Plate-forme de communication distribuée pour les Environnements Virtuels Collaboratifs 3D à fort couplage d'activité synchrone

Thèse

présentée et soutenue publiquement le 25 novembre 2003

pour l'obtention du

Doctorat de l'Université des Sciences et Technologies de Lille

(spécialité informatique)

par

Stéphane LOUIS DIT PICARD

Composition du jury

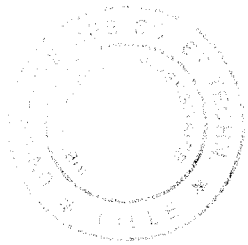
Présidente :	Laurence DUCHIEN	LIFL, Université de Lille 1
Rapporteurs :	Bruno ARNALDI	IRISA – SIAMES, Rennes
	Daniel THALMANN	EPFL – VRLab, Lausanne (Suisse)
Examinateurs :	Jean-Dominique GASCUEL	GRAVIR/IMAG – ARTIS, Grenoble
	Slim BEN ATALLAH	INRIA/IMAG – SARDES, Grenoble
Invité :	Dominique PAVY	France Télécom R&D, Lannion
Directeurs :	Christophe CHAILLOU	LIFL, Université de Lille 1
	Samuel DEGRANDE	LIFL, Université de Lille 1
	Christophe GRANSART	LIFL, Université de Lille 1

UNIVERSITÉ DES SCIENCES ET TECHNOLOGIES DE LILLE

Laboratoire d'Informatique Fondamentale de Lille — UMR 8022

U.F.R. d'I.E.E.A. – Bât. M3 – 59655 VILLENEUVE D'ASCQ CEDEX

Tél. : +33 (0)3 20 43 47 24 – Télécopie : +33 (0)3 20 43 65 66 – email : direction@lifl.fr



Remerciements

Je remercie Laurence Duchien de m'avoir fait l'honneur de présider mon jury d'examen.

Je remercie Bruno Arnaldi et Daniel Thalmann d'avoir accepté de juger ce travail et d'en être les rapporteurs.

Je remercie également Jean-Dominique Gascuel, Slim Ben Attalah ainsi que Dominique Pavy de m'avoir fait l'honneur de participer à ce jury.

Je remercie Christophe Chaillou de m'avoir ouvert les portes du bureau 212 au sortir de l'école d'ingénieurs, et de m'avoir offert la possibilité de réaliser cette thèse.

Je remercie également tout particulièrement mes deux co-directeurs de thèse, Samuel Degrande et Christophe Gransart, deux personnes qui ont toujours été présentes lorsque j'en ai eu besoin. Au cours de ces quatre dernières années, durant mon année de DEA et mes trois années de thèse, leurs compétences techniques et leurs qualités humaines m'ont beaucoup apporté et aidé dans la réalisation de cette thèse. Qu'ils trouvent ici toute ma reconnaissance et ma gratitude.

Je tiens à exprimer toute ma reconnaissance à l'équipe technique du LIFL, Bruno Boursier, Claude Dehier et Alain Fargue, ainsi qu'à l'équipe administrative, Annie Danscoine, Bérangère Doby-Dassonville, Nicole Flinois et Julienne Nzamukosha, pour leur travail afin de nous offrir les meilleures conditions de travail possibles.

La vie est faite d'expériences, celle qui vous est résumée dans les quelques pages qui suivent en est une très importante pour moi, à la fois belle et dure. Cette expérience s'accompagne de magnifiques rencontres humaines : plus que des collègues de travail, j'ai trouvé, au sein de l'équipe Graphix, de véritables amis ainsi qu'une superbe ambiance de travail (à nous faire oublier que l'on travaille). Merci donc aux membres, anciens membres de l'équipe Graphix, *spiceux* ou *spineux* : lol, fred, doug, fabrice, philippe, dual, luc, jpvdb, greg, cédric, pk, grinch, nico, gringallet, alex, mickael, juju, jérémie, sylvain, damien, carmack, manu, moby, daftpunk... Je ne sais de quoi l'avenir sera fait, mais je suis certain d'une chose : si je pars, je sais ce que je vais perdre.

Une dernière chose, si je suis en train d'écrire ces dernières lignes, c'est que la fin est proche. Cela aura été dur, mais je suis content d'y être arrivé. J'ai prononcé une phrase qui pouvait être mal interprétée (n'est ce pas fred) – *"écrire une thèse avec le style du laboratoire, c'est facile !"*. Cela en a fait rire plus d'un, maintenant je réalise pourquoi ! La rédaction de ce manuscrit, entrecoupée d'énormes moments de doute, aura été une étape dure, à mes yeux, la plus dure de ces trois années de thèse. Je pense qu'il faut le vivre pour comprendre vraiment, j'ai vu fred, et ensuite lol, rédiger mais je n'imaginais pas réellement la difficulté que cela représente. Heureusement, tous m'ont aidé à surmonter cette épreuve et m'ont motivé pour aller plus loin encore.

Un grand merci à ma famille de m'avoir soutenu tout au long de ces années et auprès de laquelle j'ai trouvé le réconfort pour mener à terme ce travail.

Merci à tous, arrêtez de manger si vite, et les poivrons, ca pique !

Table des matières

Table des figures

Introduction

1 Généralités

Depuis une dizaine d'années, les technologies informatiques ont beaucoup évolué. La puissance des micro-processeurs n'a cessé de croître[1]. Plusieurs dispositifs informatiques auparavant réservés aux professionnels se démocratisent : les ordinateurs grand public actuels sont couramment équipés d'une carte vidéo performante capable d'afficher des scènes 3D composées de plusieurs dizaines de milliers de polygones, d'un dispositif à haut débit (RNIS, Câble, Ligne Spécialisée, xDSL[2] permettant la connexion de l'ordinateur au réseau informatique mondial (Internet), d'un matériel sonore offrant une qualité de rendu inimaginable il y a quelques années.

De nombreux domaines tirent profit de ces avancées technologiques. En particulier, on assiste à l'essor des communautés virtuelles 3D sur Internet. Plusieurs utilisateurs situés aux "quatre coins" du globe se réunissent virtuellement dans un monde alternatif dans le simple but de jouer (par exemple, à des jeux comme *Quake* d'Id Software ou *Ultima Online* d'Electronic Arts), de communiquer (par exemple, avec *le* $2^{ème}$ *monde* de Canal+) ou encore de se cultiver (par exemple, en visitant des monuments aujourd'hui détruits comme la démonstration VRML 97 de *la cité Aztec de Tenochtitlán* [HW96]). Toutes ces technologies sont un formidable moyen de rapprocher des hommes, pour lesquels il aurait été difficile (voire impossible) de se rencontrer dans le monde "réel". La disponibilité de nombreux kits de développement pour la conception de telles communautés démontre l'intérêt porté à ce domaine d'activité.

