à évoluer dans le temps, même pendant une perturbation réseau. De cette manière l'environnement virtuel ne serait pas hors d'usage en attendant la réception des

mises-à-jour en provenance d'autres sites comme dans les systèmes utilisant une synchronisation forte. Notons que les parties bloquées de l'environnement virtuel ne sont pas forcément les mêmes sur tous les sites.

5.4 Mise en œuvre dans OpenMASK

Dans cette section nous allons détailler la mise en œuvre de notre nouvel algorithme au sein de la plate-forme de simulation distribuée OpenMASK. Avant de détailler notre implémentation, il est primordial de présenter quelques concepts de base d'OpenMASK ainsi que l'algorithme de synchronisation antérieur à ces travaux de thèse.

5.4.1 Distribution dans OpenMASK

Selon le modèle conceptuel d'OpenMASK, l'objet essentiel est appelé objet simulé. Un objet simulé possède un comportement autonome qui est activé à une certaine fréquence [DCDK98]. Les données publiques d'un objet sont attribuées à un certain nombre de sorties, grâce auxquelles d'autres objets peuvent lire ces valeurs rendues publiques. Ces valeurs sont mises-à-jour par un mécanisme de calcul interne à l'objet simulé. Pour pouvoir lire des valeurs à partir d'une sortie donnée, d'autres attributs sont nécessaires : les entrées. Un objet crée des entrées destinées à être branchées sur des sorties à lire. Tous les objets ont la faculté d'interagir avec n'importe quel autre objet pendant une simulation.

La distribution de l'environnement virtuel dans OpenMASK est réalisée suivant le paradigme de référentiels et de miroirs. Un référentiel est une sorte d'encapsulation de l'objet réel. Lorsqu'un objet a besoin de données possédées par un autre objet localisé sur un autre processus, un miroir de ce dernier est automatiquement créé localement. C'est à la charge du noyau de maintenir la cohérence entre les valeurs des sorties de l'objet référentiel et celles de son miroir. Pour cela, tous les miroirs d'un même référentiel doivent être reliés à ce référentiel. Une connexion par flots de données est établie entre les référentiels et leurs miroirs.

Sur chaque processus, un contrôleur gère la synchronisation et la communication entre les différents processus (y compris la communication entre les référentiels et leurs miroirs). La communication entre les contrôleurs est synchronisée.

5.4.2 Algorithme de synchronisation

Les premiers algorithmes qui implémentent la distribution au sein d'OpenMASK ont été présenté dans [DCDK98]. Il s'agissait d'une mise en œuvre simpliste et loin d'être optimale. Plus tard, ces algorithmes ont été modifiés et raffinés suite aux travaux de thèse de D. Margery présentés dans [Mar01].

Margery a présenté dans [Mar01] les différentes approches possibles pour qu'un nœud A puisse lire la valeur d'une donnée gérée par un nœud B :

- s'assurer, par construction du programme, que la donnée à lire est déjà présente sur le nœud A. C'est le cas d'une approche synchrone.
- demander la meilleure valeur possible au nœud B lorsqu'elle est nécessaire pour le nœud A. À l'instant où cette valeur parvient au nœud A, elle sera peut-être périmée, mais elle sera néanmoins utilisée.
- utiliser la dernière valeur reçue de B, en supposant que B a transmis une nouvelle valeur si cela s'est avéré nécessaire.
- metre en œuvre ou utiliser un modèle mémoire distribué à cohérence séquentielle ou de cohérence moins forte pour partager les données.

Le bilan d'après [Mar01] était qu'il n'est possible d'obtenir une exécution performante que si les messages assurant la cohérence des données transitent sur le réseau de manière parallèle au calcul de l'évolution de l'environnement virtuel.

Le contrôle du degré de relâchement de la cohérence n'est pas simple à mettre en œuvre. En tenant compte de tous ces paramètres Margery a proposé le mécanisme de contrôle et de synchronisation présenté dans l'algorithme 1.

Chaque contrôleur sur chaque processus exécute l'algorithme 1 après son initialisation. La synchronisation entre les différents contrôleurs se fait par des messages de synchronisation qui circulent entre les processus. Chaque contrôleur produit donc son message de synchronisation qui est diffusé vers les autres contrôleurs. Ensuite chaque contrôleur tente de recevoir et traiter les messages de synchronisation émis par les autres. S'il y a des contrôleurs qui n'ont pas réussi à délivrer leurs messages à temps, ils seront placés dans une liste spéciale. Ensuite, une attente bloquante est appliquée afin d'attendre la réception de messages de synchronisation à partir de tous les contrôleurs placés dans la liste.

Une valeur de latence est ajoutée à cet algorithme pour paramétrer le degré d'ancienneté des valeurs lues à partir d'un nœud. Par exemple, une latence égale à zero veut dire que l'ancienneté des données lues ne dépasse pas un pas de simulation.

5.4.3 Synchronisation en cas de problème réseau

Comme nous l'avons mentionné dans les paragraphes précédents, notre objectif est de trouver une façon de permettre à une simulation distribuée de continuer même en présence de problème au niveau réseau. L'algorithme proposé par Margery ne peut pas fonctionner correctement dans le cas d'un problème réseau car dans ce cas une forte latence va pénaliser tous les utilisateurs en ralentissant le rythme de la synchronisation et donc de la simulation, et dans le cas d'une déconnexion on aura même un blocage total de tous les utilisateurs. Dans de tels cas, le mécanisme présenté dans l'algorithme 1 va boucler indéfiniment jusqu'à réception

Algorithme 1 Algorithme de contrôle et de synchronisation présenté dans [Mar01]

```
calculDuPasDeSimulation () ;
pour tout contrôleur ≠ this faire
  émettreMessageSynchro (contrôleur) ;
fin pour
pour tout contrôleur ≠ this faire
  messsageReçu ← tenterUneRéceptionNonBloquante (contrôleur) ;
  tant que messageReçu ≠ φ faire
    traiterMessage (messageReçu) ;
    contrôleur.dateDuDernierMessageReçu = messageReçu.dateEmission () ;
    messageReçu ← tenterUneRéceptionNonBloquante (contrôleur) ;
  fin tant que
fin pour
pour tout contrôleur ≠ this faire
  si contrôleur.dateDuDernierMessageReçu + Latence + duréePasDeTemps <
  dateCourante () alors
    listeContrôleursRéceptionObligatoire.ajouter (contrôleur) ;
  finsi
fin pour
tant que listeContrôleursRéceptionObligatoire ≠ φ faire
  pour tout contrôleur ∈ listeContrôleursRéceptionObligatoire faire
    messageReçu ← réceptionBloquante (contrôleur) ;
    traiterMessage (messageReçu) ;
    contrôleur.dateDuDernierMessageReçu ← messageReçu.dateEmission () ;
    si contrôleur.dateDuDernierMessageReçu + Latence + duréePasDeTemps
    ≥ dateCourante () alors
      listeContrôleursRéceptionObligatoire.ôter (contrôleur) ;
    finsi
  fin pour
fin tant que
```

des messages de synchronisation attendus qui n'arriveront peut être jamais.

Avec notre approche nous allons éviter les attentes bloquantes. Aucun processus ne sera attendu plus d'un certain temps. Pour atteindre cet objectif nous avons dû modifier plusieurs structures dans le noyau distribué d'OpenMASK. Nos travaux ont permis d'aboutir à un nouvel algorithme de synchronisation entre contrôleurs, cet algorithme est présenté dans la section suivante.

5.4.3.1 Nouvel algorithme de synchronisation

Selon notre approche, les processus sont rangés en deux catégories différentes : les processus actifs et les processus retardés. Par défaut tous les processus sont considérés comme étant actifs. Si les messages de synchronisation en provenance d'un certain contrôleur n'ont pas été livrés à temps, ce dernier sera attendu pendant un certain temps. Une fois ce temps écoulé, le processus en question sera considéré comme étant retardé et sera enlevé de la liste des processus actifs pour être inséré dans la liste des processus retardés. L'algorithme 2 présente ce nouveau mécanisme de synchronisation.

Cet algorithme commence comme son prédécesseur par le calcul et l'envoi de messages de synchronisation. Ensuite, si la liste des contrôleurs retardés était non vide, on tente une réception non bloquante pour tester si la communication est rétablie avec ces contrôleurs. Ensuite chaque contrôleur tente de recevoir et traiter les messages de synchronisation émis par les autres. S'il y a des contrôleurs qui n'ont pas réussi à délivrer leurs messages à temps, ils seront placés dans une liste spéciale (même comportement que l'algorithme 1). Ensuite, une attente à durée déterminée est appliquée afin d'attendre la réception de messages de synchronisation à partir de tous les contrôleurs placés dans la liste. Si la durée est écoulée avant la réception de messages de synchronisation, les contrôleurs placés dans la liste spéciale sont considérés comme retardés et ils sont ôtés de la liste des contrôleurs actifs.

5.4.3.2 Discussions techniques

Nous avons implémenté l'algorithme 2 dans le noyau distribué d'OpenMASK qui se base sur une couche d'interface par passage de messages. Cette couche utilise la machine virtuelle parallèle PVM qui est une couche logicielle de programmation qui permet à des réseaux d'ordinateurs hétérogènes d'être vus comme une seule ressource de calcul.

En effet, l'environnement logiciel composé d'OpenMASK d'un côté et de PVM de l'autre côté impose quelques contraintes pour l'implémentation de notre algorithme. Pour mieux comprendre ceci nous allons présenter rapidement quelques caractéristiques de PVM.

Algorithme 2 Algorithme de contrôle et de synchronisation tolérant au délai

```
calculDuPasDeSimulation () ;
pour tout contrôleur ≠ this faire
  émettreMessageSynchro (contrôleur) ;
fin pour
si listeContrôleursRetardés ≠ φ alors
  pour tout contrôleur ∈ listeContrôleursRetardés faire
    messsageReçu = tenterUneRéceptionNonBloquante (contrôleur) ;
    si messageReçu ≠ φ alors
      listeContrôleursActifs.ajouter (contrôleur) ;
      listeContrôleursRéceptionObligatoire.ôter (contrôleur) ;
      traiterMessage (messageReçu) ;
      contrôleur.dateDuDernierMessageReçu ← messageReçu.dateEmission
      () ;
    finsi
  fin pour
finsi
pour tout contrôleur ∈ listeContrôleursActifs faire
  si contrôleur.dateDuDernierMessageReçu + Latence + duréePasDeTemps <
  dateCourante () alors
    listeContrôleursRéceptionObligatoire.ajouter (contrôleur) ;
  finsi
fin pour
tant que (listeContrôleursRéceptionObligatoire ≠ φ) et (tempsDAttente <
tempsToléré) faire
  pour tout contrôleur ∈ listeDeTousLesContrôleurs faire
    messageReçu ← tenterUneRéceptionNonBloquante (contrôleur) ;
    si messageReçu ≠ φ alors
      si contrôleur ∈ listeContrôleursRetardés alors
        listeContrôleursActifs.ajouter (contrôleur) ;
        listeContrôleursRetardés.oter (contrôleur) ;
      finsi
      traiterMessage (messageReçu) ;
      contrôleur.dateDuDernierMessageReçu ← messageReçu.dateEmission
      () ;
      listeContrôleursRéceptionObligatoire.ôter (contrôleur) ;
    finsi
  fin pour
  incrémenter tempsDAttente ;
fin tant que
si listeContrôleursRéceptionObligatoire ≠ φ alors
  pour tout contrôleur ∈ listeContrôleursRéceptionObligatoire faire
    listeContrôleursRetardés.ajouter (contrôleur) ;
    listeContrôleursRéceptionObligatoire.ôter (contrôleur) ;
    listeContrôleursActifs.ôter (contrôleur) ;
  fin pour
finsi
```

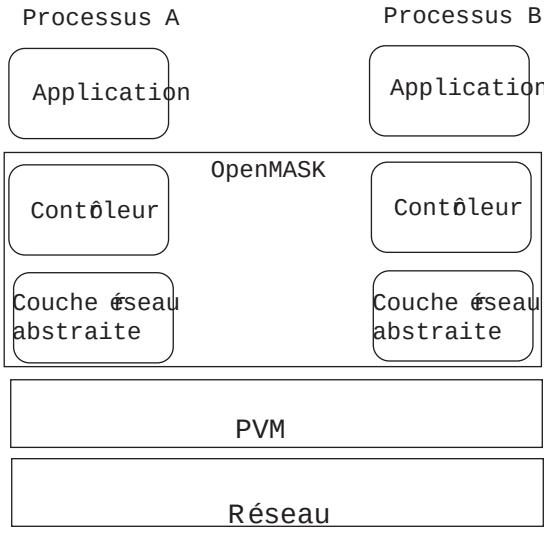

FIG. 5.2 – Les couches OpenMASK

PVM (*Parallel Virtual Machine*) PVM est une surcouche de TCP/IP garantissant le transfert de données entre plusieurs programmes, et permettant la synchronisation éventuelle de ces programmes. La philosophie de PVM est de donner à l'utilisateur la vision d'une machine parallèle virtuelle à mémoire distribuée composée d'un ensemble de machines séquentielles, parallèles ou vectorielles.

Une session PVM est normalement constituée de deux composants :

- un processus démon *pvmd* : c'est un processus qui gère les opérations des processus utilisateurs d'une application PVM et coordonne les communications entre les machines. Il y a un démon par machine composant la machine virtuelle. Les processus utilisateurs communiquent entre eux par l'intermédiaire des démons.
- la bibliothèque *pvmlib* : c'est un ensemble de routines d'interface pour le programmeur. Ces routines permettent d'initialiser et terminer un processus, d'emballer, envoyer et recevoir des messages, de synchroniser des groupes de processus etc. Ces routines ne font pas de communications directes avec les autres processus mais envoient des commandes au démon local duquel ils reçoivent des informations en retour.

PVM identifie les processus par des tids (*Task ID*). La communication entre processus se fait en définissant tout d'abord le mode d'encodage des données lors de la communication, puis en créant un buffer pour la communication, et enfin en remplissant ce buffer.

PVM fournit deux mécanismes de routage pour les messages échangés par les applications : routage direct et indirect. Le routage direct permet par exemple à deux tâches $T2$ et $T3$ de communiquer directement comme le montre la figure 5.3.

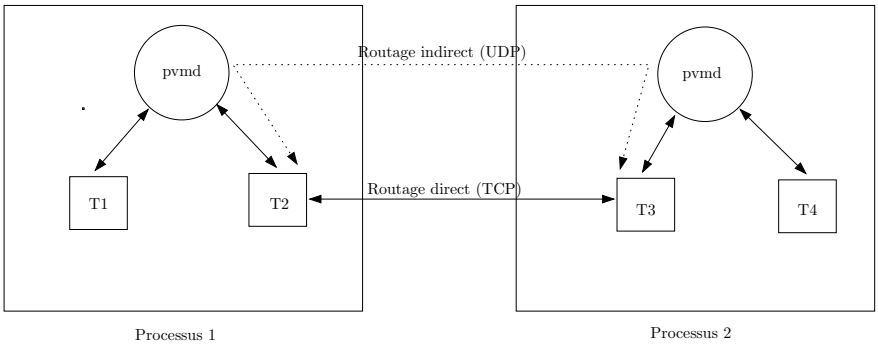

FIG. 5.3 – Routage direct et indirect

La communication directe se fait par une connexion suivant le protocol TCP. Par contre, le routage indirect exige qu'un message envoyé par exemple de $T2$ à $T3$, doivent passer à travers les démons des deux processus en question. La communication entre les démons utilise des connexions UDP. Par contre la communication entre une tâche et un démon ou inter-tâche se fait par une connexion TCP. Le choix du mécanisme de routage est contrôlé par le programmeur. Par défaut les messages sont routés indirectement.

PVM garantit que des messages envoyés d'une tâche à une autre soient reçus selon l'ordre d'envoi.

Remarques d'implémentation Pour une raison de performance nous avons choisi le routage direct entre les tâches au lieu du routage indirect proposé par défaut. La raison est que la communication inter-démons est très coûteuse (tâche1 <-> démon1 <-> démon2 <-> tâche2) par rapport à une communication directe (tâche1 <-> tâche2) surtout dans le cas d'un délai sur le réseau. Selon nos expériences, un délai ou une déconnexion provoque un blocage du démon dans le cas de routage indirect.

Après une première série de tests suite à une implémentation de notre algorithme en utilisant le routage direct, nous avons remarqué le phénomène suivant : en provoquant une coupure réseau, plus le temps de la coupure est importante, plus le processus qui a été déconnecté a besoin de temps pour rétablir l'état actuel du monde virtuel après sa reconnexion. La raison est la fiabilité excessive de PVM qui est devenue un inconvénient dans notre contexte. N'oublions pas que la synchronisation dans OpenMASK est assurée par passage de messages. Or les messages de synchronisation sont produits et envoyés par chaque processus à chaque pas de simulation. Après déconnexion, tous ces messages sont stockés dans un tampon spécial PVM au niveau socket en attendant le rétablissement de la connexion (Fig. 5.4). Après reconnexion, le temps de traitement de tous ces messages (au niveau PVM + au niveau noyau OpenMASK) devient trop cou-

teûx et provoque le phénomène cité au dessus. PVM n'offre aucune interface pour manipuler ou vider ce tampon. Dès qu'un message est envoyé dans PVM, ce message ne peut qu'être délivré. La solution intuitive est donc de ne pas envoyer de messages de synchronisation à un processus qui est considéré comme retardé.