Parallèlement à l'évolution du matériel informatique, la conception des logiciels a également subi de profonds changements : la notion d'*intergiciel* (traduction française du terme anglais *middleware*) a fait son apparition. Un *intergiciel* peut être comparé à une "colle logicielle" entre les systèmes d'exploitation des machines et les applications. Les problèmes d'hétérogénéité (diversité du matériel, des systèmes d'exploitation ou des langages de programmation) sont complètement cachés aux développeurs. Tout particulièrement, dans le domaine du développement d'applications réparties sur des réseaux informatiques, de nombreuses solutions intergicielles sont disponibles parmi lesquelles on peut citer la norme *Common Object Request Broker Architecture* (CORBA) de l'Object Management Group (OMG), *Java Remote Method Invocation* (RMI) de Sun Microsystems ou bien encore les technologies *Distributed Component Object Model* (DCOM) et *.NET* de Microsoft.

---

1. En 1965, G. Moore, co-fondateur de la société Intel, a annoncé que le nombre de transistors présents dans un micro-processeur doublerait tous les 18 mois. Cette loi dite "loi de Moore", énoncée il y a voila plus de 35 ans, reste à l'heure actuelle toujours valable.

2. La technologie DSL (en anglais, *Digital Subscriber Line* comme HDSL, SDSL, ADSL, RADSL et VDSL), etc) est en passe de dominer le marché et de s'imposer comme la technologie pour le haut débit domestique.

Dans l'Entreprise, les méthodes de travail ont évolué. Auparavant, le schéma de conception d'un produit pouvait être qualifié de "centralisé", i.e. toutes les personnes impliquées dans les différentes phases de la conception étaient réunies sur un même site. Des réunions fréquentes réunissant les différents acteurs étaient organisées autour de plans papiers. Aujourd'hui, de par l'évolution du modèle industriel (sous-traitance et filiation) visant à réduire les coûts et les temps de conception, le schéma de conception a évolué vers un modèle "décentralisé". Un excellent exemple de secteur industriel ayant évolué dans ce sens est le secteur de l'industrie automobile. Ainsi, un produit industriel est aujourd'hui le fruit de la collaboration entre plusieurs équipes de développement situées dans des lieux de production différents. Afin de coordonner l'évolution du projet, et de par l'impossibilité de réunir régulièrement les différents acteurs (ne serait-ce que dû aux coûts qu'une telle opération engendre et aux problèmes d'organisation), les entreprises ont besoin d'outils informatiques permettant une collaboration distante. Or, à l'heure actuelle, les outils d'*e-collaboration synchrone*[3] mis à disposition des entreprises apportent des solutions très limitées en terme de coopération. La visio-conférence se limite à mettre en relation les personnes mais n'offre pas réellement de collaboration et d'interaction entre les différents participants. De plus, le coût des équipements nécessaires (réseaux performants, appareillage multimédia) reste élevé. D'autres outils, plus grand public, comme le logiciel *NetMeeting* de Microsoft, font figures de "gadgets". Il devient nécessaire de proposer de nouveaux outils, supportant la collaboration distante, qui soient plus adaptés avec la chaîne de conception actuelle.

L'*e-collaboration synchrone* peut aujourd'hui tirer parti des outils électroniques tels que les environnements virtuels. En effet, nous constatons que la majorité des produits sont conçus numériquement sous la forme de maquettes virtuelles via des outils de Conception Assistée par Ordinateur (CAO) 3D comme par exemple CATIA de Dassault Systèmes. Il semblerait donc tout à fait naturel d'utiliser ces objets numériques en les intégrant dans un monde virtuel où les personnes interagiraient et collaboreraient, en temps réel, autour de ces maquettes virtuelles : c'est ce que l'on appelle la "revue numérique" de projet. Bien au delà du cadre de la conception d'un produit, de tels environnements virtuels dédiés à la collaboration peuvent également trouver leur place dans le cadre du service après-vente et de la maintenance, mais aussi dans d'autres domaines de la société comme par exemple celui de l'éducation avec l'apprentissage à distance. Malheureusement, les environnements virtuels couramment proposés sont des systèmes permettant d'accueillir un grand nombre d'utilisateurs (très fréquemment plusieurs milliers) et par conséquent limitant l'interaction avec les objets du monde virtuel rendant quasiment impossible toute activité coopérative avec les autres participants. De nouveaux systèmes, plus en adéquation avec l'activité de groupe, doivent être proposés.

3. En opposition à l'*e-collaboration asynchrone* où les acteurs n'interagissent pas en même temps. L'e-mail ou les forums de discussion (*newsgroup*) sont des exemples d'outils d'*e-collaboration asynchrone*.

Depuis plusieurs années, l'équipe GRAPHIX du Laboratoire d'Informatique Fondamentale de Lille (LIFL) s'attache à définir et concevoir un Environnement Virtuel Collaboratif (EVC) 3D pour le Travail Coopératif synchrone Assisté par Ordinateur (TCAO). Le projet Spin-3D, réalisé en partenariat avec France Télécom R&D (Lannion), a permis de proposer un nouveau concept d'interface tridimensionnelle pour le TCAO synchrone. Un tel projet se mène dans un contexte pluri-disciplinaire (travail coopératif, ergonomie des interfaces, synthèse d'images, techniques logicielles). Plusieurs thèses ont permis de faire évoluer le projet : la thèse de G. Saugis [Sau98] a permis de définir un modèle d'organisation de l'espace de travail virtuel basé sur une métaphore de table de réunion ; la thèse de C. Dumas [Dum99] a permis d'ajouter à l'interface de travail les mécanismes pour l'interaction avec les objets virtuels et la représentation des différents acteurs ; la thèse de P. Le Mer [LM01] a proposé un modèle d'architecture informatique pour la médiatisation des gestes de communication dans l'EVC. L'interface de travail met en relation, avec une interaction forte utilisateur-objet et utilisateur-utilisateur, un nombre réduit de personnes (environ cinq personnes) pour qu'ils réalisent l'activité coopérative. Aujourd'hui plus qu'un EVC pour le TCAO synchrone, notre objectif avec le projet Spin-3D est de proposer une nouvelle plate-forme générique permettant le développement aisé d'applications coopératives synchrones 3D. C'est dans ce cadre que s'inscrit cette thèse.

2 Objectifs

L'objectif de cette thèse est de proposer une couche de partage de données utilisée dans le cadre de la plate-forme de TCAO synchrone Spin-3D. Plusieurs utilisateurs, distants géographiquement, sont réunis au même instant dans une même place virtuelle dans le but de mener une tâche commune.

Il existe de nombreuses technologies pour la réalisation d'applications de TCAO synchrone mais on remarque qu'elles ne sont que très peu utilisées. De nombreux facteurs (leur coût, les limitations techniques comme la qualité de la vidéo, l'inadaptation à l'activité, etc) freinent leur utilisation.

Au travers du projet Spin-3D, nous souhaitons proposer une plate-forme pour le développement d'applications coopératives synchrones. Plus qu'une technologie, toutes nos propositions répondent au besoin d'utilisabilité. Fort de l'expérience acquise avec le développement du premier prototype, notre plate-forme doit rester un système simple d'accès et répondant aux besoins des concepteurs d'applications coopératives qui, le plus souvent, ne sont pas informaticiens. Ainsi, ils peuvent se concentrer sur le développement d'applications plutôt que de se soucier des problèmes

purement techniques.