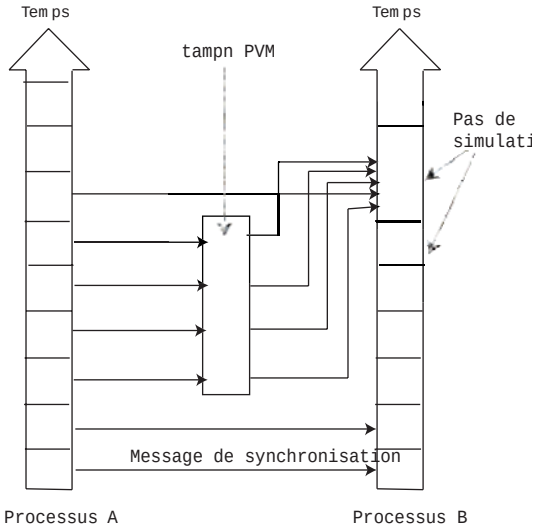

FIG. 5.4 – Pendant la déconnexion les messages de synchronisation sont stockés dans un tampon PVM

À ce stade, le problème n'est pas encore résolu, *i.e.* sans message de synchronisation OpenMASK ne peut plus détecter la reconnexion d'un processus. En effet, l'algorithme de synchronisation d'OpenMASK est totalement distribué et il n'y a pas de processus central qui orchestre la gestion des déconnexions et reconnexions. Ceci veut dire que le même algorithme est appliqué sur tous les processus. Prenons par exemple une simulation de trois processus A, B et C : si suite à une perturbation réseau nous avons perdu la communication avec le processus C, dans ce cas A va classifier C comme étant déconnecté et il ne lui envoie plus de messages de synchronisation. B va procéder de la même manière en déclarant C étant déconnecté. De son côté C va également classifier A et B comme étant des processus déconnectés et il bloque l'envoi de ses messages vers ces deux processus. Par conséquent un cas d'interblocage se produit après le rétablissement de la communication avec C. A ne pourra reconsidérer C comme actif qu'après que ce dernier ait envoyé un message à A, de la même manière C ne pourra reconsidérer A comme actif qu'après que ce dernier ait envoyé un message à C. De cette manière ces deux processus se considèrent déconnectés mutuellement (même chose pour B et C).

Deux solutions ont été proposées pour résoudre ce dernier problème :

- l'implémentation d'un service *ping* au sein d'OpenMASK pour détecter la perte ou le rétablissement de la connexion d'un ou de plusieurs processus.
- l'utilisation allégée de messages de synchronisation : une fois qu'un processus est classifié retardé, un seul message de synchronisation sera envoyé à ce processus avant de condamner tout envoi vers ce dernier. De cette manière le message de synchronisation envoyé après la déconnexion du processus en question servira comme indicateur de sa reconnexion.

Le deuxième choix a été adopté. Ce choix ne sature pas le tampon PVM (un seul message) et en même temps nous évite l'implémentation d'une structure parallèle (le *ping* personnalisé).

5.5 Synthèse et perspectives

La synchronisation des systèmes distribués est une tâche critique surtout pour les systèmes qui intègrent la notion de temps-réel. Les problèmes réseau rendent l'assurance de la synchronisation beaucoup plus compliquée en retardant ou même empêchant les messages d'arriver à temps. Nous avons présenté dans ce chapitre notre proposition d'un algorithme qui sait gérer la perte de communication avec une ou plusieurs machines pendant une simulation. Son principe est de permettre à une simulation de continuer même en présence de tels problèmes.

Bien sûr, le fait de permettre à un groupe d'utilisateurs de continuer leur simulation en présence de problèmes de communication ne suffit pas pour rendre cette continuation possible. Dans ce chapitre nous avons traité le problème au niveau couche de communication et noyau du système de réalité virtuelle. Dans les chapitres suivants nous continuons à traiter ce même problème à des niveaux plus hauts *i.e.* au niveau applicatif.

Suite à nos travaux, nous avons pu tirer quelques remarques concernant des améliorations futures au sein d'OpenMASK. Plusieurs types de messages sont encapsulés sous forme de messages de synchronisation dans OpenMASK. Toutes les mises-à-jour et les communications entre un référentiel et ses miroirs (et inversement) ainsi que des événements circulent sous forme de messages de synchronisation. Certains de ces messages sont plus critiques que d'autres. Suite aux problèmes décrits dans le paragraphe 5.4.3.2 nous avons signalé que les messages de synchronisation doivent être supprimés ou plutôt ne doivent même pas être envoyés du tout à un processus déconnecté momentanément. Un tel choix provoque parfois la perte de quelques informations critiques. Le choix idéal serait de pouvoir filtrer les messages de synchronisation de sorte à pouvoir conserver les événements critiques. Pour aller encore plus loin, il serait intéressant d'utiliser plusieurs protocoles de communication dépendant de la qualité de données échangées. Les messages non critiques pourraient être envoyés selon un protocole

non fiable.

La couche de communication utilisée actuellement est PVM. Cette bibliothèque a bien des avantages mais des inconvénients aussi. Bien que PVM offre le choix entre différentes sortes de routage lors de la communication inter-tâches, ce choix reste limité. Une communication directe entre les tâches est possible mais doit obligatoirement être appliquée via le protocole TCP. Donc ici PVM limitera nos possibilités quant à la proposition d'un schéma de filtrage de données et d'utilisation de différents protocoles selon le besoin.

Chapitre 6

Migration

6.1 Introduction

La migration d'objets d'un processus à un autre est un concept peu répandu au sein des plates-formes logicielles distribuées de réalité virtuelle. Nous avons en effet pu constater dans le chapitre 3 que très peu de plates-formes de développement d'environnements virtuels distribués offrent la possibilité de faire migrer un objet à partir de son site d'origine vers un autre site. Normalement, les objets sont associés statiquement à leur processus dès le début d'une simulation quelconque. L'association dynamique d'un objet à un processus au cours d'une simulation distribuée n'est envisageable qu'au sein des plates-formes logicielles AVIARY et WAVES détaillés dans le chapitre 3. Ces plates-formes utilisent la migration pour la répartition dynamique de charges de calcul entre les sites.

Nous allons étudier dans ce chapitre les raisons qui nous ont poussé à concevoir et implémenter un mécanisme de migration d'objets au sein de notre plate-forme de développement et d'exécution d'environnements virtuels distribués. Notre objectif principal est toujours de minimiser l'effet d'un problème réseau sur un environnement virtuel distribué. Dans cette optique, la migration d'objet d'un processus à un autre peut être fort utile dans le cas où un objet est considéré comme critique pour un utilisateur donné : l'accès à cet objet ne doit alors pas être interrompu. Or, un problème réseau est une raison majeure qui peut interrompre les manipulations faites par un utilisateur sur un objet donné surtout si cet objet n'est pas calculé localement sur le site de l'utilisateur. Dans ce cas, la migration peut assurer que l'objet soit calculé localement pour être à l'abri des perturbations réseau.

6.2 Pourquoi la migration ?

Les systèmes distribués ont recours à la migration pour assurer l'équilibrage dynamique de charges entre les nœuds. Dans notre contexte d'un système distribué d'environnements virtuels, nous proposons la migration pour une autre raison. Prenons l'exemple suivant : trois utilisateurs U_1, U_2, et U_3 partagent un même environnement virtuel coopératif et peuvent interagir à partir des sites S_1, S_2, et S_3 respectivement (Fig. 6.1). Trois objets simulés A, B, et C sont hébergés par cet environnement et ils sont associés d'une manière statique aux trois sites S_1, S_2, et S_3. L'objet A est calculé localement sur le site S_1 et il possède des miroirs sur S_2 et S_3. L'objet B est calculé sur S_2, il possède des miroirs sur S_1 et S_3, et enfin C est calculé sur S_3 et il possède des miroirs sur S_1 et S_2.

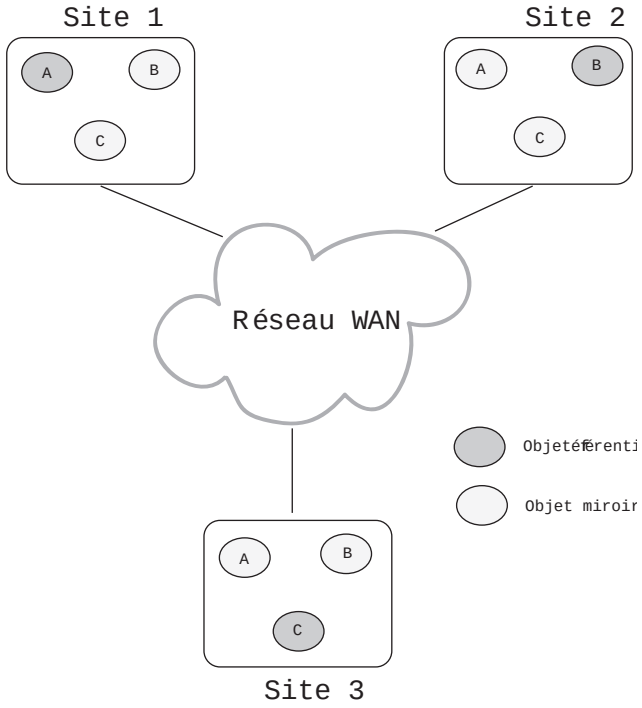

FIG. 6.1 – Trois utilisateurs interagissent au sein d'un environnement virtuel partagé

Supposons qu'au cours de la simulation, U_3 a été intéressé par la manipulation de l'objet A. Rappelons que l'utilisateur U_3 ne possède pas le calcul de A localement, il interagit seulement avec un miroir de A. Si la nature des interactions de U_3 sur A est très critique, la perte du contrôle de A à cause d'une perturbation réseau sera trop gênante voir catastrophique pour U_3. La conservation de l'objet de calcul de A localement sur le site S_3 serait donc très utile pour U_3. De

cette manière un problème réseau ne l'empêchera pas de continuer son travail en rapport avec l'objet A.

En effet, lors d'une déconnexion ou d'un fort délai, seulement un utilisateur peut conserver l'interactivité avec un objet donné. Cet utilisateur est celui qui possède l'unité de calcul associée à l'objet en question, c'est-à-dire dont l'unité de calcul est sur le site de l'utilisateur. Le principe est donc le suivant : nous ne pouvons pas empêcher la perte du contrôle d'un objet donné par tous les utilisateurs sauf un seul, mais nous pouvons par contre choisir celui qui peut garder l'interactivité de l'objet. Ce choix n'est possible que si nous pouvons faire migrer l'objet pour qu'il soit hébergé localement sur le site de l'utilisateur choisi. Un mécanisme de migration d'objets peut assurer qu'un utilisateur puisse avoir à la demande un objet donné si ses interactions avec cet objet sont classifiées comme étant "critique".

Une fois un mécanisme de migration en place, nous pouvons en profiter pour assurer d'autres services, parmi lesquels on peut citer l'équilibrage dynamique de charges, l'ajout et la suppression dynamique d'un site à une simulation en cours.

Nous allons étudier dans ce chapitre la conception et la réalisation d'un mécanisme de migration d'objets au sein du support d'exécution d'environnemenrts virtuels distribués. Par contre nous n'étudions pas les différentes méthodes possibles pour choisir l'utilisateur qui doit garder l'objet. Cet utilisateur pourrait être le plus intéressé par l'objet ou encore celui qui est le plus pénalisé par la perte du contrôle de l'objet.

6.3 Conception d'un mécanisme de migration d'objets

La conception d'un mécanisme de migration d'objets au sein d'un système distribué est une tâche très complexe. De nombreux problèmes émergent au fur et à mesure de la réalisation d'un tel mécanisme ce qui impose de nombreuses contraintes. Dans ce qui suit nous présentons les problèmes majeurs posés par la migration ainsi que les choix que nous avons faits pour les résoudre.

6.3.1 Migration d'objets : le problème

Un système de migration d'objets doit autoriser la mobilité d'objets d'un processus à un autre sans affecter le déroulement des opérations en cours. Ce système doit aussi assurer un certain niveau de transparence pendant la migration. Ceci veut dire que si une tâche était en exécution lors du déclenchement du mécanisme de migration, cette tâche doit continuer à fonctionner de la même manière que si la migration n'avait pas eu lieu.

Le mécanisme de migration d'un objet peut être plus ou moins compliqué selon la nature de l'objet à faire migrer. On peut identifier essentiellement deux types d'objets : objets actifs et objets passifs. Un objet peut être actif ou passif selon sa fonction et son utilisation par d'autres objets ou utilisateurs. Par exemple un objet de décor comme une chaise est un objet passif qui n'effectue aucun calcul particulier. La migration d'un objet passif n'est rien de plus que l'envoi du fichier qui le décrit. La migration d'un objet actif est par contre beaucoup plus compliquée. Un objet actif est une combinaison d'une instanciation d'un code en exécution et d'un état associé. Dans le cas d'un objet actif, le mécanisme qui assure la migration doit avoir la capacité de suspendre son exécution sur une machine et de la reprendre plus tard sur une autre machine. Cette opération doit se dérouler tout en conservant l'état de l'objet lors de sa suspension.

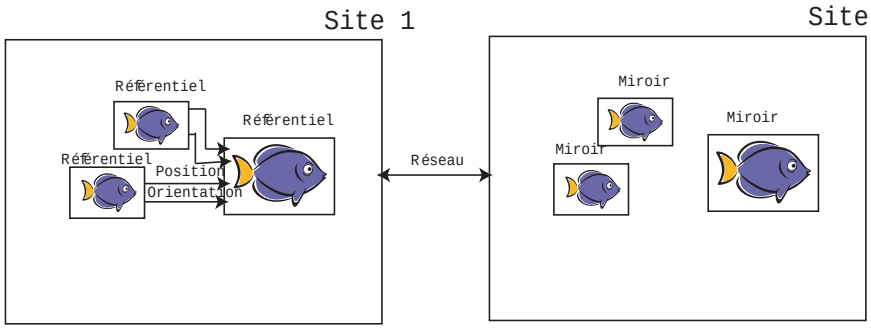

FIG. 6.2 – Exemple de communication entre les objets avant la migration du poisson guide

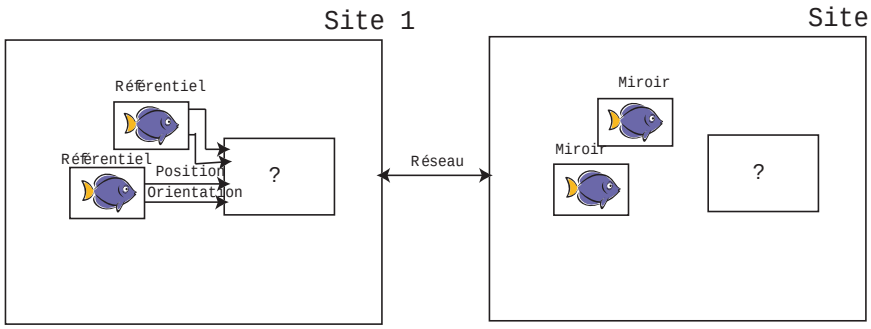

FIG. 6.3 – Communication entre les objets pendant la migration

Un objet actif peut être aussi en interaction avec d'autres objets. Cette interaction ne doit pas être brisée lors de la migration. Plusieurs formes d'interaction peuvent exister entre les objets d'un univers virtuel. La forme la plus simple d'interaction entre deux objets est la communication entre eux c'est-à-dire lorsqu'un

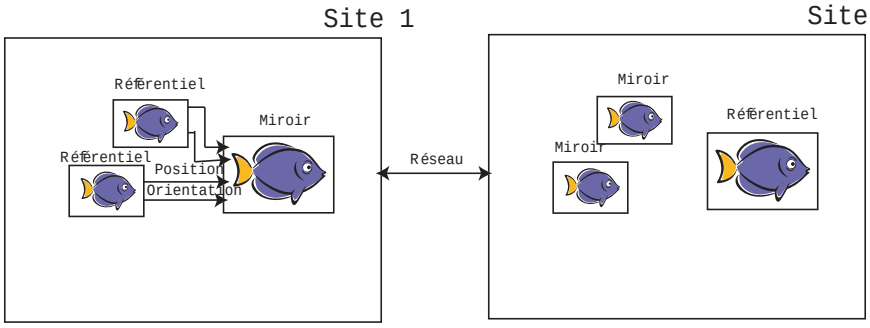

FIG. 6.4 – Communication entre les objets après la migration

objet lit les valeurs d'attributs d'un autre objet. Prenons l'exemple très simple schématisé par la figure 6.2. Dans cet exemple deux sites partagent un environnement virtuel distribué. Cet environnement simule une troupe de trois poissons. Le gros poisson est un poisson guide qui suit une trajectoire aléatoire calculée par l'unité de calcul qui lui est associée. Les deux autres suivent tout simplement la trajectoire prise par le poisson guide en lisant ses attributs de position et d'orientation. Si à un instant donné on décide de faire migrer le poisson guide du site 1 vers le site 2 (Fig. 6.3), cette migration ne doit pas affecter la trajectoire du reste de la troupe ni sa visibilité sur les deux sites (Fig. 6.4). Notons que dans le cas général les objets n'ont pas forcément de représentation géométrique, nous avons choisi ici ce cas très simple à titre d'exemple.

6.3.2 Transparence et portabilité

Un mécanisme de migration peut être conçu et implémenté sur plusieurs niveaux d'un support d'exécution d'environnements virtuels distribués comme on peut le voir sur la figure 6.5. Il peut par exemple intégrer la couche de communication du système. Notons qu'une telle couche de communication n'est pas forcément une entité élémentaire du système lui même. Cette couche peut être totalement indépendante du système qui l'utilise. En effet, plusieurs plates-formes de développement d'environnements virtuels utilisent des bibliothèques de communication qui sont totalement indépendantes de la conception du système. Par exemple, OpenMASK utilise la bibliothèque à passage de messages PVM. L'avantage d'un mécanisme de migration à ce niveau est la disponibilité de plusieurs bibliothèques de communication qui offrent un tel service. MPVM en est un exemple. Par contre nous ne recommandons pas le choix d'utiliser une bibliothèque de communication qui supporte la migration. L'inconvénient est que ce type de migration est extérieur à la plate-forme de développement d'environnements virtuels. Si on décide de remplacer la couche de communication par une

autre couche pour une raison quelconque, on perd par conséquent le mécanisme de migration.

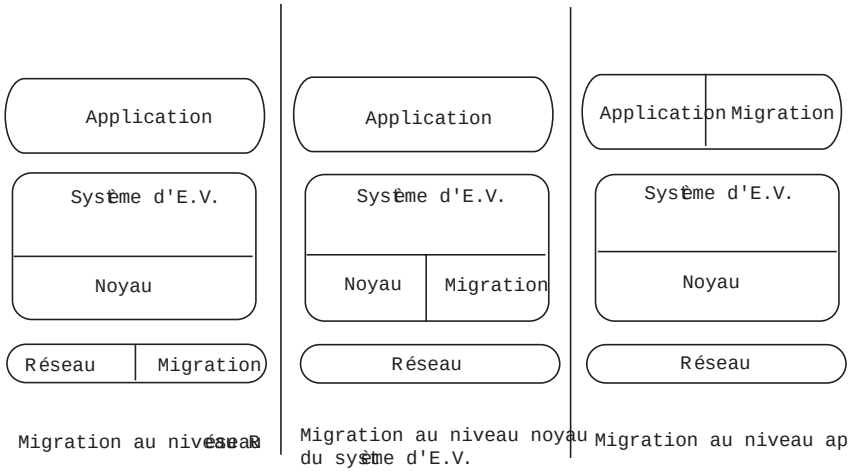

FIG. 6.5 – Les différents niveaux d'un mécanisme de migration

Le mécanisme de migration peut intégrer un niveau plus élevé que celui de la couche de communication. Il peut faire partie du noyau du support d'exécution d'environnements virtuels distribués. En intégrant le noyau, la migration devient un service offert par le système d'environnements virtuels et non pas par d'autres couches comme le cas évoqué dans le paragraphe précédent. Ceci est un avantage du point de vue portabilité : quelque soit la couche de communication utilisée, ce service reste valide. L'inconvénient du point de vue réalisation est peut être la difficulté de mise en œuvre.

Le dernier niveau où le mécanisme de migration peut surgir est le niveau applicatif. Là non plus ce mécanisme ne fait plus partie du système d'environnements virtuels mais plutôt partie d'une application. À ce niveau le service de migration n'est pas portable ni transparent car c'est à la charge du programmeur d'application de l'assurer.

Nous avons choisi le deuxième niveau pour intégrer le mécanisme de migration d'objets au sein d'une plate-forme de développement et d'exécution d'environnements virtuels. Nous voulons que ce service soit offert par le système lui même pour conserver sa portabilité et pour assurer sa transparence. Ce choix, bien que difficile à réaliser, possède des avantages que nous croyons primordiaux pour converger vers une plate-forme de réalité virtuelle "indépendante".

6.3.3 Les choix possibles pour la migration

Pour concevoir un mécanisme de migration d'objets qui réponde aux besoins évoqués dans la section 6.3.1, et qui puisse intégrer le noyau d'une plate-forme de

développement et d'exécution d'environnements virtuels nous avions deux choix : la re-création d'objet sur le site cible ou le changement d'état du réplica de l'objet sur le site cible. Dans ce qui suit nous présentons les avantages et les inconvénients de chaque méthode.

6.3.3.1 Re-création d'objets

La méthode de re-création d'objets consiste à re-créer l'objet à migrer sur le site cible. L'objet clone re-créé sur le site cible doit avoir les mêmes propriétés ainsi que le même état actuel que l'objet à migrer. Après la re-création, l'objet sur le site d'origine doit être détruit. Toutes les fonctionnalités de l'objet doivent être conservées après la terminaison de l'opération.

La re-création est une opération complexe qui nécessite beaucoup de précautions. D'abord il faut récupérer toutes les informations liées à l'objet à migrer. Il faut aussi dénombrer tous les objets qui sont liés ou connectés à l'objet qui migre. Ensuite il faut sérialiser ces informations et les faire transiter sur le réseau vers le site destination. Après la re-création de l'objet sur le site destination et la destruction de l'objet initial, il va falloir rebrancher tous les autres objets de la même manière qu'avant la destruction sans perdre aucune information. Pour résumer, l'opération se déroulera suivant les étapes suivantes :

1. Suspendre le calcul de l'objet
2. Sauvegarder tous les attributs de l'objet à migrer
3. Dénombrer les objets dépendants de l'objet à migrer et identifier les attributs dont chacun dépend
4. Débrancher les objets connectés
5. Envoyer toutes ces informations vers le site destination
6. Détruire le miroir de l'objet s'il existe sur le site destination
7. Re-créer l'objet sur le site destination
8. Détruire l'objet initial sur le site source
9. Créer un miroir de l'objet sur le site source si nécessaire
10. Rebrancher les autres objets sur le miroir
11. Reprendre le calcul de l'objet

La réalisation du mécanisme de migration d'objets suivant ces étapes est une tâche qui risque d'être coûteuse en temps d'exécution. Notons aussi que l'identification de tous les objets qui dépendent directement ou indirectement de l'objet à migrer ainsi que la sérialisation de toutes ces informations nécessitent des modifications sur la structure interne de l'objet lui même.

6.3.3.2 Métamorphose

Cette méthode utilise les infrastructures communes de la plupart des plates-formes de développement d'environnements virtuels distribués. Beaucoup d'architectures de distribution utilisent le modèle objet/fantôme que nous appelons aussi référentiel/miroir. Normalement c'est le référentiel qui détient l'unité de calcul de l'objet qui décide son comportement interne. Un miroir est une forme simplifiée de l'objet qui ne réalise aucun calcul. Un miroir se contente d'offrir la même interface externe que son référentiel à l'intention des autres objets virtuels. Les résultats des calculs faits par le référentiel sont transmis vers tous ses miroirs comme nous l'avons expliqué dans le chapitre précédent.

La méthode du changement d'état consiste donc à transformer un miroir en un référentiel et un référentiel en un miroir selon le besoin. Au lieu de transmettre toutes les informations et attributs de l'objet on ne transmet que les informations nécessaires pour attacher une unité de calcul à un miroir cible. Aucune destruction ne sera faite selon le principe de changement d'état. Les étapes à appliquer peuvent être résumées ainsi :

1. Suspendre le calcul de l'objet
2. Débrancher le miroir cible de son référentiel initial
3. Transformer le miroir cible en référentiel
4. Débrancher les autres miroirs du référentiel initial
5. Rebrancher ces miroirs au nouveau référentiel
6. Transformer le référentiel initial en miroir
7. Brancher ce miroir au nouveau référentiel
8. Reprendre le calcul de l'objet

Au cas où il n'y avait pas de miroirs sur le site cible, il faut en créer un pour le transformer en référentiel par la suite. Au cas où il n'y a plus besoin de miroir sur le site d'origine du référentiel, on y détruit le miroir.

L'avantage majeur de cette méthode est qu'aucun objet n'est détruit (sauf si on n'a plus besoin de l'objet) ce qui évite le débranchement et le rebranchement des objets dépendants de l'objet à migrer ainsi que leur identification.

6.3.3.3 Synthèse

Les deux méthodes de migration d'objets que nous venons de citer ont une certaine ressemblance. Dans la première il faut identifier tous les objets qui dépendent de l'objet à faire migrer, tandis que dans la seconde il faut identifier tous les miroirs de celui-ci afin de les orienter vers le nouveau site hébergeur. Par contre, la différence de base se résume ainsi :

- pour chaque migration, la première méthode propose de créer un objet, tandis qu'avec la deuxième méthode une création n'est envisagée que dans le cas d'une migration vers un site ne disposant pas de miroir
- pour chaque migration, la première méthode propose de détruire un objet, tandis qu'avec la deuxième méthode une destruction n'est envisagée que dans le cas d'une migration d'un objet sans avoir besoin de transformer l'ex-référentiel en miroir. Dans ce cas l'objet est détruit sur le site d'origine

Nous pensons que la deuxième méthode est plus avantageuse car elle est moins coûteuse en terme de création et destruction d'objets virtuels, c'ets donc cette seconde méthode que nous allons mettre en œuvre au sein de notre environnement.

6.4 Mise en œuvre de la migration d'objets

Nous avons implémenté le mécanisme de migration d'objets dans le noyau de la plate-forme OpenMASK selon les choix décrits dans les sections précédentes. Nous allons détailler dans ce qui suit notre méthode qui consiste à modifier l'état d'un objet au sein d'OpenMASK. Rappelons qu'OpenMASK réplique ses objets selon le principe des référentiels et des miroirs (5.4.1). Nous allons d'abord détailler les fonctions d'un référentiel et celles d'un miroir afin de mieux comprendre les rapports qu'ils entretiennent et comment nous allons pouvoir transformer un référentiel en miroir et inversement.

6.4.1 Les objets OpenMASK

Dans sa version distribuée, OpenMASK a deux notions d'objets : les référentiels et les miroirs. Les objets référentiels et les objets miroirs ont de nombreuses fonctionnalités en commun. Ils conservent cependant quelques différences de comportement. Le modèle référentiel/miroir est complètement transparent à l'utilisateur. En effet, l'objet simulé associé à un référentiel est presque identique à celui associé à un miroir sauf qu'il peut être initialisé différemment. Dans ce dernier cas l'utilisateur peut attacher à un référentiel un comportement qui lui est propre. Ce comportement est défini pendant la phase d'initialisation de l'objet référentiel. Par contre, ce qui fait la différence principale entre les deux, c'est le gestionnaire d'objet en question qui leur est couplé. Deux types de gestionnaires d'objets existent dans OpenMASK : le gestionnaire d'objet référentiel et le gestionnaire d'objet miroir. C'est le gestionnaire associé à l'objet qui contrôle sa vraie nature. Ainsi, nous pouvons dire qu'un référentiel est un mariage entre un objet simulé et un gestionnaire de référentiel. Nous pouvons aussi définir un miroir de la même manière, mariage d'un objet simulé et d'un gestionnaire de miroir (Fig. 6.6).

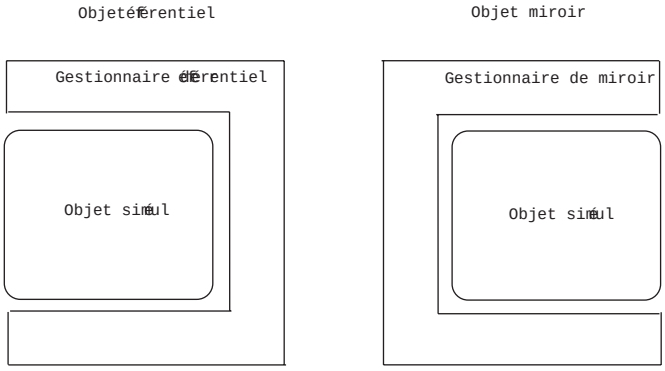

FIG. 6.6 – Référentiel et miroir

6.4.1.1 L'objet référentiel

Dans une simulation distribuée sur plusieurs sites, un objet simulé est assigné à un processus qui est à son tour assigné à un site. Ainsi à l'intérieur de chaque processus se trouvent un certain nombre d'objets qui y sont assignés et que nous appelons des référentiels. Le référentiel exécute le calcul interne de l'objet qui est responsable de son évolution et de son comportement. Le référentiel répond aux requêtes d'abonnement et désabonnement en provenance de ses miroirs et traite aussi les différents événements reçus par ces derniers. Nous pouvons dire qu'un référentiel est un objet "maître" qui possède l'unité de calcul, contrairement aux miroirs qui sont des objets sans "intelligence" (ils ne réalisent aucun calcul) qui ne font que refléter l'apparence externe de leurs référentiels.

6.4.1.2 L'objet miroir

Un objet miroir possède la même interface que son référentiel distant et se situe en local. La fonction d'un objet miroir est d'empaqueter les paramètres et les événements reçus pour envoi au référentiel distant qui ouvre et traite ces messages. Un miroir est créé localement dès que son référentiel est utilisé d'une manière ou d'une autre sur n'importe quel processus autre que celui qui détient le référentiel. Un miroir n'exécute aucun calcul interne. En effet, un miroir peut être vu comme une jonction entre l'objet référentiel distant et les objets locaux, permettant ainsi leurs interactions.

À la création d'un miroir sur un site donné, un canal de communication par flots de données est créé par le noyau OpenMASK entre le site détenant le miroir et celui du référentiel. Il demande ensuite l'abonnement pour pouvoir lire les attributs de sorties de son référentiel et les tenir à la disposition des référentiels situés sur son processus.

6.4.2 Métamorphose

Nous avons choisi de réaliser un mécanisme de migration d'objets par changement d'état. Ceci veut dire qu'un objet donné doit être capable de se métamorphoser pour changer de statut. Ainsi un miroir peut devenir référentiel et un référentiel peut devenir miroir.

Dans le contexte particulier d'OpenMASK, la réalisation de la migration par changement d'état peut être envisagée selon deux méthodes différentes.

6.4.2.1 Métamorphose par mutation interne

La première méthode consiste à re-concevoir la structure interne des référentiels et des miroirs de manière à ce qu'ils implémentent exactement la même interface. Ceci veut dire qu'un objet référentiel contiendra exactement les mêmes fonctionnalités qu'un objet miroir ou en d'autres termes il est référentiel et miroir à la fois. Ces fonctionnalités seront activées ou désactivées selon le comportement voulu de l'objet (référentiel ou miroir). Du côté de l'objet miroir, ce sera exactement la même chose (Fig. 6.7).

L'avantage de cette méthode est la mutation simple d'un objet référentiel vers un objet miroir. Pour une transformation d'un référentiel en un miroir par exemple, il suffit d'activer les méthodes propres au comportement d'un miroir au sein de l'objet même et désactiver les méthodes qui définissent le comportement d'un référentiel. L'inconvénient est la modification importante de la structure interne d'un objet simulé dans OpenMASK. Ceci contredit notre but de réaliser le mécanisme de migration en utilisant la limite de l'infrastructure existante d'OpenMASK pour ne pas trop changer le noyau ni modifier son principe de fonctionnement. Pour cette raison, nous avons décidé d'adopter une deuxième méthode.

6.4.2.2 Métamorphose par changement de nature

La deuxième méthode consiste à ne pas modifier le comportement d'un objet référentiel ni d'un objet miroir. Nous allons en fait remplacer le gestionnaire associé à un objet sans détruire l'objet lui même.

L'opération consiste donc à détruire le gestionnaire associé à un objet (référentiel par exemple), re-créer un autre gestionnaire (de miroir), et associer ce nouveau gestionnaire à l'objet sans modifier aucun de ses attributs.

Il faut aussi prendre en considération le cas où un utilisateur a décidé d'associer un comportement spécial à son objet référentiel (voir 6.4.1). Dans ce cas nous devons transmettre ce comportement et l'attacher au nouveau référentiel. Pour cela nous avons défini deux méthodes à redéfinir par l'utilisateur qui souhaite exprimer des aspects propres à un objet donné. Ces deux méthodes sont *emigrate()*

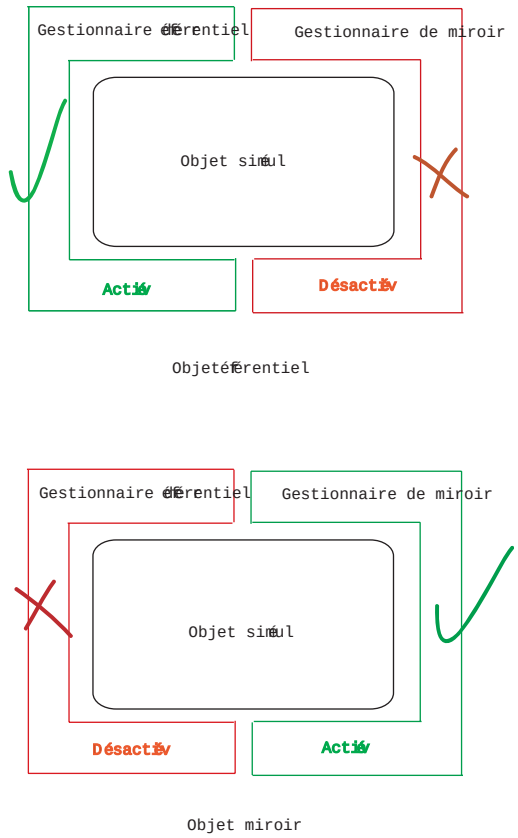

FIG. 6.7 – Métamorphose par mutation interne

et *immigrate()*. Dans ces méthodes, un utilisateur doit spécifier le comportement particulier à faire migrer avec l'objet d'un site à un autre.