Avec les applications développées au dessus de notre plate-forme, un petit groupe de personnes doit être capable de se réunir (virtuellement) et coopérer pour réaliser une tâche définie à l'avance. La plate-forme de développement doit être "utilisable" : le développement d'une application doit rester simple et les différents problèmes liés à la distribution géographique des participants doivent être cachés au concepteur d'applications coopératives. La conception d'une application doit se limiter dans la majorité des cas à la création de contenus partagés (ie. les différents documents de la réunion coopérative). Cependant, la plate-forme doit rester adaptable tant en terme d'ergonomie de l'interface, afin de répondre réellement aux besoins des utilisateurs, qu'en terme de conception, afin de s'adapter au contexte technique.

2.1 Objets partagés

Dans une application coopérative, les utilisateurs interagissent avec des objets partagés. Un objet partagé peut être défini comme étant un objet de l'environnement virtuel visible et manipulable par les différents participants présents pendant la session. Pour cela, un objet partagé expose plusieurs propriétés modifiables par tous les participants. Il existe plusieurs mécanismes permettant de partager des objets. Une première solution est d'utiliser une bibliothèque de programmation et spécifier, au chargement des objets, les différentes propriétés partagées en utilisant des appels de fonctions. À un plus haut niveau, on peut également utiliser des mécanismes purement descriptifs qui restent plus simple d'accès (notamment aux *designers* non familières avec la programmation), comme par exemple indiquer au sein de la définition des objets leurs propriétés partagées.

Pour des raisons d'utilisabilité de notre plate-forme, nous préférons proposer un système descriptif. Notre terminal de coopération est construit autour d'un navigateur VRML étendu permettant de gérer la présentation de l'interface de travail et les mécanismes d'interaction propres à Spin-3D. Les différents objets 3D présents dans l'interface de travail utilisent le langage de description VRML 97. La solution que nous souhaitons proposer doit permettre aux concepteurs (programmeurs confirmés ou novices) de créer des objets VRML 97 partagés à partir d'objets déjà existants, qu'ils auraient pu par exemple récupérer sur le web. Ces différents objets VRML 97 sont donc intégrés très facilement dans l'interface de travail.

L'état des propriétés partagées d'un objet doit être maintenu cohérent chez tous les participants présents à la réunion virtuelle afin de donner l'illusion qu'ils partagent effectivement le même monde: les différents évènements qui se produisent sur un poste doivent être transmis

aux autres postes participant à la session virtuelle en utilisant la plate-forme de communication. La couche de description que nous souhaitons proposer doit permettre de masquer au concepteur d'applications la plate-forme de communication sous-jacente, et les problèmes de gestion de la distribution des utilisateurs et de maintien de la cohérence des données partagées. De plus, un système descriptif reste, pour le concepteur d'applications, complètement indépendant de la couche de transport utilisée, ainsi la plate-forme de communication reste interchangeable de façon transparente.

Dans une application de TCAO, le mode de partage des objets évolue au cours de la session. Un utilisateur peut avoir envie de faire des modifications sur un objet, dans un espace privé. À un instant donné, le mode de partage d'un objet peut être public : l'objet est présent dans toutes les interfaces, et il est susceptible d'être manipulé par différents utilisateurs. Alors qu'à un autre instant, le mode de partage du même objet peut être privé : l'objet est uniquement présent dans l'espace privé de l'utilisateur. Le système que nous souhaitons proposer doit donc être capable de gérer de telles situations.

2.2 Plate-forme de communication

2.2.1 Communication interne

La plate-forme de communication permet à une machine de communiquer avec les autres afin d'assurer la cohérence de l'environnement virtuel 3D. D'une part, la plate-forme de communication doit offrir aux utilisateurs l'interactivité nécessaire au traitement de la tâche coopérative. En effet, lorsqu'un utilisateur manipule un objet (partagé ou pas), le système doit être réactif afin de ne pas gêner l'utilisateur dans sa manipulation et dans la compréhension de son activité. D'autre part, maintenir la cohérence des différents objets partagés entre les participants est une chose importante : le bon déroulement de l'activité coopérative en dépend. Le manque de cohérence sur les objets partagés peut complètement perturber le bon déroulement de l'activité coopérative en introduisant une incompréhension de la part des utilisateurs.

Nous distinguons trois types de données qui peuvent être échangées entre les différentes machines présentes durant la session virtuelle :

- des informations ponctuelles correspondant au contrôle de l'interface ou à des actions ponctuelles de l'utilisateur, comme par exemple, une action de clic sur un bouton,
- des flux d'informations correspondant à des évolutions continues et illimitées dans le temps,

comme par exemple le flux sonore utilisé par les participants pour communiquer,
- des flux d'informations correspondant à des évolutions continues et limitées dans le temps, comme par exemple des animations d'objets présents dans l'environnement,

Pour le premier type d'information, on a besoin d'une *cohérence forte.* Ces informations doivent être transmises de façon fiable assurant que tous les participants aient la même vue du monde virtuel 3D partagé. Pour le deuxième type de données, on peut tolérer une *cohérence relâchée* : dans un flot, des informations peuvent ne pas être toutes reçues, les informations suivantes rétabliront la consistance. Enfin, le troisième type de données nécessite à la fois une *cohérence relâchée* et une *cohérence forte* : en effet, par exemple dans le cas d'une manipulation d'objet (translation, rotation), la dernière information, donnant l'état final de la manipulation, est importante et doit être transmise de façon fiable pour assurer la cohérence du monde ; les informations qui la précèdent peuvent être transportées sur un canal non fiabilisé : elles ont pour but de renseigner les participants sur l'évolution de l'activité[4].

Pour l'utilisabilité de notre plate-forme de développement, nous devons cacher ces différents problèmes aux concepteurs d'applications. Cependant, tout en masquant aux concepteurs d'applications les différents problèmes techniques, liés notamment au maintien de la cohérence des données partagées, nous souhaitons offrir une certaine flexibilité en leurs permettant de configurer le comportement des objets partagés de l'application. Pour cela, les concepteurs d'applications doivent être capable de spécifier, au niveau de la description du partage de chaque objet, le type de l'information partagée.

2.2.2 Communication externe

Dans le but de fournir une plate-forme de développement d'applications coopératives synchrones, nous souhaitons que les concepteurs puissent développer leurs propres applications coopératives. Cependant, il n'est pas concevable de vouloir y intégrer toutes les applications : passer d'une application 2D à une application 3D n'est pas évident, les mécanismes d'interaction et de présentation ne sont pas les mêmes. De plus, les utilisateurs sont habitués au fonctionnement de l'application existante. Inversement, il n'est pas concevable de vouloir intégrer la plate-forme de collaboration dans les applications existantes. Il nous faut donc plutôt proposer un mécanisme permettant d'interfacer des applications existantes (que nous qualifierons d'"externes") avec la plate-forme.