Nous avons réalisé l'implémentation de cette méthode au sein du noyau OpenMASK. Les détails du fonctionnement du protocole de migration sont discutés dans la section suivante.

6.4.3 Protocole de migration

La migration d'un objet d'un site ou processus à un autre est déclenchée par l'envoi d'un événement spécial dans le noyau distribué d'OpenMASK. Cet événement comporte le nom de l'objet à faire migrer et le nom du processus destination. L'événement de migration sera reçu par tous les contrôleurs au prochain pas de simulation. À la réception de l'événement chaque contrôleur local a en effet un travail particulier à réaliser. Un contrôleur doit donc choisir parmi quatre cas possibles celui qui correspond à son état :

– **le contrôleur possède le référentiel de l'objet à faire migrer :** dans

ce cas le référentiel doit être transformé en miroir. Ceci veut dire que son gestionnaire de référentiel associé doit être détruit et remplacé par un gestionnaire de miroir. Une fois l'objet transformé en miroir il faut le brancher sur son nouveau référentiel.

- **le contrôleur est sur le processus destination et possède un miroir de l'objet à faire migrer :** ceci veut dire que nous sommes déjà sur le site cible, donc il faut transformer le miroir en référentiel. Inversement au cas précédent, c'est le gestionnaire de miroir qui est détruit et remplacé par un gestionnaire de référentiel.
- **le contrôleur est sur le processus destination et ne possède pas de miroir de l'objet à faire migrer :** dans ce cas il faut commencer par créer un miroir de l'objet sur ce processus en utilisant la méthode standard de création de miroir. Ensuite nous appliquons la procédure de transformation de miroir en référentiel comme indiquée dans le paragraphe précédent.
- **le contrôleur possède l'un des miroirs de l'objet à faire migrer :** dans ce dernier cas nous devons gérer la situation des autres miroirs de l'objet à migrer. Rappelons que ces miroirs sont branchés à leur référentiel par une connexion en flots de données. Il faut donc débrancher ces miroirs de leur référentiel qui se transforme en miroir puis les rebrancher sur leur nouveau référentiel.

Ces différentes étapes ne doivent pas être traitées dans n'importe quel ordre. Par exemple on ne peut pas déclencher le mécanisme de réorientation des miroirs vers leur nouveau référentiel si ce dernier n'a pas encore été métamorphosé en référentiel pour qu'il puisse traiter les demandes de branchement. En même temps il faut éviter de perdre du temps en exécutant toutes les étapes de migration en autant de pas de simulation.

Après quelques études, nous avons trouvé un compromis qui permet l'achèvement de la procédure de migration en deux étapes pendant deux pas de simulation. La première étape consiste à faire exécuter l'algorithme 3 par tous les contrôleurs. Cet algorithme consiste à transformer le miroir sur le processus destination en référentiel. En parallèle les autres miroirs se débranchent de leur référentiel actuel qui est en train de migrer sur un autre processus. L'étape suivante est l'application de l'algorithme 5. Ici, nous transformons le référentiel à migrer en miroir et nous le branchons sur son nouveau référentiel. En même temps nous nous occupons du branchement des autres miroirs sur les autres processus vers leur nouveau référentiel.

Algorithme 3 Algorithme exécuté par tous les contrôleurs à la réception d'un événement de migration

```
// nomProcessusDestination est le nom du processus destination vers lequel
un objet doit migrer
// nomObjet est le nom de l'objet à faire migrer
si nomProcessusAbritant (nomObjet) = nomProcessusDestination alors
   // cas trivial : cela veut dire que dans ce cas on ne fait rien car le processus
   est déjà sur le processus destination
sinon si contrôleur.nomDuProcessus = nomProcessusDestination alors
   // Donc c'est le contrôleur du processus qui détient le miroir qui se trans-
   formera en référentiel
   mir ← listeDesMiroirs.chercher (nomObjet) ;
   transformerEnRéférentiel (mir) ; // voir algo. 4
   envoyerEvénement (migrationEtape2) ; // voir algo. 5
sinon si listeDesMiroirs.chercher (nomObjet) ≠ φ alors
   // Dans ce cas c'est un contrôleur qui a un des miroirs de l'objet à faire
   migrer
   // donc il faut désabonner ce miroir de leur ex-référentiel
   annulerAbonnementAuRéférentiel (nomObjet) ;
sinon si listeDesRéférentiels.chercher (nomObjet) ≠ φ alors
   // Dans ce cas c'est un contrôleur qui a le référentiel de l'objet à faire migrer
   // il faut sauvegarder ses valeurs internes avant de le transformer en miroir
   dans
   // l'étape suivante
   sauvegarderValeursInternes (nomObjet) ;
finsi
```

Algorithme 4 Algorithme de transformation d'un miroir en référentiel

```
// nomProcessusDestination est le nom du processus destination vers lequel
un objet doit migrer
// nomObjet est le nom de l'objet à faire migrer
gestionnaireMiroir ← récupérerLeGestionnaireMiroir (nomObjet) ;
gestionnaireMiroir.annulerAbonnementVersRéférentiel () ;
listeDesMiroirs.ôter (nomObjet) ;
delete gestionnaireMiroir ;
nouveauRéférentiel ← new gestionnaireRéférentiel (nomObjet) ;
listeDesRéférentiels.ajouter (nouveauRéférentiel) ;
activerRéférentiel (nouveauRéférentiel) ;
```

Algorithme 5 Algorithme de l'étape 2 de la migration

```
// nomProcessusDestination est le nom du processus destination vers lequel
un objet doit migrer
// nomObjet est le nom de l'objet à faire migrer
si nomProcessusActuel ≠ nomProcessusDestination alors
  si nomProcessusActuel ≠ nomProcessusAbritantExRéférentiel alors
    // Donc c'est un processus qui a un des miroirs du référentiel à migrer
    // Il faut brancher ce miroir à son nouveau référentiel
    brancherMiroirVersProcessus (nomProcessusDestination) ;
  sinon
    // donc c'est un processus qui a l'ex-référentiel
    // il faut le transformer en miroir puis le rebrancher à son nouveau réfé-
    rentiel
    référentiel ← listeDesRéférentiels.chercher (nomObjet) ;
    transformerEnMiroir (référentiel) ; // voir algo. 6
    brancherMiroirVersProcessus (nomProcessusDestination) ;
  finsi
finsi
```

Algorithme 6 Algorithme de transformation d'un référentiel en miroir

```
// nomProcessusDestination est le nom du processus destination vers lequel
un objet doit migrer
// nomObjet est le nom de l'objet à faire migrer
gestionnaireRéférentiel ← recupérerLeGestionnaireRéférentiel (nomObjet) ;
listeDesRéférentiels.ôter (nomObjet) ;
delete gestionnaireRéférentiel ;
nouveauMiroir ← new gestionnaireMiroir (nomObjet) ;
listeDesMiroirs.ajouter (nouveauMiroir) ;
brancherMiroirVersProcessus (nomProcessusDestination) ;
```

6.5 Évaluation du mécanisme de migration

Rappelons que le temps nécessaire pour un objet de migrer d'un site A vers un site B est de l'ordre de deux pas de simulation OpenMASK. Ceci est dû à deux raisons :

1. l'utilisation d'événements pour déclencher la migration : avec OpenMASK, un événement envoyé pendant un pas de simulation P est traité au pas $P+1$.
2. la migration se déroule en deux phases : il y a des procédures qui doivent strictement être exécutées avant d'autres, d'où le mécanisme de migration en deux phases.

6.5.1 L'application de test

Pour étudier le comportement du mécanisme de migration d'objets et estimer sa performance, nous avons fait plusieurs expérimentations. Ces expérimentations ont pour but essentiel de mesurer l'efficacité du mécanisme de migration en terme de perception visuelle, c'est-à-dire pour répondre à la question suivante : est-ce que la migration d'un objet se déroule d'une manière entièrement transparente pour l'utilisateur ? Ou bien ce dernier peut-il percevoir quelques indices indiquant qu'un objet est en train de migrer ?

Pour cette étude, nous avons réalisé une application simple comportant des objets virtuels en mouvement continu. Ces objets se suivent les uns des autres en parcourant une trajectoire. La figure 6.8 nous montre l'objet 1 suivant une trajectoire, l'objet 2 suit l'objet 1, l'objet 3 suit l'objet 2 et ainsi de suite.

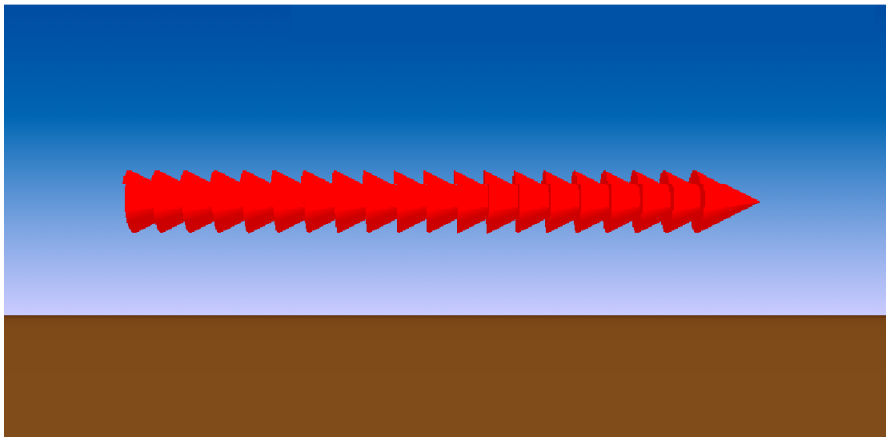

FIG. 6.8 – Exemple de migration d'objets en mouvement

Cette application est donc un univers virtuel partagé entre deux processus A et B et distribué sur deux sites interconnecté par un réseau LAN. Nous avons placé 100 objets en mouvement dans la scène dont les référentiels résident sur le processus A. Les expérimentations consistent à faire migrer les objets du processus A vers le processus B et à étudier l'impact de la migration en terme de perception visuelle. En effet, l'impact est étudié de deux points de vues : du point de vue humain (perception de l'impact par l'utilisateur) et du point de vue système en mesurant le temps nécessaire pour terminer l'opération de migration. Nous avons fait des mesures sur des tests qui consistent à faire migrer un seul objet au début, puis à augmenter le nombre d'objets à faire migrer au fur et à mesure jusqu'enfin faire migrer tous les objets vers le processus B.

6.5.2 Les résultats

Les résultats sont visualisés par les courbes données dans la figure 6.9. L'axe des abscisses montre le nombre d'objets à faire migrer du processus A résidant sur le premier site vers le processus B résidant sur le deuxième site. L'axe des ordonnés montre le temps nécessaire en millisecondes pour achever la migration sur les deux processus simultanément.

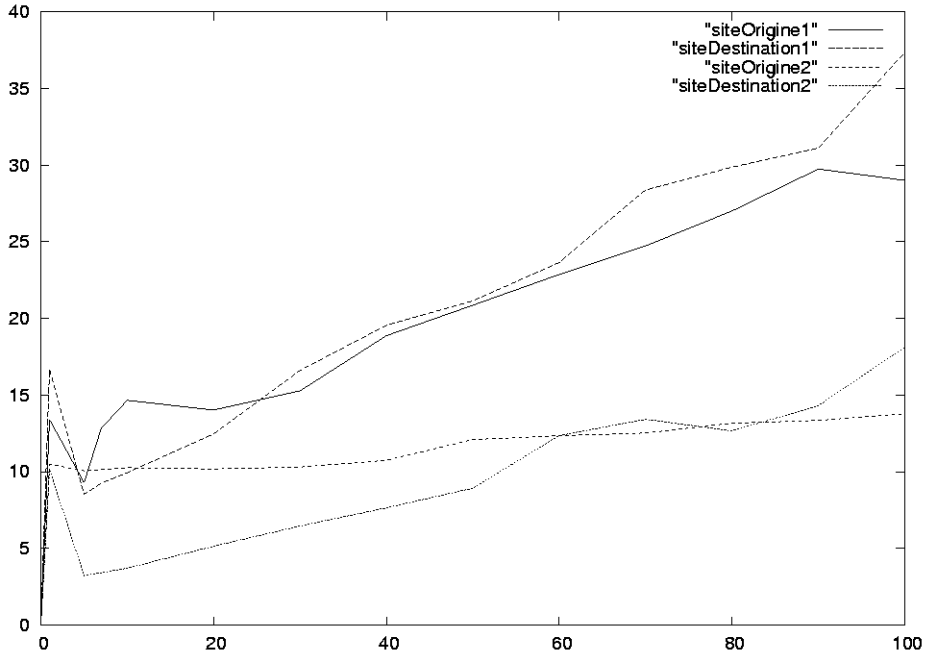

FIG. 6.9 – Temps de migration de N objets en millisecondes

Le premier test s'est déroulé entre deux machines Pentium III, 1 GHz dis-

posant de 768 M de mémoire vive. Du point de vue visuel, on observe que l'application se fige un très court instant au moment de la migration, sur chacun des processus, quelque soit le nombre d'objets qui migrent. Cette impression est confirmée par les mesures effectuées puisque le coût minimal de la migration semble se situer entre 10 et 15 millisecondes, et qu'il augmente ensuite faiblement en fonction du nombre d'objets à faire migrer. En effet le temps nécessaire à la migration de n objets semble pouvoir être évalué à l'aide d'une fonction affine du type : tempsDeMigration = 10 ms + n * 0.25 ms.

Le deuxième test s'est déroulé entre deux machines Pentium IV 3GHz disposant de 1 G de mémoire vive. Nous pouvons remarquer sur la figure 6.9 une amélioration nette du temps de migration (les courbes du bas). Du point de vue visuel, la migration n'est quasiment plus perceptible, et dans ce deuxième test la formule estimant le temps de migration de n objets est plutôt du type : tempsDeMigration = 10 ms + n * 0.05 ms. Cette amélioration (diminution de la pente de la courbe) est dûe à la puissance plus importante des machines utilisées pour ce test. Le coût minimal reste cependant le même, ceci est peut-être dû à l'utilisation de PVM, il faudra faire d'autres expérimentations pour s'en assurer.

6.5.3 Changement d'état vs re-création

Pour faire une comparaison entre la technique de migration par changement d'état et celle par destruction et re-création d'objet, nous avons fait l'expérience suivante : nous avons comparé la dégradation en terme de perception visuelle d'une application consistant à faire migrer un certain nombre d'objet d'un site à un autre avec la même application créant localement d'une manière dynamique le même nombre d'objet. Dans le dernier cas les performances sont nettement moins bonnes surtout si le nombre d'objets est important.

6.6 Synthèse et perspectives

La migration d'objets au sein d'un environnement virtuel distribué offre de nombreuses perspectives en terme de dynamicité. Sans la migration un système est statique c'est-à-dire qu'un objet assigné à un processus au début d'une simulation donnée est condamné à rester attaché à ce même processus jusqu'à la fin de la simulation. Une perte de communication avec ce processus signifie la perte de communication avec les objets qui y résident. De plus, une dynamicité au niveau ajout et suppression de processus à une simulation en cours n'est pas réalisable sans un mécanisme de migration qui assure la récupération des objets résidant sur le processus qui désire quitter la simulation.

Nous avons présenté dans ce chapitre notre proposition, conception et réalisation de l'infrastructure d'un mécanisme de migration d'objets au sein d'une plate-

forme de développement d'environnements virtuels distribués. La réalisation a été faite dans le noyau du système OpenMASK. Rappelons que la migration d'objets est un service qui a été réalisé dans très peu de telles plates-formes. Malheureusement nous n'avons pas pu tirer profit du savoir faire de quelques systèmes qui offrent ce service (notamment AVIARY et WAVES) par manque d'informations rendues publiques sur ce sujet. Nous avons donc réalisé notre propre contribution indépendamment de toute implémentation qui pourrait exister.

Maintenant que ce service est fonctionnel, nous pouvons offrir facilement beaucoup d'autres services et d'extensions en se basant sur la migration. Par exemple il serait possible de réaliser un mécanisme d'équilibrage de charges qui utiliserait la migration d'objets à partir d'un site chargé vers un autre moins chargé. L'extension la plus directe à nos réalisations dans ce chapitre serait peut être l'implémentation d'un algorithme qui automatiserait la migration d'un objet d'un processus à un autre selon des critères choisis par l'utilisateur. Actuellement, c'est à la charge de l'utilisateur de choisir l'objet à migrer et le site destination. Plusieurs scénarios pourraient être imaginés, on pourrait par exemple choisir de faire migrer automatiquement un objet vers le site de l'utilisateur qui interagit le plus avec.

L'un des services basés sur le mécanisme de migration d'objets est l'ajout et la suppression dynamique de processus et de sites à une simulation en cours. Nous avons réalisé ce service et nous le présentons dans le chapitre suivant.

Chapitre 7

Gestion dynamique des processus

7.1 Introduction

L'ajout et la suppression dynamique d'un processus ou d'un site rendent les environnements virtuels distribués beaucoup plus dynamiques. Dans le schéma statique, on fixe le nombre de processus participant à une simulation dès le départ. Ce nombre ne peut pas être modifié plus tard pendant la simulation sans arrêter cette dernière et la relancer de nouveau.

La possibilité de rajouter et de supprimer dynamiquement un processus à une simulation en cours sans la perturber a beaucoup d'avantages. Cela permet à un utilisateur de rejoindre un groupe d'utilisateurs sans arrêter ou relancer l'application en cours. Cela permet aussi à un utilisateur de quitter un environnement virtuel partagé sans obliger tous les autres utilisateurs à le quitter eux aussi. L'ajout dynamique d'un processus ou d'un site peut être également très utile dans le cas où il y a besoin de rajouter des ressources de calcul au cours d'une simulation.

La dynamicité est célèbre dans beaucoup de jeux vidéos temps réel sur internet, dans certains systèmes d'exploitation distribués et aussi dans quelques plates-formes de développement d'environnements virtuels distribués. Nous avons étudié et réalisé la possibilité d'ajouter et de supprimer un processus tandis qu'une simulation est en cours. Notre conception profite du mécanisme de migration d'objets présenté dans le chapitre précédent pour rendre la plate-forme OpenMASK dynamique. En effet la migration permet de garder la persistance des objets après la déconnexion définitive d'un site en migrant ses objets référentiels vers d'autres sites.

Nous allons présenter dans ce chapitre notre conception d'un système d'environnements virtuels dynamique, ainsi que sa réalisation et son implémentation au sein du noyau d'OpenMASK.

7.2 Pourquoi la dynamicité des sites ?

La dynamicité dans les environnements virtuels distribués présente certaines différences par rapport à ce que l'on peut trouver dans les systèmes d'exploitation distribués. Le but de l'ajout dynamique d'un site dans un système d'exploitation distribué est avant tout d'exploiter les ressources situées sur ce site. Ces ressources peuvent être mémoire, processeur, ou autres. L'élargissement du calcul vers d'autres sites peut être motivé par la montée croissante en terme de besoin de ressources pendant une opération de calcul intensive par exemple. Le motif peut aussi être la reconfiguration des ressources suite à la perte d'une machine à cause d'une défaillance quelconque. Dans ce cas il faut remplacer la machine défaillante par une autre machine pour ne pas perturber le calcul. Donc, pour résumer, pour un système d'exploitation distribué, une machine est vue comme une ressource de calcul. Dans le cas d'un environnement virtuel distribué, ceci reste vrai mais le plus intéressant est l'ajout d'un utilisateur en plus de sa machine. En rajoutant un site à un environnement virtuel il faut parfois rajouter aussi un affichage comme par exemple dans le cas d'un jeu vidéo multi-joueurs. L'affichage est la fenêtre sur l'environnement virtuel qui est primordiale pour un utilisateur. Un utilisateur doit pouvoir se connecter à une simulation ou pouvoir quitter une simulation à n'importe quel moment.

7.3 Conception d'un environnement virtuel dynamique

Pour qu'un environnement virtuel distribué puisse supporter la dynamicité il faut que le noyau de son support d'exécution sache gérer de nouvelles situations liées à l'arrivée et au départ d'un processus après que la simulation soit initialisée. Il doit pouvoir répondre à des demandes de rejoindre ou de quitter l'environnement virtuel partagé. La conception d'un tel mécanisme dynamique dépend largement de l'architecture du système distribué qui veut l'intégrer. Dans la famille des architectures centralisées, c'est le serveur central qui a la charge de répondre à des requêtes d'ajout ou de suppression de sites. Dans celle des architectures distribuées il n'y a pas de machine centrale. Chaque machine gère une partie du monde virtuel. Dans ce cas il faut choisir en général entre deux scénarios :

1. assigner un processus particulier comme étant le serveur du mécanisme de l'ajout et de la suppression. Dans ce cas, toutes les requêtes d'ajouts ou de suppressions de nouveaux sites doivent être traitées par ce serveur particulier.

2. garder le schéma distribué et autoriser les contrôleurs de tous les sites à répondre à des telles requêtes, ce qui peut être très complexe à gérer.

Dans le contexte de la plate-forme OpenMASK nous avons choisi d'adopter un scénario mixte. Rappelons qu'OpenMASK suit le schéma distribué. La couche de distribution utilisée par la version distribuée d'OpenMASK est PVM. Selon PVM un seul processus est capable de créer ou détruire d'autres processus participant à une même simulation. Ce processus "maître" est le premier processus créé par la machine virtuelle PVM lors de l'initialisation d'une simulation. Par exemple, prenons le cas d'une simulation qui comporte trois processus. Le premier créé joue le rôle d'un maître en créant les deux autres. Son rôle de maître se limite à la création et la suppression d'autres processus. En dehors de cela il n'a aucune particularité par rapport aux autres processus.

En tenant compte de cet aspect de la distribution selon PVM, les premiers concepteurs d'OpenMASK ont fait le choix suivant : pour une simulation comportant n participants, $n+1$ processus vont être créés réellement. Le premier processus joue le rôle du maître pour créer les n processus de la simulation. Après avoir créé tous les processus sur leurs sites associés, le processus maître se transforme en serveur de noms.

Nous devions alors concevoir un mécanisme permettant l'ajout et la suppression dynamique de sites en prenant en considération les contraintes exigées par l'architecture du noyau OpenMASK d'un part et la couche de distribution PVM d'autre part. D'où le choix d'un schéma mixte qui donne à l'utilisateur l'illusion que n'importe quel site peut rajouter dynamiquement un autre site, bien qu'en vérité ce n'est que le processus maître qui assure ce service. De cette manière nous associons un nouveau service au processus maître. C'est donc le contrôleur de ce processus qui va gérer la création et la suppression dynamique d'un processus à une simulation en cours.

7.3.1 Ajout dynamique d'un processus

Le premier service que nous avons réalisé est l'ajout dynamique d'un processus à une simulation en cours comportant plusieurs processus. Nous permettons un ajout d'un nouveau processus via deux méthodes différentes pour répondre à deux cas typiques. Le premier cas est celui du rajout d'un processus suite à une décision prise par l'un des contrôleurs des processus déjà existants dans la simulation. Nous appelons ce cas "ajout par extension" des sites déjà existants. Le deuxième cas est celui de la demande d'utilisateur externe d'être ajouté à une simulation en cours. Ce cas est appelé "ajout à la demande".

7.3.1.1 Ajout par extension

N'importe quel utilisateur déjà connecté à une simulation peut ajouter un ou des processus à la simulation en cours. Il suffit de le demander au contrôleur local en précisant le site destination et un nom associé au nouveau processus. Le reste de l'opération d'ajout dynamique est transparent à l'utilisateur. En effet ce n'est pas le contrôleur local qui crée le nouveau processus car, comme nous l'avons expliqué dans les sections précédentes un seul contrôleur a le droit de le faire : ce contrôleur est celui associé au premier processus créé. Les contrôleurs locaux acheminent donc la demande d'un ajout vers le processus maître. Ce dernier crée le nouveau processus sur le site spécifié dans la demande. Ainsi, le nouveau site est considéré comme faisant partie du groupe et il peut recevoir toutes les mises-à-jour des objets du monde de la façon habituelle.

7.3.1.2 Ajout à la demande

Contrairement à l'ajout par extension où le demandeur est un site qui est déjà connecté à la simulation, l'ajout à la demande concerne des sites externes qui demandent leur première connexion. Un site demandeur doit faire parvenir sa demande au processus maître. Ceci peut se faire de deux façons. La première méthode consiste à faire communiquer le site demandeur directement avec le processus maître pour lui transmettre la requête. Cette liaison peut se faire à travers une communication par socket sur un port prédéfini. La deuxième méthode consiste à créer un objet de simulation qui a pour objectif d'écouter sur un port prédéfini et transmettre les requêtes de connexions au processus maître. Après la réception de la requête par le processus maître, la suite de la procédure de l'ajout dynamique est exactement identique à celle de l'ajout par extension.

7.3.2 Suppression dynamique d'un processus

La suppression dynamique consiste à permettre à un site de se déconnecter sans perturber les autres sites participant à la simulation. Nous ne traitons pas le cas d'une déconnexion suite à une défaillance d'un site. Dans cette étude nous sommes concerné par une déconnexion à la demande comme par exemple un joueur d'un jeu multi-joueurs qui veut quitter l'environnement. Pour ne pas perturber tout l'environnement il faut pouvoir récupérer tous les objets référentiels qui existent sur le site sortant, ce qui est possible grâce au mécanisme de migration présenté dans le chapitre précédent. Nous pouvons résumer la démarche de suppression dynamique d'un site aux étapes suivantes :

- détecter tous les référentiels existant sur le site sortant,
- choisir un ou plusieurs sites pour recevoir ces référentiels,
- faire migrer tous les référentiels vers le ou les sites choisis,

– supprimer le processus du site sortant de l'ensemble des processus de l'environnement virtuel.

L'idéal serait de pouvoir répartir les objets sur les sites les moins chargés mais nous ne disposons pas encore d'un mécanisme de répartition de charges automatique. Nous pouvons aussi imaginer plusieurs scénarios de distribution d'objets d'un site après sa déconnexion. On peut par exemple associer les objets sur les sites selon l'intérêt des utilisateurs d'avoir localement tel ou tel objet. On peut aussi détruire les objets qui ne sont pas utiles pour les utilisateurs restants ou encore dont la nature ne se prête pas à la migration, comme l'avatar de l'utilisateur qui a quitté la simulation.

Nous n'avons pas abordé l'étude des différentes possibilités de distribuer les objets d'un site sortant. Le plus facile serait de distribuer les objets sur tous les sites d'une manière équitable. Nous avons adopté cette dernière méthode relativement simple pour l'implémentation actuelle au sein d'OpenMASK.

7.4 Mise en œuvre de la dynamicité

Nous avons implémenté le mécanisme qui permet à un site de rejoindre ou quitter une simulation en cours dans le noyau distribué d'OpenMASK. Nous allons décrire les algorithmes qui ont permis la réalisation de ce mécanisme mais d'abord analysons la manière dont OpenMASK procède pour la création d'une simulation d'une manière statique. L'algorithme de création statique d'une simulation composée de plusieurs processus est antérieur à mes travaux de thèse. Dans cette version, le nombre de processus est fixé statiquement dans un fichier de configuration fourni par l'utilisateur au moment de l'initialisation de l'application. Le fichier de configuration doit être le même sur tous les sites participant à une même simulation.

La première étape d'exécution d'une application distribuée au dessus d'OpenMASK est la création et l'initialisation des contrôleurs. Pendant la phase d'initialisation, chaque contrôleur doit appliquer une des deux procédures suivantes. Si le contrôleur est le premier créé, il doit dans ce cas administrer la création de la simulation en jouant le rôle du maître comme nous avons expliqué dans les sections précédentes. Ce contrôleur maître fait alors appel à l'algorithme 7. Tous les autres contrôleurs doivent appliquer la deuxième procédure qui consiste à joindre la simulation créée par le contrôleur maître. Dans ce dernier cas l'algorithme 8 est appelé.

7.4.1 Ajout dynamique d'un site

Pour ajouter dynamiquement un nouveau site à une simulation en cours il faut tout d'abord en faire la demande au contrôleur serveur. Nous avons modifié

Algorithme 7 Algorithme de création d'une simulation distribuée

nomDuSiteLocal = siteMaître ;
pour tout contrôleur ∈ listeContrôleurs **faire**
  ajouter le site associé à la simulation ;
fin pour
pour tout contrôleur ∈ listeContrôleurs **faire**
  créer un processus associé ;
  créer un lien de communication avec ce processus ;
  envoyer le nom du processus au contrôleur associé ;
fin pour
créer une table de correspondance (ID, infoProcessus) ;
pour tout contrôleur ∈ listeContrôleur **faire**
  envoyer la table de correspondance ;
fin pour
Mode serveur activé

Algorithme 8 Algorithme de connexion à une simulation distribuée dans le cas statique

se connecter au contrôleur maître ;
joindre le groupe de processus "esclaves" ;
attendre la réception du nom du processus courant (envoyé par le serveur) ;
attendre la réception de la table de correspondance (envoyée par le serveur) ;
pour tout contrôleur ∈ listeContrôleur **faire**
  $nombreDeSitesEsclave \leftarrow nombreDeSitesEsclave + 1$;
  créer un canal de communication vers tous les autres contrôleurs ;
fin pour
synchroniser tous les processus

le comportement du serveur de façon à lui permettre de traiter une telle demande. À la réception d'une requête d'ajout d'un nouveau site, le serveur fait appel à l'algorithme 9. Cet algorithme assure la création d'un nouveau contrôleur qui sera associé à un nouveau processus créé sur le site précisé par la requête. Une fois la création d'un nouveau processus terminée, il faut notifier les différents contrôleurs de cette nouvelle extension. Chaque contrôleur applique alors l'algorithme 10.

Algorithme 9 Algorithme d'ajout dynamique d'un site à une simulation en cours

initialiser nouveau contrôleur ;
ajouter le site associé à la simulation ;
créer un processus associé ;
créer un lien de communication avec ce processus ;
envoyer le nom du processus au contrôleur associé ;
créer une table de correspondance (ID, infoProcessus) ;
envoyer la table de correspondance au nouveau contrôleur ;
envoyer un message de notification d'ajout d'un site à tous les autres contrôleurs ;

Algorithme 10 Algorithme de mises-à-jour des contrôleurs suite à l'ajout d'un site

listeContrôleur.ajouter (nouveauContrôleur) ;
$nombreDeSitesEsclave \leftarrow nombreDeSitesEsclave + 1$;
créer un canal de communication vers nouveauContrôleur ;
envoyer un message de succès d'initialisation ;
synchroniser tous les processus ;

7.4.2 Suppression dynamique d'un site

Pour quitter une simulation, le contrôleur local du processus désirant quitter envoie une requête de départ au contrôleur maître. À la réception de la requête, le contrôleur maître doit choisir un ou plusieurs sites pour héberger les objets référentiels du site partant. Une fois les sites hébergeurs repérés, le contrôleur maître déclenche le mécanisme de migration de tous ces objets référentiels vers les sites choisis. Après la terminaison de ces migrations, le maître déconnecte le site partant et envoie une notification vers les sites restants (Algo. 11). Ces derniers doivent mettre à jour le nombre de sites restants ; la liste des processus participant à la simulation, et enfin détruire le canal de communication avec le site partant (Algo. 12).

Algorithme 11 Algorithme de la suppression dynamique d'un site d'une simulation en cours

repérer tous les référentiels sur le site partant ;
migrer tous les référentiels vers d'autres sites ;
déconnecter le site partant de la simulation ;
détruire le processus associé ;
détruire le lien de communication avec ce processus ;
mettre à jour la table de correspondance (ID, infoProcessus) ;
envoyer un message de notification de suppression d'un site à tous les autres contrôleurs ;

Algorithme 12 Algorithme de mises-à-jour des contrôleurs de la suppression d'un site

listeContrôleur.supprimer (contrôleurPartant) ;
$nombreDeSitesEsclave \leftarrow nombreDeSitesEsclave - 1$;
Supprimer le canal de communication vers contrôleurPartant ;
synchroniser tous les processus ;

7.5 Synthèse et perspectives

Ce chapitre a présenté les modifications de comportement du contrôleur maître OpenMASK qui ont permis l'ajout et la suppression dynamique de site au cours d'une simulation distribuée en cours. Ce mécanisme profite directement du mécanisme de migration d'objets pour assurer la conservation des objets du site qui désire quitter.

Il serait intéressant de définir et étudier un algorithme de répartition d'objets sur les différents sites avant de déconnecter un site donné. Dans la version actuelle nous choisissons les sites hébergeurs au hasard sans tenir compte de leur charge.

Chapitre 8

Métaphores d'interactions pour la prise de conscience des problèmes réseau

8.1 Introduction

Dans les chapitres précédents nous avons évoqué l'influence de la couche réseau sur le comportement du noyau du support d'exécution d'environnements virtuels distribués. Les solutions proposées, étudiées et réalisées concernent les couches basses du support d'exécution d'environnements virtuels distribués et ne prennent pas en compte la dimension humaine. Seule la dimension technologique a été prise en compte. Or, les environnements virtuels distribués sont destinés à être utilisés par des utilisateurs humains et pas seulement par des entités numériques. La dimension humaine impose d'autres contraintes que celles imposées par les différentes couches formant une plate-forme logicielle de réalité virtuelle. Dans ce chapitre nous allons prendre en considération la manière dont un utilisateur interagit avec l'environnement virtuel. Nous étudions la façon dont un problème réseau pourrait dégrader la qualité de la coopération au sein d'un environnement virtuel. En effet, un tel problème peut interrompre l'interaction d'un utilisateur avec les objets brisant ainsi la coopération si l'interaction avait lieu pour réaliser une tâche coopérative. Ce problème peut aussi perturber l'utilisateur du fait de la perte le contrôle des objets qu'il peut entrainer si cette perte est accompagnée d'une mauvaise compréhension de ce qui se passe dans l'environnement virtuel.

Dans ce chapitre nous allons proposer de nouvelles métaphores graphiques/visuelles permettant d'informer les utilisateurs de la présence de problèmes réseau aussitôt que ces problèmes surgissent. L'objectif est de rendre l'utilisateur conscient qu'une partie des objets de l'environnement virtuel ne sont peut être pas à jour à cause d'un fort délai ou une déconnexion par exemple. Ces méta-

phores visuelles ont pour but d'attirer l'attention de l'utilisateur sur le fait qu'un problème réseau est survenu et que par conséquent quelques objets ne sont plus accessibles. Un tel mécanisme doit permettre aux utilisateurs un meilleur niveau de compréhension de ce qui se passe lors d'une déconnexion.