4. Cela n'est vrai que si la trajectoire n'est pas importante pour la compréhension de l'activité par les utilisateurs.

Le mécanisme de partage proposé doit tenir compte du fait que l'utilisateur est capable de modifier un objet soit par une manipulation directe, via l'interface de l'EVC 3D, soit par une manipulation indirecte, via une application externe.

3 Organisation du mémoire

Cette thèse est organisée en deux parties. Une première partie, consacrée à l'état de l'art, est composée de quatre chapitres ayant pour but de montrer les différents domaines couverts et les différentes contraintes auxquelles doit répondre une plate-forme de communication d'un EVC 3D pour le TCAO :

- le premier chapitre présente les Environnements Virtuels 3D ainsi que les principes de base du TCAO. Cette étude permet de mettre en évidence l'utilisation des environnements virtuels 3D pour la réalisation d'un logiciel de TCAO, permettant ainsi de dégager les concept d'Environnement Virtuel Collaboratif 3D. Nous mettrons en évidence les contraintes, du point de vue de l'utilisateur, que doit respecter un EVC 3D afin que la coopération soit possible,
- le chapitre 2 présente les différents aspects techniques liés à la distribution géographique des participants et permettant d'assurer la cohérence du monde virtuel 3D partagé. Ces différents aspects sont le plus souvent mis en avant pour la réalisation de communautés virtuelles 3D regroupant un grand nombre d'utilisateurs. Nous discuterons de l'importance de ces différents aspects dans le cadre d'un système de TCAO où le nombre de participants est réduit afin de permettre une coopération efficace entre eux,
- pour offrir un système simple à utiliser, nous devons masquer les aspects techniques afin que les concepteurs d'applications coopératives se concentrent sur la partie applicative (ie. le contenu). Le chapitre 3 présente différents systèmes d'EVC 3D de haut niveau que l'on peut trouver sur le web. En nous appuyant sur les concepts de base du TCAO identifiés dans le premier chapitre, nous montrerons qu'au niveau du concepteur du terminal de travail, les systèmes existants sont dédiés à la création de communautés virtuelles 3D ou de simulations distribuées et ne permettent pas à leurs utilisateurs de réellement coopérer,
- le chapitre 4 présente le système d'EVC 3D développé par l'équipe Graphix du LIFL. Nous montrerons les spécificités de l'interface de travail permettant de répondre aux différentes contraintes du TCAO. Nous présentons les différentes solutions techniques proposées par un premier prototype du terminal de travail. Nous montrerons que des techniques modernes comme VRML 97 et CORBA peuvent faciliter le développement du terminal de travail.

La deuxième partie est consacrée à la présentation des mécanismes de communication et de partage intégrés à notre plate-forme de développement d'applications coopératives. Dans un premier temps, nous présentons les mécanismes de communication utilisés pour maintenir la cohérence des objets dupliqués dans chaque terminal. Afin de proposer un système simple d'utilisation, nous proposons, au dessus de la couche de communication, des mécanismes d'abstraction de haut niveau permettant la conception d'objets 3D partagés utilisables dans l'interface et le développement d'applications coopératives complexes. Cette partie est divisée en trois chapitres :

- le chapitre 5 présente les mécanismes de communication utilisés pour que les différents terminaux communiquent entre eux afin de maintenir la cohérence du monde partagé. Nous détaillons les deux canaux de communication offrant des niveaux de cohérence différents, un canal à cohérence forte et un canal à cohérence faible. À plus haut niveau, nous avons défini deux services permettant de gérer l'environnement virtuel 3D,
- le chapitre 6 présente une abstraction de haut niveau des mécanismes de communication proposés afin de faciliter le travail du concepteur d'applications coopératives. Nous proposons d'utiliser le langage descriptif VRML 97 pour concevoir les différents objets 3D présents dans l'interface de travail. Nous intégrons également la description du partage de chaque objet 3D dans le même fichier VRML 97 que celui contenant la description de la géométrie et de l'interaction,
- le chapitre précédent se propose uniquement de pouvoir intégrer des objets 3D partagés dans l'interface. Pour réaliser des applications coopératives plus complexes, nous souhaitons offrir d'autres mécanismes de développement. Des applications ne se prêtent pas au mécanisme d'interaction 3D proposé par notre interface de travail. Nous proposons donc de pouvoir connecter ces applications sur notre terminal de travail afin qu'elles en manipulent le contenu. Le chapitre 7 présente l'interface de programmation inspirée de l'External Authoring Interface VRML 97 à laquelle nous avons intégré de nouveaux services.

Première partie

État de l'art

Chapitre 1

Les EVC 3D : un support pour le TCAO

Sommaire

Dans ce chapitre, nous présentons les Environnements Virtuels (3D) ainsi que leurs applications. Nous présentons également le Travail Coopératif Assisté par Ordinateur (TCAO) ainsi que les différents concepts de base qu'un logiciel de TCAO (collecticiel) doit s'efforcer de respecter pour offrir à ses utilisateurs de bonnes conditions de travail en terme de communication et de coopération.

1.1 Environnement Virtuel 3D

En 1986, Jaron Lanier "invente" le terme de réalité virtuelle. Il n'existe pas de définition universelle du terme de réalité virtuelle. D'un point de vue fonctionnel, le terme réalité virtuelle désigne des mondes alternatifs synthétisés par des ordinateurs, et avec lesquels nous interagissons par diverses technologies. Le mode de présentation utilisé par la réalité virtuelle n'est pas forcément la représentation 3D. Le monde synthétisé peut très bien être une représentation informatique de notre réalité, lorsqu'il s'agit par exemple de simulations (simulateurs de vol, etc), ou bien il peut également être complètement artificiel et n'être qu'une construction mentale de la part de ses utilisateurs, comme par exemple dans le cas des MUD (abréviation du terme anglais *Multi User Dungeon*)[5].

Depuis quelques années, de par les améliorations des matériels informatiques (augmentation de la puissance de calcul des micro-processeurs, développement des technologies 3D), la tendance générale est d'utiliser la 3D comme mode de visualisation de la réalité virtuelle. Le terme de réalité virtuelle est souvent associé à la synthèse d'images 3D temps réel, introduisant ainsi une certaine confusion. Nous préférons employer le terme de "réalité virtuelle 3D" lorsque la technique de visualisation est la synthèse d'images 3D.

Depuis son apparition en 1986 dans le langage, le terme générique de réalité virtuelle a donné naissance à une multitude d'autres termes, tel que *environnement virtuel 3D* (EV 3D)[6]. Un environnement virtuel 3D est un système de réalité virtuelle 3D mettant l'accent sur un monde artificiel disposant d'une composante sociale permettant de régir les relations entre les différentes personnes présentes dans le système. Un environnement virtuel 3D peut être *mono-utilisateur* : un seul utilisateur est projeté dans le monde virtuel, les autres acteurs du système sont contrôlés par l'ordinateur, c'est par exemple le cas dans de nombreux jeux vidéo. Le développement des

5. Les MUD sont des versions informatisées, textuelles et multi-utilisateurs des jeux de rôles tels que "Donjons et Dragons".

6. En anglais, Virtual Environment (VE).

réseaux informatiques a contribué à l'essor des environnements virtuels 3D *multi-utilisateurs* : plusieurs utilisateurs sont projetés dans un même lieu virtuel, dans lequel ils peuvent se voir, communiquer, voire même travailler ou jouer ensemble.

Environnement Virtuel Collaboratif 3D

Depuis quelques années, le terme *environnement virtuel 3D collaboratif* (EVC 3D)[7] a fait son apparition marquant une nuance de taille dans l'ensemble des environnements virtuels 3D. En effet, la notion d'EVC 3D introduit une notion de coopération entre les différents utilisateurs. Dans un EVC, certaines règles sont à respecter afin de permettre la communication et la coopération entre les utilisateurs pour, par exemple, rendre la sensation de présence des autres, renseigner les utilisateurs sur l'activité des autres.

Il est à noter une certaine confusion dans l'utilisation du terme "collaborer". Le terme "coopérer" semble être plus adapté pour caractériser le type d'activité couvert par notre projet, dans le sens où le terme de coopération inclut une certaine notion de manipulation (opération). À l'heure actuelle, la plupart des EVC 3D disponibles ne proposent pas réellement aux utilisateurs de coopérer en proposant un mécanisme orienté autour de la tâche.

Les environnements virtuels 3D coopératifs sont complexes à développer tant ils requièrent de compétences issues de domaines divers. Certains s'intéressent à des aspects synthèse d'images temps réel et interface homme-machine (IHM) permettant la représentation du monde virtuel et l'interaction avec celui-ci. D'autres s'intéressent à l'aspect multi-utilisateurs des environnements virtuels 3D. Sur ce dernier aspect, tout d'abord d'un point de vue technique, certains s'intéressent aux technologies réseaux à mettre en oeuvre pour supporter le monde virtuel et la distribution des utilisateurs. Ensuite, au delà de ces aspects purement techniques, des facteurs humains sont également à prendre en compte lors de la conception d'un environnement virtuel : en effet, certains s'intéressent aux relations et à la communication inter-humains dans l'environnement virtuel. Cela relève du Travail Coopératif Assisté par Ordinateur (TCAO) que nous présenterons par la suite (cf. 1.2, page 28). Nous pensons que la technologie mise en œuvre doit être liée aux besoins des utilisateurs. Cependant dans beaucoup de propositions, les aspects humains passent après les choix technologiques. Une telle approche, dont la philosophie est que *les utilisateurs s'adapteront plus ou moins facilement*, est vouée à l'échec. Ceci est d'autant plus vrai pour un environnement virtuel 3D collaboratif où les utilisateurs sont amenés à travailler ensemble. Au contraire, nous pensons que les choix technologiques doivent répondre aux besoins des utilisateurs identifiés par le TCAO : c'est la philosophie de la communauté du TCAO et de l'IHM où les aspects liés à

7. En anglais, *Collaborative Virtual Environment* (CVE) ou bien encore *Networked Collaborative Virtual Environment* (NCVE).

l'utilisateur sont plus importants que les aspects technologiques.

1.1.1 Domaines d'application des Environnements Virtuels 3D

La liste d'applications proposée n'est pas exhaustive mais elle permet de dépeindre quelques secteurs d'activité dans lesquels les environnements virtuels jouent un rôle important.

1.1.1.1 La simulation

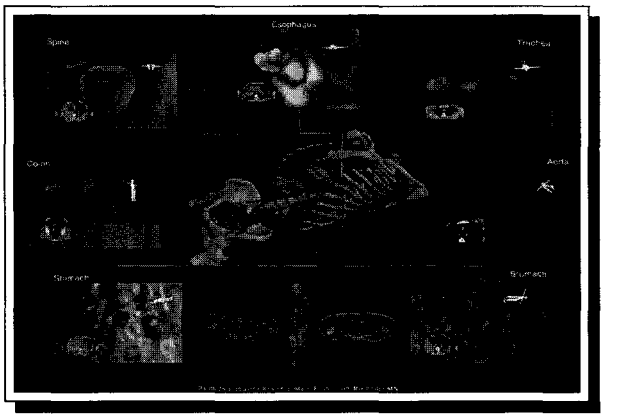
La simulation consiste à calculer, à l'aide d'un ordinateur, un phénomène réel en tenant compte des lois qui régissent la nature. On peut considérer que c'est le domaine des Environnements Virtuels 3D qui requiert le plus en terme de moyens technologiques (puissance de calcul, synthèse d'images 3D temps réel, dispositifs mécaniques pour l'interaction homme-machine et la rétro-action du système, dispositifs de rendu, etc). Le plus souvent, les différentes avancées technologiques issues de la simulation sont réutilisées dans d'autres domaines (notamment celui des divertissements).

Très souvent, les simulateurs sont utilisés pour l'assistance en opération ou la formation à un moindre coût, parmi lesquels on peut citer les domaines médicaux et militaires.

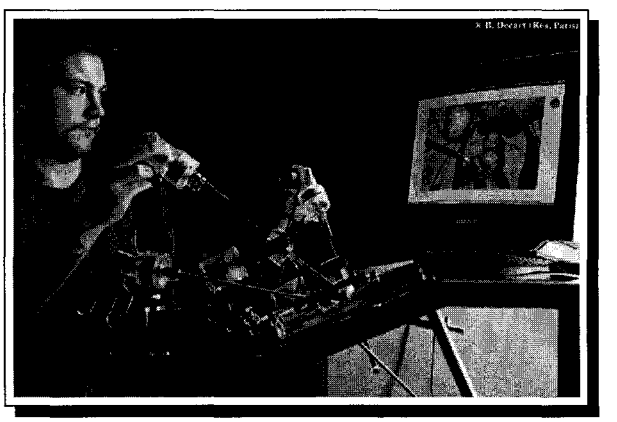
Comme le montre la figure 1.1, l'utilisation des environnements virtuels 3D en chirurgie peut constituer un excellent support pour des activités telles que l'enseignement de l'anatomie et des pathologies aux étudiants, l'entraînement aux procédures chirurgicales des nouveaux chirurgiens, la "téléchirurgie" [8] ou bien encore la planification d'opérations complexes.

Les industries militaires et aérospatiales ont compris depuis longtemps l'importance de la simulation et de l'entraînement. En particulier, les simulateurs de vol militaires ou civils sont devenus des instruments d'utilisation courante pour la formation et l'entraînement des pilotes (voir figure 1.2). Cependant le coût de telles installations se révèle très élevé de par les technologies qu'elles requièrent (affichage, mécanisme à verrins, etc). D'autres simulateurs sont développés pour l'entraînement des troupes au sol comme par exemple les plate-formes de l'armée américaine SIMNET [CDG+93] et NPSNET [MZP+94].