Dans les sections suivantes nous détaillons les effets d'un problème réseau sur la coopération dans un environnement virtuel distribué ainsi que l'avantage de rendre un utilisateur conscient de l'existence d'un tel problème. Ensuite nous présentons notre mécanisme qui permet de rendre l'utilisateur conscient de l'existence d'un problème à travers des métaphores visuelles. Nous terminons par la présentation et l'étude de ces métaphores.

8.2 Pourquoi la compréhension de l'univers virtuel ?

Pour un utilisateur, interagir avec un objet virtuel signifie pouvoir manipuler et contrôler cet objet au sein d'un environnement virtuel. Comme nous l'avons expliqué dans le chapitre 4 un objet peut être contrôlé soit par un seul utilisateur exclusivement, soit par plusieurs utilisateurs d'une manière collaborative. Les travaux classiques de ce domaine se focalisent sur la manière de rendre les utilisateurs conscients des interactions d'autres utilisateurs avec les objets du monde virtuel [GG98, FBHH99, FGV+00]. Ces travaux considèrent que la transmission au niveau réseau est fiable et ne se préoccupent pas du problème du délai. Par conséquent, rare sont les métaphores qui permettent de rendre un utilisateur conscient de l'existence d'une incohérence probable suite à un fort délai ou une déconnexion. Notons que suite à un problème réseau tel que le délai ou la déconnexion, on peut perdre le contrôle d'un ou de plusieurs objets virtuels. On peut aussi ne plus percevoir les interactions des autres utilisateurs distants ce qui fait croître l'incohérence entre les différents points de vue du monde virtuel.

Dans ce chapitre nous nous intéressons aux interactions coopératives en environnements virtuels distribués et partagés par plusieurs participants. Ce domaine se base sur la faible latence offerte par les réseaux haut débit. Un problème réseau peut causer des dommages aux interactions distantes qu'un utilisateur est en train de réaliser. Nous avons remarqué dans l'étude faite dans le chapitre 4 qu'aucune solution proposée ne résoud ce problème d'une manière globale. Nous avons donc décidé de traiter ce problème tout en admettant que nous ne pouvons pas l'éviter entièrement. Le principe de notre méthode est donc de fournir aux utilisateurs la conscience de l'existence d'un problème réseau lors de son occurence. De cette manière les utilisateurs pourront comprendre la nature du problème et par la suite adapter momentanément leurs comportements vis-à-vis de ce problème.

8.2.1 Effets du délai sur l'interaction et la coopération

En environnement virtuel distribué, la qualité de l'interaction avec un objet distant va être dépendante de la qualité de service offerte par le réseau, et plus particulièrement de la latence du réseau. Une faible latence permet des temps de réaction très courts et rend une interaction distante presque aussi fluide qu'une interaction locale. Par contre, si la latence augmente ou fluctue au cours d'une interaction distante, l'objet en interaction risque de réagir par saccades, ce qui dégrade la qualité de l'interaction.

En terme d'interactivité d'un objet, les interactions d'un utilisateur sur un site isolé par une déconnexion ne sont plus perçues par les autres sites distants. Ce même utilisateur ne perçoit plus à son tour les interactions des autres utilisateurs. Cela provoque une rupture de coopération entre les différents utilisateurs.

Si un objet virtuel suit un schéma d'évolution linéaire, nous pouvons dans ce cas atténuer l'effet d'un problème réseau en utilisant un mécanisme de prédiction suivant la méthode du *dead-reckoning*. Dans le cas d'une évolution complexe ou totalement imprévisible comme l'interaction d'un utilisateur, aucun mécanisme de prédiction n'est capable de résoudre même partiellement le problème. Les interactions d'un utilisateur dépendent de son humeur et de ses compétences, ce qui n'est régi par aucun algorithme connu de nos jours. Par conséquent nous nous trouvons impuissants face à ce défi technique.

Des systèmes d'environnements virtuels distribués comme SPIN [DDS+99] dupliquent tous les objets de l'environnement et exécutent le calcul localement et en parallèle entre tous les sites. Dans ce cas le délai sur le réseau n'est plus significatif du point de vue de l'animation, par contre le problème de l'interaction demeure sans solution. Même si le calcul de l'objet est dupliqué, l'interaction d'un utilisateur, elle, ne peut pas l'être.

8.2.2 Compréhension et conscience de l'univers virtuel

Nous avons expliqué dans le chapitre 4 l'impact de la compréhension qu'un utilisateur a de l'environnement virtuel sur la performance de ses actions au sein de cet univers. Les études qui ont été faites par Vaghi et Al.[VGB99] ainsi que celles faites par Fraser et Al. [FBHH99] convergent vers l'importance de faire prendre conscience à un utilisateur qu'un problème réseau est survenu. Cette prise de conscience rend l'utilisateur plus tolérant envers les anomalies provoquées par un fort délai ou une déconnexion momentanée. Un utilisateur peut aussi changer sa stratégie ou son centre d'intérêt en attendant le retour à la normale. Sans une telle compréhension l'utilisateur peut mettre en question la fiabilité du système ou même ses propres compétences ou encore son habileté.

8.3 Conception d'un mécanisme d'information

Rappelons que notre objectif est de rendre les utilisateurs conscients d'un éventuel problème au niveau réseau. En l'absence d'un mécanisme fournissant l'information d'une déconnexion, un utilisateur est complètement ignorant de ce qui se passe dans les couches basses du support d'exécution de l'environnement virtuel distribué. L'intérêt de fournir une telle information n'est pas d'impliquer l'utilisateur dans le fonctionnement du système mais plutôt de l'aider à être conscient qu'une anomalie liée à un problème réseau est quelque chose de "normal", ou en tout cas de pris en compte par le système.

La détection d'un problème de communication se fait au niveau de la couche de communication du support d'exécution d'environnements virtuels distribués. Par contre pour informer l'utilisateur nous devons utiliser des métaphores visuelles à plus haut niveau, celui du service de visualisation offert par le support d'exécution d'environnements virtuels. L'information ainsi que son traitement doivent donc s'acheminer à travers les différentes couches du système pour être enfin affichée sous forme d'une métaphore visuelle. Dans ce paragraphe nous allons étudier la conception de notre mécanisme assurant l'interception d'un problème, l'acheminement de l'information, la détection des objets touchés, et enfin la visualisation au sein d'un environnement virtuel.

Nous allons séparer les conséquences d'un délai ou déconnexion sur l'interaction suivant deux points de vues. Le premier point de vue est celui d'un utilisateur interagissant avec un objet calculé sur un site distant (objet miroir). Le second point de vue est celui d'un utilisateur interagissant sur un objet calculé localement sur son site (objet référentiel).

8.3.1 Point de vue d'un site détenant un miroir

Dans ce cas un utilisateur possède une copie miroir de son objet virtuel. Or, suite à un problème réseau ce miroir ne peut plus recevoir de mises-à-jour en provenance de son référentiel. Une fois le problème reconnu et traité au niveau de la couche noyau du support d'exécution d'environnements virtuels, notre système d'information doit localiser tous les miroirs du site local qui ont perdu la communication avec leur référentiel. Pour une question de cohérence ces miroirs ne doivent plus être interactifs. En effet, une fois un miroir déconnecté de son référentiel, il n'est plus en mesure de lui communiquer quelque changement que ce soit. Il est donc inutile et même dangereux de laisser un utilisateur interagir avec un tel miroir car cet utilisateur ne pourra avoir aucun retour d'interaction tant que la déconnexion perdurera. Ce serait donc seulement au moment de la reconnexion que l'utilisateur pourrait mesurer les conséquences de son interaction.

Le choix le plus raisonnable est donc d'interdire toute interaction avec un

miroir déconnecté de son référentiel. Ce référentiel pourra quant à lui évoluer lors de la déconnexion, créant donc une incohérence temporaire entre les univers déconnectés, la cohérence étant alors de retour au moment de la reconnexion.

Enfin, pour résumer, le système d'informations a trois tâches principales à remplir :

1. localiser tous les miroirs isolés qui ne communiquent plus avec leur référentiels.
2. interdire l'interaction avec ces miroirs.
3. révéler à l'utilisateur l'indisponibilité de ces objets momentanément à travers une métaphore visuelle adaptée.

8.3.2 Point de vue d'un site détenant un référentiel

Nous allons traiter dans cette section le cas où l'utilisateur possède l'objet référentiel sur son site local. Le problème dans ce cas est que cet utilisateur n'a aucune indication concernant ce que les autres utilisateurs perçoivent de ses interactions et manipulations sur l'objet pendant une déconnexion. Or, dans un environnement virtuel coopératif, la conscience mutuelle des interactions est primordiale pour une coopération efficace entre les utilisateurs. D'où l'importance de mettre l'utilisateur au courant que les autres utilisateurs ne perçoivent plus ses interactions au sein de l'environnement virtuel à cause de l'occurence d'un problème de communication entre les sites.

Dans ce contexte le système d'informations a deux tâches principales :

1. localiser tous les référentiels sur le site local qui possèdent au moins un miroi localisé sur un site déconnecté.
2. révéler à l'utilisateur à l'aide d'une métaphore adaptée que ses interactions sur ces objets ne sont pas perçues par d'autres utilisateurs à cause d'un problème réseau.

8.4 Mise en œuvre du mécanisme d'information

Le mécanisme d'information a été réalisé et implémenté dans différentes couches de la plate-forme OpenMASK. En effet la réalisation s'étend sur trois niveaux. La détection d'un problème réseau est réalisée au niveau de la couche de communication d'OpenMASK comme nous l'avons déjà détaillé dans le chapitre 5. Une fois un fort délai ou une déconnexion détecté, c'est à la charge du noyau de fournir les informations concernant tous les miroirs et les référentiels touchés et leur localisation sur les différents sites. Enfin les métaphores sont fournies à un niveau de service de mise en évidence.

Après la détection d'un problème de communication entre les différents sites participant à une simulation distribuée, le mécanisme d'information peut être activé ou désactivé de deux manières. Le premier mode d'activation est manuel, il est réalisé à la demande d'un utilisateur. Dans ce cas un utilisateur peut demander l'activation ou la désactivation du mécanisme à l'aide par exemple d'un bouton 3D qu'il retrouve dans la scène virtuelle. Le second mode est complètement automatique, le mécanisme est alors activé par le noyau d'OpenMASK suite à la détection d'un problème réseau. Il est aussi désactivé automatiquement une fois les perturbations terminées et la communication normale rétablie.

Nous allons détailler dans les sections suivantes la réalisation du mécanisme d'information suivant deux points de vues différents. Deux sous-systèmes intègrent le mécanisme d'information : le système de marquage et le système de génération d'échos. Le système de marquage est activé à partir du site détenant un ou des miroirs qui ne reçoivent plus de mises-à-jour de leur référentiel. Le système de génération d'échos est activé à partir du site détenant un ou des référentiels qui ont perdu la communication avec leurs miroirs à cause d'un problème réseau.

8.4.1 Système de marquage

Le système de marquage sert, comme son nom l'indique, à marquer les objets miroirs isolés [ED03]. Ce système, comme tous les services implémentés au dessus d'OpenMASK, est distribué et non centralisé. Il est exécuté sur chaque processus d'une manière indépendante de son exécution sur les autres processus. Après son activation sur un site suite à une détection de problème de communication, le système de marquages inspecte localement s'il existe des objets miroirs. Si des miroirs sont détectés, le système examine si parmi ces miroirs il y en a dont le référentiel associé est localisé sur un site déconnecté. Ceci veut dire que ces miroirs ne reçoivent plus de mises-à-jour en provenance de leurs référentiels, ce qui implique que leur état actuel est incertain. Ensuite le système de marquage filtre ces miroirs non-à-jour pour ne garder que les miroirs qui sont visualisés et ont une représentation géométrique. Ces derniers seront marqués visuellement par une sphère semi-transparente qui les englobe.

Figure 8.1 on peut voir le point de vue d'un utilisateur souhaitant interagir avec quelques objets qui sont des pièces détachées d'une voiture. Lors d'un problème réseau les objets miroirs sont englobés par les sphères semi-transparentes que l'on peut voir.

Le marquage des objets a deux objectifs principaux. Le premier est d'attirer l'attention de l'utilisateur sur le fait que les objets marqués ne sont peut être pas à jour. Ils sont potentiellement en train d'être manipulés par un autre utilisateur sans qu'on puisse perçevoir ses manipulations à cause d'un problème réseau. Le

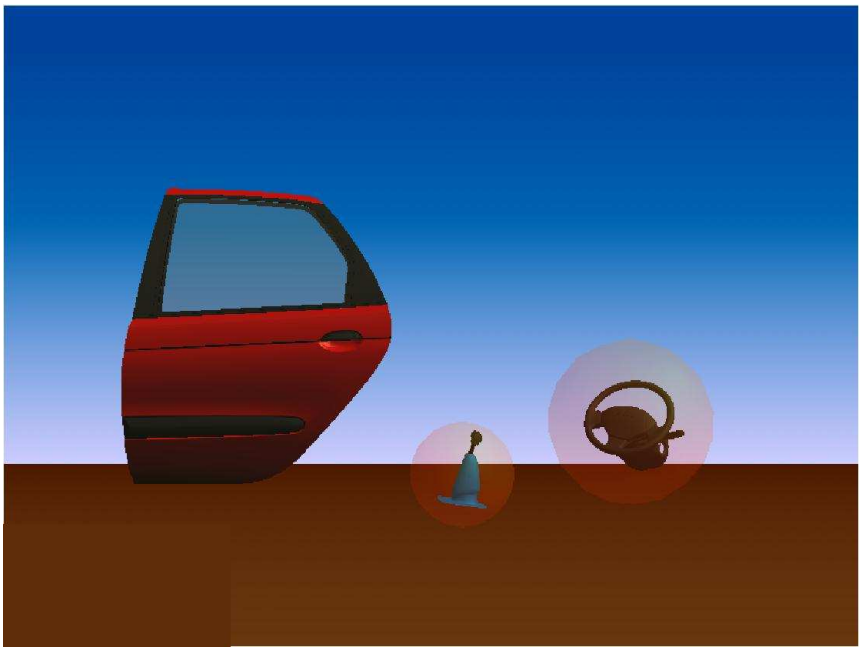

FIG. 8.1 – Système de marquage

second objectif est d'interdire toute tentative d'interaction avec ces objets pour les raisons que nous avons citées dans les sections précédentes. Le principe est le suivant : pour interagir avec un objet OpenMASK, il faut pouvoir sélectionner via la souris ou n'importe quel dispositif et interagir avec. Or, englober un objet par un autre qui n'est pas interactif empêche toute prise de contrôle de l'objet englobé, car il n'est alors plus possible de le désigner directement.

8.4.2 Système de génération d'échos

Le système de génération d'échos agit localement sur chaque site détenant des référentiels. Après son activation sur un site suite à une détection de problème de communication, le système de génération d'échos inspecte localement s'il existe des objets référentiels. Au cas où des référentiels sont localisés sur le site local, le système d'information doit faire comprendre à l'utilisateur de ce site que les évolutions de ces objets (qui peuvent être dues à des interactions) ne sont pas perçues par tous les utilisateurs. Deux versions du système d'échos ont été réalisées. La première version est une implémentation statique, la deuxième est dynamique. Ces deux versions sont présentées dans ce qui suit ainsi que leurs avantages et inconvénients.

8.4.2.1 Version statique

La version statique est la première version implémentée du système d'échos [ED03]. Sur chaque processus nous associons un objet écho à tous les miroirs de l'environnement virtuel. L'objet écho a la même forme géométrique que son miroir associé, mais cette forme est rendue semi-transparente et son rôle est de suivre le miroir auquel il est associé. Les échos sont spécifiés manuellement dans un fichier de configuration et initialisés au début de la simulation. Or, un objet écho est un objet OpenMASK standard c'est-à-dire un objet référentiel sur le site de sa création et il aura à son tour des miroirs sur les autres sites. Les miroirs d'un écho sont visualisés au même endroit que l'objet original associé (Fig. 8.2). Dans le cas d'une simulation qui se déroule sans problème de communication au niveau réseau, l'écho ne se remarque pas car il demeure masqué par l'objet original. En cas de délai sur le réseau la communication entre les référentiels et leurs miroirs prend un peu plus de temps que la normale. Dans ce cas nous percevons un décalage entre le mouvement d'un objet référentiel et celui de son miroir sur un site distant. Ce décalage est rendu visible directement à l'utilisateur à travers le décalage entre l'objet original et l'objet écho : l'objet écho est considéré comme un indicateur de la position actuelle de l'objet distant, c'est donc de cette façon que l'objet référentiel associé est perçu par les autres utilisateurs. Si un site distant se déconnecte totalement, l'écho se fige sur l'écran, indiquant que le miroir distant ne reçoit plus du tout les mises-à-jour.