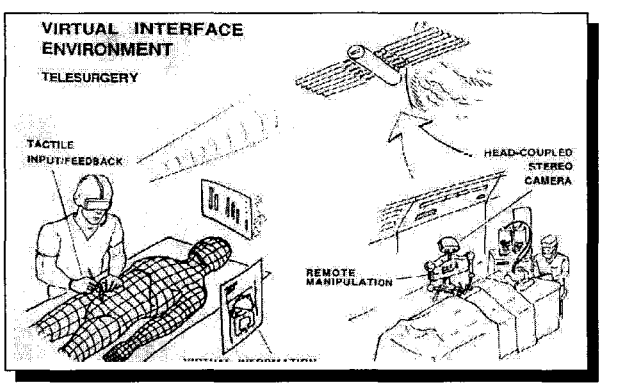
8. Opération chirurgicale à distance.



(a) Logiciel d'exploration de différentes parties du corps humain à partir d'un modèle 3D (Biomedical Imaging Resource, Mayo Foundation)



(b) Logiciel d'entraînement à la chirurgie non invasive (SPIC, LIFL)



(c) Le principe de la téléchirurgie (NASA)

FIG. 1.1 – *Utilisation des Environnements Virtuels 3D dans le domaine médical*

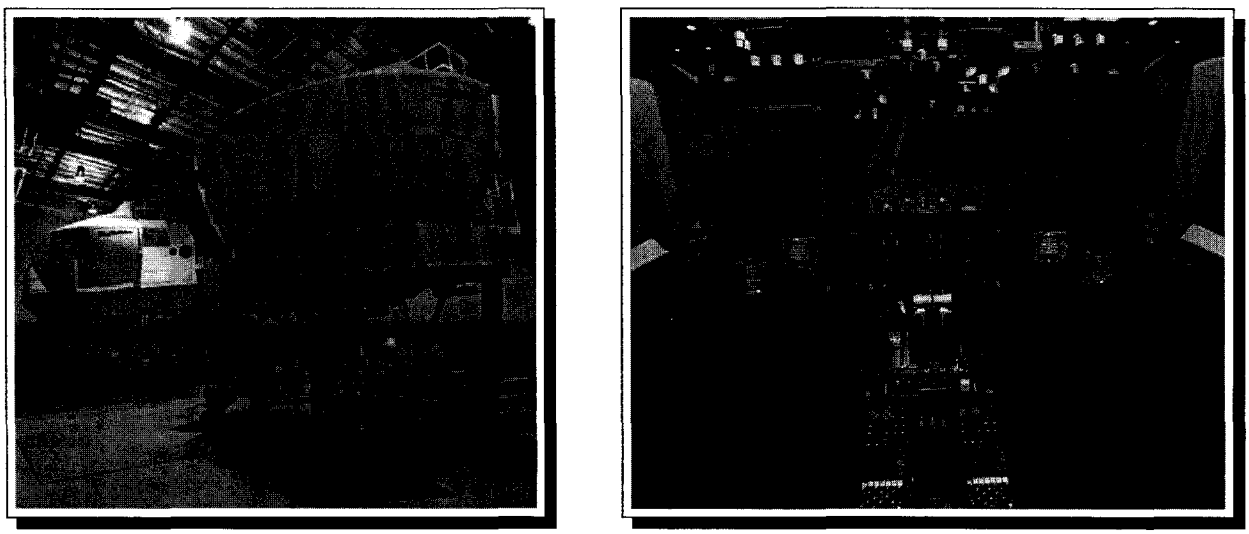


FIG. 1.2 – *Simulateur de vol d'avion civil (Boeing)*

1.1.1.2 Divertissements

Les jeux vidéo sont de grands consommateurs des différentes réalisations techniques d'environnements virtuels 3D. De fait, ils en sont même devenus l'un des moteurs en influençant l'évolution des technologies d'affichage 3D. Il y a quelques années les mondes virtuels 3D intégrés dans les jeux vidéo correspondaient à des environnements virtuels 3D mono-utilisateur. Aujourd'hui avec la démocratisation des matériels informatiques performants, la généralisation de la mise en réseau des ordinateurs et la vulgarisation d'Internet, les jeux vidéo correspondent à des environnements virtuels 3D multi-utilisateurs : ils permettent à plusieurs utilisateurs de co-exister dans le monde virtuel 3D, de communiquer et d'interagir. Les jeux vidéo du type FPS (abréviation du terme anglais *First Person Shooter*), comme le jeu Quake d'ID Software sont de bons exemples de l'utilisation des environnements virtuels 3D multi-utilisateurs dans l'industrie du jeux vidéo.

Les outils de communication temps réel 3D sur Internet (*chat 3D*), comme par exemple *le* $2^{ème}$ *monde* de Cryo et Canal+ ou le site web de *Cybertown*[9] (cf. figure 1.3), se multiplient et constituent eux aussi d'excellents exemples de communautés virtuelles 3D. Ces logiciels associent un mode texte à une vue 3D et permettent à leurs utilisateurs de dialoguer entre eux (très souvent de façon textuelle, et parfois en utilisant la voix). Cependant, ils offrent très peu de fonctionnalités permettant une réelle coopération entre les utilisateurs du système. La vue 3D est essentiellement utilisée comme un support ludique à la communication et permet de renforcer le sentiment de présence des utilisateurs.

9. `http://www.cybertown.com`

FIG. 1.3 – *Une capture d'écran du site de Cybertown, en haut la vue 3D du monde et en bas la fenêtre de chat*

1.1.1.3 Travailler autrement

Les environnements virtuels peuvent introduire de nouveaux concepts dans les méthodes de travail, comme l'a fait il y a quelques années la messagerie électronique. Au sein des entreprises, les technologies numériques ont pris une place importante dans la conception des produits, par exemple l'utilisation des techniques de synthèse d'image 3D pour le maquettage. Nous voyons apparaître des outils basés sur les techniques issues des environnements virtuels 3D et utilisés pour de la revue numérique de projet, comme par exemple les différents outils proposés par Tornado Technologies Inc[10]. La demande en outils de collaboration synchrone se fait de plus en forte et correspond à des besoins de la part des entreprises : les réunions à distance, le rapprochement des bureaux d'études distants, l'assistance technique des utilisateurs, etc. Il est donc tout à fait naturel de s'intéresser à l'utilisation des environnements virtuels 3D comme outil de collaboration.

Dans d'autres domaines, comme par exemple celui de l'enseignement, les environnements virtuels (3D) devrait également jouer un rôle important dans les prochaines années : même, ils commencent déjà à faire leur apparition dans les techniques d'apprentissage. En proposant de nouvelles techniques aux élèves permettant, sans doute, de mieux retenir les connaissances et d'augmenter leurs motivations, les environnements virtuels peuvent être un formidable vecteur pour l'apprentissage. Toutes les techniques d'apprentissage peuvent bénéficier de l'apport des environnements virtuels, en particulier dans le domaine de l'enseignement à distance.

1.1.2 Techniques informatiques pour la conception d'Environnements Virtuels 3D

Plusieurs techniques sont nécessaires pour appréhender le développement d'un système de réalité virtuelle 3D comme les environnements virtuels 3D. Tout d'abord, il faut être capable de fournir un navigateur ainsi que les moyens pour visualiser et interagir avec le monde virtuel 3D. Ensuite, il faut produire des contenus utilisables par le navigateur, ie. le concepteur de monde virtuel doit être capable de décrire des données 3D.

10. `http://www.tornadoviz.com`

1.1.2.1 Techniques d'approche pour la réalisation du navigateur

Tout d'abord pour créer le navigateur de monde virtuel, il existe plusieurs techniques. Les différentes possibilités se déclinent sur une échelle menant du langage "naturel" (approche "haut niveau") au langage "machine" (approche "bas niveau"). Ainsi, n'importe quelle personne, à son niveau (du *designer* au programmeur), est alors capable de développer un environnement virtuel 3D.

1.1.2.1.1 Approche "haut niveau"

Le développement d'un environnement virtuel 3D nécessite la connaissance de nombreux outils informatiques tels que les langages orientés objet, la synthèse d'images temps réel, les réseaux, les langages de modélisation. Dès lors, il peut être intéressant de proposer, aux personnes possédant des connaissances limitées dans ces domaines, des *outils Auteur* permettant de créer des environnements virtuels 3D très simplement.

L'auteur de l'environnement virtuel 3D compose des scènes 3D à l'aide d'un éditeur graphique dédié. Il spécifie certains éléments 3D dynamiques en utilisant les différents menus de l'interface de développement, comme dans *World-Up* de Sense8[11], *Virtools Dev* de Virtools Corporation[12] ou *Atmosphere* d'Adobe[13]. L'environnement virtuel est visualisé au travers d'un programme fournit avec l'éditeur graphique ; le programme de visualisation peut être une application exécutable indépendante (*standalone*), ou encore un plug-in s'intégrant à un navigateur *web*. On obtient à peu près le même résultat avec un fichier VRML, contenant des scripts définissant le comportement des objets, visualisé, par exemple, avec *Cosmoplayer* de Silicon Graphics ou *Cortona* de Parallel Graphics. Le système VRML sera présenté dans la suite de ce chapitre, ainsi que dans l'annexe A (page 224).

Ces différents systèmes "prêts à l'emploi" ont tous le défaut de n'être pas ou peu extensibles : l'ajout de nouvelles fonctionnalités ne se fait pas de façon triviale. Il faut donc être sûr que l'application voulue ne sort pas du cadre de leurs possibilités.

---

11. `http://www.sense8.com`

12. `http://www.virtools.com`

13. `http://www.adobe.com/products/atmosphere`

1.1.2.1.2 Approche "intermédiaire"

On trouve ensuite différentes boîtes à outils logicielles comprenant un format de description de la scène 3D et possédant des méthodes pour manipuler les différents objets de la scène. Les plus connues sont *Performer* de Silicon Graphics Inc., *WorldToolkit* de Sense8 et plus récemment *Java3D* de Sun Microsystem. Ils sont composés d'une bibliothèque offrant de nombreuses fonctionnalités avancées permettant de manipuler la scène 3D tout en masquant l'accès aux données 3D. Sur le même niveau, on trouve également l'EAI (*External Authoring Interface*) définie dans la norme VRML 97 et qui permet de manipuler et modifier la scène contenue dans le navigateur VRML 97 en utilisant un langage de programmation (comme *Java* par exemple).

Là encore, comme précédemment avec les outils Auteur, certaines de ces boîtes à outils sont extensibles mais il n'est pas aisé de proposer de nouvelles fonctionnalités, notamment en ce qui concerne l'aspect multi-utilisateurs. Il est à noter que certaines boites à outils proposent un support du multi-utilisateurs, comme par exemple Avango [Tra99], VR Juggler [JBBCN98], NPSNET [MZP+94], Virtus [Saa99]. Cependant, ils sont réservés à un public averti, essentiellement des programmeurs.

1.1.2.1.3 Approche "bas niveau"

Une dernière approche est celle des bibliothèques de programmation bas-niveau. Il s'agit d'un ensemble de fonctions élémentaires permettant le rendu d'images 3D en temps réel. Le calcul des images s'effectue en utilisant différents algorithmes [FvDFH90][WW92] : rendu projectif, z-buffer, ombrage de Gouraud, plaquage de textures, transparence, etc. Aujourd'hui, ces différents algorithmes sont directement pris en charge (i.e câblés) par le processeur des différentes cartes graphiques (GPU, *Graphic Processor Unit*) disponibles sur le marché (NVidia, ATI, Intergraph). Le programmeur crée sa scène 3D en utilisant les différentes fonctions élémentaires fournies par la bibliothèque (positionnement d'un point, spécification de ses attributs graphiques comme sa texture ou bien sa couleur, etc). Contrairement à l'approche boîte à outils, la gestion des données 3D de la scène, les outils permettant leurs manipulations et leurs modifications, sont entièrement à la charge du programmeur. La bibliothèque 3D se charge uniquement de la partie calcul de l'image 3D. Bien souvent, l'approche bibliothèque de programmation 3D est plus simple que les boîtes à outils mais nécessite beaucoup plus de programmation pour arriver à des résultats équivalents. Néanmoins cette approche permet une plus grande souplesse et les limitations sont uniquement d'ordre techniques (liées à la machine), et non plus liées à la bibliothèque comme

dans le cas des boîtes à outils.

Les bibliothèques de programmation 3D les plus répandues sont *OpenGL* (librairie issue des recherches de SGI et disponible sur de nombreuses plate-formes) et *DirectX* (librairie propriétaire de Microsoft, disponible uniquement sur la plate-forme Windows). Il existe dans la communauté du logiciel libre d'autres bibliothèques comme par exemple SDL (*Simple Direct Media Layer*)[14].

1.1.2.2 Description des données 3D

Dans un système d'environnement virtuel 3D, le concepteur doit être capable de décrire les différents objets présents dans le monde 3D. Nous allons maintenant présenter les différentes solutions qui s'offrent à lui.

1.1.2.2.1 Formats propriétaires

Il existe de nombreux formats de description de données 3D, propriétaires ou du domaine public. Chaque logiciel de modélisation, comme par exemple *3DSMax* de Discreet, *Softimage .XSI* ou *Maya* d'Alias|Wavefront, et *Autocad* d'Autodesk, possède son propre format de fichier. Des outils, comme par exemple *Deep Exploration* de Right|Hemisphere, permettent facilement de passer d'un format à un autre. Ces différents formats sont beaucoup utilisés en modélisation professionnelle mais très peu sur Internet pour l'échange de données 3D (excepté les formats `.3ds` de *3DSMax* `.dxf` d'*Autocad*).