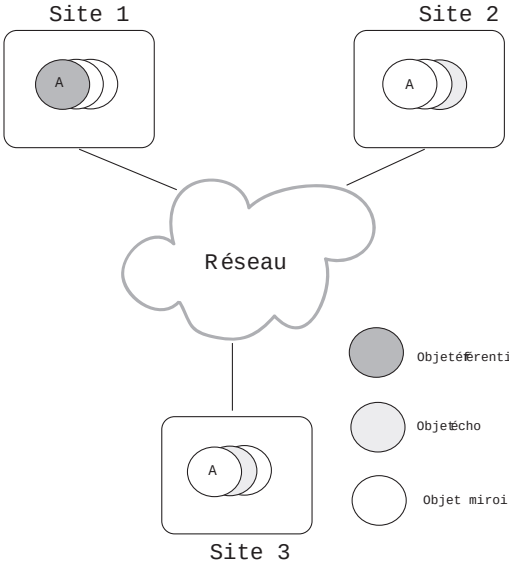

FIG. 8.2 – Système de génération d'échos statiques

Cette méthode est facile à réaliser, mais elle a quelques inconvénients et limi-

tations. L'inconvénient majeur est que chaque référentiel dans l'environnement virtuel aura autant d'échos que de miroirs associés (*i.e.* nombre de sites participants). Ceci peut encombrer visuellement la scène pendant l'interaction avec l'objet. De plus, le temps de calcul des échos peut devenir important surtout si le nombre de sites participant s'est élevé. Par exemple, supposons qu'une simulation distribuée comporte N sites. Si nous disposons de X Objets dans l'environnements, ceci veut dire qu'on a X référentiels et $X(N-1)$ miroirs. En appliquant la méthode d'échos statiques nous devons obtenir $X(N-1)$ échos référentiels. Ces derniers vont générer $X(N-1)(N-1)$ échos miroirs. Nous pouvons donc facilement imaginer l'explosion du nombre d'échos miroirs créés dans le cas d'un grand nombre de sites participants. Pour éviter ces inconvénients nous avons décidé de modifier complètement la façon de créer les échos, ce qui a abouti à une version dynamique améliorée, présentée dans ce qui suit.

8.4.2.2 Version dynamique

Dans la version dynamique du système d'échos nous ne créons initialement aucun écho. Les échos sont créés dynamiquement seulement après qu'un problème réseau soit signalé. Le système de génération d'échos génère alors des objets clones de ces référentiels qui ont le même état et la même forme géométrique que ces derniers sauf qu'il sont rendus semi-transparents. Ces objets sont créés avec le même état que celui de leur référentiel associé au moment de la détection du problème de communication. Ils conservent donc le dernier état pris par les référentiels au moment de l'occurence du problème réseau. Une fois ces échos créés, ils n'évoluent plus tant que la déconnexion est en cours. Même si les objets référentiels changent d'état (suite par exemple à une manipulation quelconque de l'utilisateur), les échos restent figés dans leur état actuel. Un écho figé schématise l'état d'un miroir sur un site distant qui ne reçoit plus de mises-à-jour de son référentiel et qui est figé à son tour.

Le propriétaire du référentiel peut continuer à interagir avec son objet tout en ayant conscience que ses manipulations ne sont pas vues par tous les utilisateurs. On peut donc espérer un retour à la normale moins chaotique pour les sites ayant eu des miroirs déconnectés. Après le rétablissement de la connexion le système de génération est désactivé et les échos disparaissent indiquant le retour à la nomale.

Figure 8.3 nous pouvons voir le point de vue d'un utilisateur interagissant avec quelques objets. La porte et le volant sont des objets référentiels. Lors d'un problème de communication avec les autres sites, nous pouvons voir apparaître les objets échos qui schématisent l'état des objets tel qu'il est perçu sur les sites qui ne reçoivent plus de mises-à-jour.

Le système de génération d'échos n'est pas seulement activé dans le cas d'un site détenant un référentiel. Ce système peut être aussi activé sur les sites détenant

FIG. 8.3 – Système de création d'échos

des miroirs qui ont des "frères" miroirs isolés. Deux miroirs sont dits frères s'ils sont miroirs d'un même référentiel distant. Dans un tel cas, l'utilisateur peut toujours interagir avec son objet miroir car il n'a pas perdu la communication avec son référentiel. Par contre ses interactions ne seront pas perçues par les autres sites détenant des miroirs qui ne reçoivent plus de mises-à-jour suite à un problème réseau quelconque.

L'avantage de la version dynamique est que les échos ne sont créés que suite à un problème et seulement sur les sites concernés, contrairement à la version statique qui crée les échos dès le début et sur tous les sites. Dès que le problème est terminé les échos sont détruits.

Dans le cas d'une simulation comportant plusieurs sites participant (nombre de sites strictement supérieur à deux) nous pouvons faire l'un des trois choix suivants :

- créer un seul écho en cas de problème réseau quelque soit le nombre de sites isolés. Ceci indique à l'utilisateur qu'au moins un site ne perçoit pas ses interactions et plus généralement les évolutions des objets référentiels hébergés par ce site. Dans ce cas l'utilisateur n'a aucune connaissance concernant lequel ou lesquels des sites sont déconnectés.
- créer autant d'échos que de sites isolés suite à un problème réseau. Dans ce cas l'utilisateur peut savoir combien de sites ne reçoivent plus ses mises-à-jour sans toutefois pouvoir les identifier.

– créer autant d'échos que de sites isolés suite à un problème en associant une couleur différente à chaque site de la simulation. Dans ce cas l'utilisateur peut savoir combien ont été isolé et lesquels. On peut aussi par exemple afficher les noms des sites au lieu d'utiliser des codes de couleurs.

8.4.3 Les objets locaux

Nous avons déjà expliqué que les métaphores qui servent à fournir la conscience d'un problème à un utilisateur sont visualisées sous la forme d'objets 3D (sphères semi-transparentes et échos). Dans la version distribuée d'OpenMASK, tous les objets visualisables sont répliqués sur les différents sites selon le principe référentiel/miroir. Un objet visualisable doit donc être vu de la même manière sur tous les sites. Or, les échos et les miroirs ne doivent pas être perçus sur tous les sites. Suite à un problème réseau, chaque site perçoit les conséquences de ce problème d'une manière différente. Par conséquent la méthode standard de création et de réplication d'objets d'OpenMASK n'était pas convenable pour la création d'objets de métaphores sauf dans le cas du système de création d'échos statiques qui en tire profit. D'où la nécessité de créer ces objets d'une manière spéciale sans les répliquer forcément. Nous avons donc défini un nouveau type d'objet OpenMASK non distribuable que nous avons baptisé "objet local". Un objet local OpenMASK est un objet qui est créé sur un site spécifique et ne doit pas avoir de réplications sur les autres sites. En d'autres termes un objet local est un référentiel qui ne doit pas avoir de miroirs sur les autres sites. Ces objets locaux servent pour l'affichage des objets de métaphores sur un site spécifique.

8.5 Intérêt du système d'information

Pour illustrer l'intérêt de nos travaux, nous allons présenter un exemple fictif d'utilisation de ces mécanismes. Cet exemple est le suivant : trois bibliothécaires, situés sur des sites distants, doivent ranger différentes sections d'une bibliothèque virtuelle, et des perturbations réseau vont apparaître progressivement durant leur session de travail.

Voyons comment notre système d'information, couplé avec le mécanisme de migration d'objets, peut leur permettre de réaliser cette tâche malgré les perturbations au niveau du réseau :

1. dès que des problèmes réseau vont commencer à se manifester, les bibliothécaires vont pouvoir se rendre compte, grâce aux mécanismes de marquage et d'écho, de la présence d'anomalies sur le réseau. Ces anomalies vont se manifester par l'apparition et la disparition des marques et des échos que

le système associe aux différents objets virtuels (livres dans notre contexte) pendant que les utilisateurs (les bibliothécaires) accomplissent leur travail.

2. en plus de la conscience du problème, les marques et les échos fournissent aux utilisateurs une information à propos des objets dont ils vont perdre le contrôle dans le cas d'une déconnexion temporaire de leur site. En d'autres termes, un utilisateur perdra le contrôle de tous les objets marqués par une sphère et il gardera le contrôle de tous les objets couplés avec des échos.

3. chaque bibliothécaire a intérêt à garder le contrôle de tous les livres qui doivent être rangés dans sa section pour ne pas interrompre son activité de rangement dans le cas d'une éventuelle déconnexion temporaire de son site. Chaque bibliotécaire va donc pouvoir demander au système de faire migrer sur son site tous les livres qu'il aura à ranger et dont les référentiels sont actuellement sur un site distant avec lequel il y a des perturbations réseau. À ce stade, c'est au mécanisme de migration d'assurer la migration des référentiels de tous les livres demandés par un bibliothécaire vers son site local.

4. une fois la migration de ces objets terminée, les utilisateurs peuvent désactiver temporairement, s'ils le souhaitent, le système d'information, afin de pouvoir plus facilement se concentrer sur leur travail. De cette manière, chaque utilisateur peut travailler dans sa section sans être perturbé par les dérangements du réseau. On peut également espérer que ces problèmes réseau vont progressivement disparaître.

5. quand leur section est rangée, les bibliothécaires peuvent alors passer à une autre section. On espère donc ici que les problèmes réseau auront complètement disparu durant cette phase de rangement. Si ce n'est pas le cas mais si le réseau fonctionne quand même en mode dégradé, chaque bibliothécaire peut alors se diriger vers une nouvelle section, faire de nouveau migrer les livres qui l'intéressent, et travailler localement en attendant le retour à la normale du réseau.

Dans un tel contexte, nos travaux vont donc permettre aux utilisateurs de travailler malgré des perturbations réseau, en se focalisant sur des tâches qu'ils peuvent réaliser seuls plutôt que sur des tâches nécessitant une coopération étroite entre utilisateurs, en espérant pouvoir réaliser ces tâches fortement coopératives une fois les perturbations réseau terminées. De plus, ces utilisateurs vont bénéficier d'un plus grand confort d'utilisation du système car ils vont pouvoir travailler avec des objets qu'ils auront rapatriés sur leur site pour interagir avec eux sans être affectés, temporairement, par ces perturbations réseau.

8.6 Conclusion et perspectives

Les solutions techniques présentées dans les chapitres précédents ont pour objectif de préparer le support d'exécution d'environnements virtuels distribués afin d'être capable de gérer un problème au niveau réseau. Or, les conséquences d'un problème réseau ne peuvent pas être neutralisées entièrement quelle que soit la technique utilisée. D'où la nécessité de préparer l'utilisateur à envisager ce problème à son tour. Les métaphores définies et réalisées dans ce chapitre ont pour but de traiter les conséquences d'un problème réseau au niveau utilisateur en lui fournissant la conscience de l'existence d'un problème insurmontable techniquement. Pour résumer, notre philosophie se traduit donc par l'inutilité d'essayer de cacher ce genre de problème tant qu'on ne peut pas cacher ses conséquences. Le dégât d'un problème réseau peut être moins important si l'utilisateur a bien conscience de son existence.

Deux métaphores correspondant à deux points de vues différents ont été présentées et étudiées dans ce chapitre. La sphère semi-transparente sert à empêcher les tentatives d'interactions avec un objet miroir tout en donnant la conscience à l'utilisateur que cet objet n'est plus à jour suite à un problème de communication. L'écho semi-transparent est un clone de l'objet original et sert à donner à l'utilisateur une idée concernant la façon dont l'objet en question est vu par les utilisateurs distants suite à un problème de communication. Ces deux métaphores sont gérées par deux mécanismes : le système de marquages et le système d'échos. Ces mécanismes sont implémentés d'une manière distribuée et ne dépendent pas d'un serveur central.

Nous pensons que des extensions peuvent enrichir encore notre étude. Comme par exemple définir, conçevoir et étudier d'autres métaphores d'interactions en cas de problème réseau. Ces métaphores doivent être significatives et le plus intuitives possible. Une autre extension possible est d'essayer de déterminer ce qu'un utilisateur a vraiment besoin de savoir et quelles sont les connaissances primordiales pour une coopération réussie au sein d'un environnement virtuel distribué. Ce dernier point nécessitera peut être une collaboration avec des chercheurs en psychologie pour bien analyser le comportement d'un utilisateur d'environnement virtuel.

Conclusion

Nous venons de présenter dans cette thèse notre manière de gérer les conséquences d'un problème réseau lors d'une simulation distribuée et coopérative d'environnements virtuels. Nous avons pu voir que le réseau est un facteur qui agit directement sur le comportement des des environnements virtuels distribués en perturbant le fonctionnement de leur support d'exécution. Si un problème de transmission de données surgit sur le réseau, ce problème va directement affecter le fonctionnement de l'interaction à distance ainsi que la coopération entre plusieurs utilisateurs.

Dans les travaux présentés dans l'état de l'art, nous avons pu remarquer que les problèmes réseau qui peuvent surgir lors d'une simulation distribuée ne sont pris en compte que par très peu de support d'exécution d'environnements virtuels. Or, nos travaux ont pour but de mettre l'accent sur ces problèmes et d'essayer de les gérer au sein du noyau du support d'exécution d'environnements virtuels distribués.

Nous avons conçu et réalisé plusieurs mécanismes qui sont implémentés à des niveaux différents de la plate-forme OpenMASK. Suite à ces réalisations, une simulation distribuée au dessus d'OpenMASK n'est plus bloquée par un problème au niveau réseau (ou même suite à la défaillance d'un des sites participant à une simulation). Au moment de la déconnexion le noyau OpenMASK détecte la perte de communication avec le site concerné et il alarme les utilisateurs en masquant l'interaction avec les objets miroirs (grâce au système du marquage), puis en créant des échos schématisant l'état des objets ne recevant pas de mise à jour sur les autres sites (grâce au système d'échos).

Nous avons également conçu et réalisé un mécanisme de migration d'objets permettant de faire migrer les objets critiques d'un site vers un autre et ce pour prévenir la perte d'accès à ces objets dans le cas d'un problème réseau. De cette manière, un utilisateur a la possibilité de faire migrer sur son site les objets qui l'intéressent. Cet utilisateur n'a alors plus à se soucier de la perte de communication avec le site qui les détenait initialement.

Enfin nous avons conçu et réalisé un mécanisme d'ajout et de suppression dynamique de sites au cours d'une simulation distribuée.

Les résultats de l'ensemble de ces travaux de recherche et leur implémenta-

tion au sein de la plate-forme OpenMASK offrent de nouvelles perspectives aux concepteurs d'univers virtuels. Parmi les perspectives les plus intéressantes on peut citer la possibilité d'utiliser le mécanisme de migration pour permettre à terme le passage à l'échelle sur des grilles ainsi que l'équilibrage de charges. Au niveau de la perception visuelle des problèmes réseau offerte aux utilisateurs il serait intéressant de faire évaluer plusieurs types de métaphores visuelles. Ces tests, éventuellement en présence de psychologues, pourraient nous aider à mieux comprendre l'impact positif ou négatif de l'information apportée par ces métaphores et donc devrait nous aider à déterminer les métaphores réellement utiles aux utilisateurs.

Bibliographie

[AB92] Aukstakalnis, S. and Blatner, D. *Silicon Mirage*. S.F. Roth (Berkely, CA: Peachpit Press), 1992.

[ABC+03] Arnaldi, B., Burkhardt, J.M., Chauffaut, A., Coquillart, S., Duval, T., Donikian, S., Fuchs, P., Grosjean, J., Harrouet, F., Klinger, E., Lourdeaux, D., Mellet d'Huart, D., Moreau, G., Paljic, A., Papin, J.P., Stergiopoulos, P., Tisseau, J., and Viaud-Delmon, I. Le traité de la réalité virtuelle, 2ème édition. 2003.

[BBFG94] Benford, S., Bowers, J., Fahlén, L., and Greenhalgh, C. Managing mutual awareness in collaborative virtual environments. *Proceedings of the conference on Virtual reality software and technology, Singapore*, pages 223 – 236, 1994.

[BDG+93] Beguelin, A., Dongarra, J., Geist, A., Manchek, R., Otto, S., and Walpole, J. Pvm: Experiences, current status and future directions. *Proceedings of Supercomputing*, pages 765–766, 1993.

[BFT99] Bolot,J-C., Fosse-Parisis, S., and Towsley, D. Adaptive fec-based error control for internet telephony, vol.3. *Proc. Infocom '99, New York*, pages 1453–60, 1999.

[BH97] Bowman, D. and Hodges, L. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environements. *Proceedings of Symposium on Interactive 3D Graphics*, pages 35–38, 1997.

[BL85] Barak, A. and Litman, A. Mos: a multicomputer distributed operating system. *Software Practice and Experience*, pages 725–737, August 1985.

[BLB95] Barak, A., Laden, O., and Braverman, A. The *NOW* mosix and its preemptive process migration scheme. *Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments, 7(2)*, pages 5–11, 1995.

[BP01] Barbieri, T. and Paolini, P. Cooperation metaphors for virtual museums. *Proceedings Museums and Web 2001, Seattle, USA*, 2001.

[Bro95] Broll, W. Interacting in distributed collaborative virtual environments. *Proceedings of the IEEE Virtual Reality Annual International Symposium - VRAIS'95, Los Alamitos, USA*, pages 148–155, 1995.

[Bro98] Broll, W. Dwtp an internet protocol for shared virtual environments. *Proceedings of the third symposium on Virtual reality modeling language*, pages 49–56, 1998.

[BS85] Barak, A. and Shiloh, A. A distributed load-balancing policy for a multicomputer. *Software Practice and Experience*, pages 901–913, September 1985.

[Bur93] Burdea, G. Virtual reality systems and applications. *Electro'93 International Conference, Short Course, Edition, NJ*, 1993.

[BWA96] Barrus, J., Watters, R., and Anderson, D. Locales and beacons: Efficient and precise support for large multi-user virtual environments. *IEEE Computer Graphics and Apllications*, 16(6):50–57, 1996.

[BZWM97] Brutzman, D., Zyda, M., Watsen, K., and Macedonia, M. Virtual reality transfer protocol (vrtp) design rationale. *Workshop on Enabling Technology : Infrastructure for Collaborative Enterprise (WET ICE)*, pages 179 – 186, 1997.

[CCC+95] Casas, J., Clark, D.L., Conuru, R., Otto, S.W., Prouty, R.M., and Walpole, J. Mpvm: A migration transparent version of pvm. *Computing Systems, 8(2)*, pages 171–216, 1995.

[CDG+93] Calvin, J., Dickens, A., Gaines, B., Metzger, P., Miller, D., and Owen, D. The simnet virtual world architecture. *Proceedings of the IEEE Annual International Symposium (VRAIS)*, pages 450–455, 1993.

[CH93] Carlsson, C. and Hagsand, O. Dive - a multi user virtual reality system. *Proceedings of VRAIS93*, pages 394–400, 1993.

[Cho90] Chowdhury, S. The greedy load sharing algorithm. *Journal of Parallel and Distributed Computing, Volume 9*, pages 93–99, 1990.

[DCDK98] Donikian, S., Chauffaut, A., Duval, T., and Kulpa, R. Gasp: from modular programming to distributed execution. In *Computer Animation'98, IEEE, Philadelphie, USA*, pages 79–87, june 1998.

[DDS+99] Dumas, C., Degrande, S., Saugis, G., Chaillou, C., Viaud, M., and Plenacoste, P. Spin: a 3d interface for cooperative work. *Virtual Reality Society Journal*, 1999.

[Dee93] Deering, S. Mbone-the multicast backbone. *CERFnet Seminar*, 1993.

[DFW98] Dahmann, J., Fujimoto, R., and Weatherly, R. The dod high level architecture : An update. *Proceedings of the 30th conference on Winter simulation WSC '98*, pages 797 – 804, 1998.

[DIS94] DIS Steering Committee. The dis vision : A map to the future of distributed simulation. *Ver. 1, IST-SP-94-01*, 1994.

[DL02] Duval, T. and Le Tenier, C. Interactions 3d coopératives en environnements virtuels avec openmask pour l'exploitation d'objets techniques. *Virtual Concept'2002, Biarritz, France*, pages 116–121, 2002.

[ED03] El Zammar, C. and Duval, T. Management and awareness of delay perception when interacting within networked virtual environments. *VRIC'2003, Laval, France*, 2003.

[FBHH99] Fraser, M., Benford, S., Hindmarch, J., and Heath, C. Supporting awareness and interaction through collaborative virtual interfaces. *UIST'99, Asheville, USA*, pages 27–36, 1999.

[FGV+00] Fraser, M., Glover, T., Vaghi, I., Benford, S., Greenhalgh, C., Hindmarch, J., and Heath, C. Revealing the realities of collaborative virtual reality. *CVE'2000, San Francisco*, pages 29–37, 2000.

[FJL+96] Floyd, S., Jacobson, V., Liu, C., McCanne, S., and Zhang, L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, pages 784–803, 1996.

[Fre91] Fredman, D. Experience building a process migration subsystem for unix. *Proceedings of the Winter USENIX Conference*, pages 349–355, 1991.

[Fun95] T. Funkhouser. Ring: A client-server system for multi-user virtual environments. *Symposium on Interactive 3D Graphics*, pages 85 – ff, 1995.

[Gau95] Gaudrault, A. Load balancing in a distributed virtual reality environment. *in fulfilment of the thesis requirement for the degree of Master of Mathematics in Computer Science Waterloo, Ontario, Canada*, 1995.

[GB95] Greenhalgh, C. and Benford, S. Massive: a distributed virtual reality system incorporating spatial trading. *Proc. IEEE 15th International Conference on Distributed Computing Systems (DCS'95)*, page 27, 1995.

[GB97] Greenhalgh, C. and Benford, S. Boundaries, awareness and interaction in collaborative virtual environments. *Proceedings of the 6th International WET-ICE*, pages 193–198, 1997.

[GBD+04] Gutwin, C., Benford, S., Dyck, J., Fraser, M., Vaghi, I., and Greenhalgh, C. Revealing delay in collaborative environments. *CHI 2004*, pages 503–510, 2004.

[GG98] Gutwin, C. and Greenberg, S. Design for individuals, design for groups: Tradeoffs between power and workspace awareness. *CSCW'98, Seattle, Washington, US*, pages 207–216, 1998.

[GG99] Gutwin, C. and Greenberg, S. A framework of awareness for small groups in shared-workspace groupware. *Technical Report 99-1, Department of Computer Science, University of Saskatchewan, Canada*, 1999.

[GGC96] Greenberg, S., Gutwin, C., and Cockburn, A. Awareness through fisheye views in relaxed-wysiwis groupware. *Proceedings of Graphics Interface*, pages 28–38, 1996.

[Gib83] Gibson, W. Neuromancer. *Novel*, 1983.

[Gig93] Gigante, M. In *Virtual Reality: Definitions, History and Applications.*, pages 3–14. Virtual Reality Systems, Academic-Press, ISBN 0-12-22-77-48-1, 1993.

[GLK94] Gossweiler, R., Laferriere, R., and Keller, M. An introductory tutorial for developing multi-user virtual environments. *Presence: Teleoperators and Virtual Environments, Vol. 3, No. 4*, pages 255–264, 1994.

[Gru94] Grudin J. Computer-supported cooperative work: History and focus. *Computer, vol. 27*, pages 19–26, 1994.

[Gut01] Gutwin, C. Effects of network delay on group work in shared workspaces. *Proc. ECSCW*, pages 299–318, 2001.

[Han97] Hand, C. A survey of 3d interaction techniques. *Computer Graphics Forum*, pages 269–281, 1997.

[HDW03] Huang, J., Du, Y., and Wang, C. Design of the server cluster to support avatar migration. *IEEE Virtual Reality, Los Angeles*, page 7, 2003.

[HL95] Hofer, R. and Loper, L. Dis today. *Proceedings of the IEEE*, pages 1124–1137, 1995.

[HOK97] Hsu, C-Y., Ortega, A., and Khansari, M. Rate control for robust video transmission over burst-error wireless channels. *Visual Commun. Image Processing, VCIP'97*, pages 1200–1211, 1997.

[Isd93] Isdale, J. What is virtual reality ? a homebrew introduction and information resource list. 1993.

[Kar96] Karlsson, G. Asynchronous transfer of video. *IEEE Communications*, pages 118–126, 1996.

[Kaz95] Kazman, R. Hidra: An architecture for highly dynamic physically based multi-agent simulations. *International Journal in Computer Simulation, 5*, pages 149–164, 1995.

[Kaz96] Kazman, R. Load balancing, latency management and separation of concerns in a distributed virtual world. *Parallel Computations - Paradigms and Applications*, pages 480–497, 1996.

[KR03] Kurose, J. and Ross, K. *Analyse structurée des réseaux.* Pearson Education, 2003.

[Kru91] Krueger, M. Artificial reality ii. *Addison Wesley, second edition. ISBN 0-201-52260-8*, 1991.

[LDJ+97] Leigh, J., DeFanti, T., Johnson, A., Brown, M., and Sandin, D. Global tele-immersion: Better than being there. *Proceedings of ICAT '97, University of Tokyo, Japan*, pages 10–17, 1997.

[LG93] Liang J. and Green M. Geometric modelin using six degrees of freedom input devices. *Proceedings of the 3d International conference abd CAD and Computer hics (Beijing, China)*, pages 217–222, 1993.

[Lit87] Litzkow, M. Remote unix - turning idle workstations into cycle servers. *Proceedings of the Summer USENIX Conference*, pages 381–384, 1987.

[LS92] Litzkow, M. and Solmon, M. Supporting checkpointing and process migration outside the unix kernel. *Proceedings of the USENIX Winter Conference*, pages 283–290, 1992.

[MAC+02] Margery, D, Arnaldi, B., Chauffaut, A., Donikian, S., and Duval, T. Openmask: Multi-threaded or modular animation and simulation kernel or kit : a general introduction. *VRIC 2002 Proceedings*, 2002.

[Mar01] Margery, D. Environnement logiciel temps-réel distribué pour la simulation sur réseau de pc. *Document de thèse*, 2001.

[MHKE99] Mauve, M., Hilt, V., Kuhmunch, C., and Effelsberg, W. Rtp/i - an application-layer protocol for the transmission of interactive media with real-time characteristics. *Proceedings IEEE Multimedia Systems*, 1999.

[Min95] Mine, M. Virtual environment interaction techniques. *UNC Chapel Hill CS Dept., Technical Report TR95-018*, 1995.

[MN94] Murakami, T. and Nakajima, N. Direct and intuitive input device for 3d shape deformation. *Proceedings of CHI*, pages 465–470, 1994.

[MT95] Miller, D. and Thorpe, J. Simnet : The advent of simulator networking. *Proceedings of the IEEE*, pages 1114–1123, 1995.

[MZP+94] Macedonia, M., Zyda, M., Pratt, D., Barham, P., and Zeswitz, S. Npsnet: a network software architecture for large scale virtual environments. *Presence, Vol3, No.4*, pages 265–287, 1994.

[NJ93] Null, C.H. and Jenkins, J.P. Nasa virtual environment research,applications, and technology. In *Washington, DC: National Aeronotics and Space Administration*, 1993.

[OF03] Olwal, A. and Feiner, S. The flexible pointer: An interaction technique for selection in augmented and virtual reality. *Conference Supplement of ACM Symp. on User Interface Software and Tech. (UIST '03)*, pages 81–82, 2003.

[OG02] Oliveira, J. and Georganas, N. Velvet: An adaptive hybrid architecture for very large virtual environments. *IEEE*, pages 555–580, 2002.

[OKKT93] Ohya, J., Kitamura, Y., Kishino, F., and Terashima, N. Virtual space teleconferencing: Real-time reproduction of 3d human images. *Journal of Visual Communication and Image Representation, Vol. 6, No. 1*, pages 408–414, 1993.

[Par94] Partridge, C. Gigabit networking. *Addison-Wesley, Reading, Massachusetts*, 1994.

[PBF02] Pinho, M., Bowman, D., and Freitas, C. Cooperative object manipulation in immersive virtual environments: Framework and techniques. *VRST'02*, pages 171–178, 2002.

[PBZI96] Poupyrev, I., Billinghurst, M., Zelzink, R., and Ichikawa, T. Go-go interaction technique: Non-linear mapping for direct manipulaiton in vr. *Proceedings of UIST'96*, pages 79–80, 1996.

[PFC+97] Pierce, J., Forsberg, A., Conway, M., Hong, S., Zelzink, R., and Mine, M. Image plane interaction techniques in 3d immersive environments. *Proceedings of Symposium on Interactive 3D Graphics, Providence, Rhode Island, United States*, pages 39 – ff, 1997.

[PK99] Park, K. and Kenyon, R. Effects of network characteristics on human performance in the collaborative virtual environment. *Proc. of IEEE Virtual Reality'99, Bonn*, pages 104–111, 1999.

[PWBI98] Poupyrev, I., Weghorst, S., Billinghurst, M., and Ichikawa, T. Egocentric object manipulation in virtual environments: Empirical evaluation of interaction techniques. *Computer Graphics Forum 17(3)*, pages 41–52, 1998.

[Rhe91] Rheingold, H. Virtual reality. *Secker and Warburg, London*, 1991.

[RJ01] Ruhleder, K. and Jordan, B. Co-constructing non-mutual realities: Delay generated trouble in distributed interaction. *Journal of CSCW, 10(1)*, pages 113–138, 2001.

[RT95] Roschelle, J. and Teasley, S. D. The construction of shared knowledge in collaborative problem solving. *C. O'Malley (Ed.), Computer Supported Collaborative Learning*, pages 69–97, 1995.

[Saw91] Sawler, Robert, Matusof. Issues concerning cue correlation and synchronous networked simulators. *AIAA*, 1991.

[SB89] Schroeder, M. and Burrows, M. Performance of firefly rpc. *SRC Research report 43*, pages 1–17, 1989.

[SCFJ96] Schulzrinne, H., Casner, S., Frederick, R., and Jacobson, J. Rtp : A transport protocol for real-time applications. *IETF RFC1889*, 1996.

[SCP95] Stoakly, R., Conway, M., and Pausch, R. Virtual reality on a wim: Interactive worlds in miniature. *proceedings of CHI'95*, pages 265–272, 1995.

[Sin94] Singh, G. Bricknet: A software toolkit for networks-based virtual worlds. *Presence: Teleoperators and Virtual Environments, M1T Press, Boston*, pages 19–34, 1994.

[Sko95] Skordos, P. Parallel simulation of subsonic fluid dynamics on a cluster of workstations. *Proceedings of the fourth IEEE International Symposium on High Performance Distributed Computing, Pentagon City, Virginia*, pages 6–16, 1995.

[SSL96] Salvador, T., Scholtz, J., and Larson, J. The denver model for groupware design. *SIGCHI Bulletin, 28(1)*, pages 52–58, 1996.

[SW94] Snowdon, D. and West, A. The aviary vr system: A prototype implementation. *6th ERCIM Workshop*, 1994.

[SWH93] Snowdon, D., West, A., and Howard, T. Towards the next generation of human-computer interface. *Informatique 93: Interface to Real and Virtual Worlds*, pages 399–408, 1993.

[SZ99] Singhal, S. and Zyda, M. Networked virtual environments - design and implementation. *Addison Wesley*, 1999.

[SZaF96] Stone, S., Zyda, M., and Brutzman, D. abd Falby, J. Mobile agents and smart networks for distributed simulations. *Proceedings of the 14th Distributed Simulations Conference, Orlando, FL*, pages 11–15, 1996.

[Tan90] Tanenbaum, A. *Réseaux : architectures, protocoles, applications.* InterÉditions, Paris, 1990.

[TBG+00] Torguet, P., Balet, O., Gobbetti, E., Jessel, J-P., Duchon, J., and Bouvier, E. Cavalcade a system for collaborative virtual prototyping. *International Journal of Design and Innovation Research (IJODIR)*, pages 76–89, 2000.

[Tis01] Tisseau J. Réalité virtuelle : autonomie in virtuo. *HDR, document de synthèse, Université de Rennes 1*, 6 décembre 2001.

[TLC85] Theimer, M., Lantz, K., and Cheriton, D. Preemptable remote execution facilities for the v system. *Proceedings of the 10th ACM Sumposium on OS Principles*, pages 2–12, 1985.

[Tra99] Tramberend, H. Avocado: A distributed virtual reality framework. *Proceedings of IEEE Virtual Reality*, pages 14–21, 1999.

[VGB99] Vaghi, I., Greenhalgh, C., and Benford, S. Coping with inconsistency due to network delays in collaborative virtual environments. *Proceedings of the ACM symposium on Virtual reality software and technology*, pages 42–49, 1999.

[VLR+03] Vallée, G., Lottiaux, R., Rilling, L., Berthou, J-Y, Dutka-Malhen, I., and Morin, C. A case for single system image cluster operating systems: Kerrighed approach. *Parallel Processing Letters, 13(2)*, pages 95–122, 2003.

[VMB+02] Vallée, G., Morin, C., Berthou, J-Y, Dutka-Malhen, I., and Lottiaux, R. Process migration based on gobelins distributed shared memory. *Proc. of the workshop on Distributed Shared Memory (DSM'02) in CCGRID 2002/, Berlin, Germany*, pages 325–330, 2002.

[WAB+97] Waters, R., Anderson, D., Barrus, J., Brogan, D., Casey, M., McKeown, S., Nitta, T., Sterns, I., and Yerazunis, W. Diamondpark and spline: A social virtual reality system with 3d animation, spoken interaction and runtime modificability. *Presence : Teleoperators and Virtual Environements*, 1997.

[War90] Ware, C. Using hand position for virtual object placement. *Visual Computing 6*, pages 245–253, 1990.

[WAS97] Waters, R., Anderson, D., and Schwenke, D. Design of the interactive sharing transfer protocol. *Proceedings of the 6th Workshop on Enabling Technologies on Infrastructure for Collaborative Enterprises*, pages 140 – 147, 1997.

[WHH+93] West, A., Howard, T., Hubbold, R., Murta, A., Snowdon, D., and Butler, D. Aviary - a generic virtual reality interface for real applications. *Virtual Reality Systems (sponsored by the BCS), May 1992. Also published in Virtual Reality Systems, eds R.A. Earnshaw, M.A. Gigante and H. Jones, Academic Press, 1993*, pages 213–236, 1993.

[WJ88] Ware, C. and Jessome D.R. Using the bat: A six-dimensional mouse for object placement. *IEEE Computer Graphics and Applications*, pages 65–70, 1988.

[Wlo95] Wloka, M. Lag in multiprocessor vr. *Presence, 4, 1*, pages 50–63, 1995.

[Wut99] Wuthrich, C.A. An analysis and a model of 3d interaction methods and devices for virtual reality. *Proceedings of the Eurographics Workshop*, pages 18–29, 1999.

[Zel92] Zeltzer, D. Autonomy, interaction, presence. *Presence, Vol. 1, No. 1*, pages 127–132, 1992.

[Zyd99] Zyda, M. A brief history of shared networked virtual environments. *SIGGRAPH'99*, 1999.

Table des figures

Liste des algorithmes