1.1.2.2.2 Le format VRML

Sur le web, nous trouvons beaucoup plus facilement des formats libres de droits. Notamment, le format VRML [BPP95][HW96][CBM97] (normalisé ISO en 1997) qui depuis quelques années s'est imposé comme le format de description de données 3D. La plupart des modeleurs 3D sont capables d'exporter au format VRML les données 3D qu'ils produisent. Pour visualiser une scène VRML (1.0 ou 2.0) et pouvoir explorer le monde virtuel, il suffit d'un navigateur web auquel on a ajouté un plug-in capable d'interpréter la scène et de l'afficher dans le navigateur web même. Sur

14. `http://www.libsdl.org`

le marché, on trouve différents navigateurs VRML, on peut entre autres citer CosmoPlayer de SGI, Blaxxun Contact de Blaxxun, et Cortona de ParallelGraphics. Le format ascii utilisé pour le codage d'une scène 3D VRML est à la fois un avantage pour la facilité d'édition, mais aussi un inconvénient pour la taille des fichiers générés. Cependant, les différents navigateurs VRML du marché supportent le format VRML compressé (fichier VRML ayant subi une compression *zip*).

Le langage VRML permet de décrire sous la forme d'un arbre la géométrie 3D du monde. L'arbre est composé de *nœuds* : on peut distinguer deux types de nœuds, premièrement des nœuds permettant de créer la hiérarchie 3D (ce que l'on appelle des *grouping node*, comme par exemple les nœuds `Group` ou `Transform`), deuxièmement, des nœuds permettant de spécifier le rendu (sonore, visuel, géométrique). L'arbre créé à l'aide de ces différents nœuds est couramment appelé "graphe de scène" [HW96]. Il est possible de proposer de nouveaux nœuds en utilisant le mécanisme de `PROTO` (décrit à l'intérieur du fichier) et d'`EXTERNPROTO` (décrit à l'extérieur du fichier) disponibles dans le langage VRML 97. Ces deux mécanismes permettent de décrire l'interface d'un nouveau nœud afin de pouvoir l'intégrer dans le graphe de scène. Il est à noter que, dans le cas d'un `EXTERNPROTO`, si le navigateur est incapable de localiser sa définition, le nœud sera considéré comme vide.

Chaque nœud est composé de *champs* disponibles en entrée, sortie, ou en entrée/sortie (`EventIn`, `EventOut`, `ExposedField`). La dynamique du monde 3D est obtenue en connectant des `EventOut` à des `EventIn` (de même type) en utilisant le mécanisme de `ROUTE`. L'évolution du monde produit des changements de valeurs (par exemple à l'aide de timers) qui sont répercutés grâce à l'exposition des `EventOut`. Tous les `EventIn` connectés à ces `EventOut` sont alors prévenus du changement de valeur, ce qui déclenche alors une cascade d'évènements. Le concepteur peut également ajouter au graphe de scène des noeuds (appelés `Sensors` dans la norme) permettant de gérer l'interaction avec l'utilisateur. Ces noeuds particuliers permettent de capter ce qui se passe dans le monde 3D. Un noeud `Sensor` possède plusieurs champs (en sortie uniquement) de façon à réagir aux actions de l'utilisateur et à les propager dans le graphe de scène en utilisant le mécanisme de `ROUTE`. Pour gérer des comportements plus complexes, le langage VRML 97 permet d'inclure dans le graphe de scène des scripts. Tout comme les autres nœuds natifs du langage, le nœud `Script` expose différentes propriétés (définies par le concepteur) qui peuvent être connectées à d'autres nœuds en utilisant le mécanisme de `ROUTE`. Un nœud `Script` contient du code compilé Java ou bien du code interprété écrit en ECMAScript (dérivé du langage JavaScript). Les scripts interagissent avec le contenu VRML 97 en utilisant la Browser Script Interface (BSI) [HW96] : cette interface permet, entre autre, aux scripts de retrouver des nœuds nommés explicitement (ie. possédant un DEF), de modifier les `ROUTES` existantes ou encore de modifier le contenu du navigateur VRML 97 (tout le graphe de scène ou en partie).

Pour développer des comportements plus complexes, la norme VRML 97 propose l'External Authoring Interface (EAI). L'EAI est une API permettant à des programmes externes (i.e. situés hors du navigateur VRML 97) d'interagir avec le contenu du navigateur VRML 97. La spécification de l'EAI ne fait pas partie intégrante de la norme, en effet, c'est seulement une annexe. Cela signifie que les développeurs de navigateur ont le choix de l'implémenter ou non. Néanmoins, la majorité des navigateurs VRML 97 proposent cette fonctionnalité. L'EAI définit un ensemble de services de haut niveau permettant à l'application externe d'accéder aux nœuds, à leurs propriétés et aux évènements de la scène VRML. La spécification de l'EAI décrit le comportement de ces services mais reste neutre quand à leur implémentation : aucun langage cible (ie. le langage utilisé pour le développement d'applications externes) n'est spécifié, une entière liberté est laissée aux développeurs. Néanmoins dans la pratique, tous les navigateurs VRML 97 supportent Java comme langage pour l'implémentation des applications externes. À l'heure actuelle, au vue des différentes implémentations, l'EAI est uniquement utilisée pour la communication entre une scène VRML et des applications externes (*applets* Java) incluses dans la même page HTML et s'exécutant dans le contexte du navigateur web.

Le lecteur trouvera en annexe une présentation détaillée des langages VRML 1.0, VRML 97 et de l'EAI (cf. annexe A, page 224).

1.1.2.2.3 Les futurs formats sur Internet

Dans les prochaines années, on devrait également trouver sur le web de nouveaux formats. Le format X3D (*eXtensible 3D*) [X3D] est en passe de devenir le successeur des formats VRML 1.0 et VRML 97. La principale différence avec VRML est l'utilisation du langage XML (*eXtensible Markup Language*) pour la description de la scène 3D. La norme X3D devrait bientôt faire l'objet d'une soumission auprès de l'organisation internationale de normalisation ISO[15].

Pour la description des données multimédia 3D, le format MPEG-4 [MP4] reprend également les grands principes du format VRML 97. La principale différence est l'utilisation d'un codage binaire de la scène afin de réduire la taille des fichiers générés. Avec la technique de compression utilisée, la taille d'un fichier MPEG-4 binaire est inférieure à celle du même fichier MPEG-4 au format ascii (équivalent à VRML 97) zippé. Cependant, contrairement à X3D dont la norme est complètement disponible, le format MPEG-4 ne peut pas être considéré comme étant du domaine public : la plupart des informations sont uniquement disponibles pour les membres du

15. La norme est à l'heure actuelle terminée, et est entrée dans une phase de relecture.

consortium MPEG-4. De plus, son utilisation est soumise à licence[16]. Cependant comme nous le verrons par la suite (cf. 3.1.2, page 91), MPEG-4 est bien plus qu'un simple format de fichier mais propose une architecture logicielle pour le développement d'applications multimédia (audio, vidéo, et 3D).

1.1.2.3 Périphériques d'entrée/sortie

Une autre étape essentielle lors de la conception d'un environnement virtuel est le choix des dispositifs de visualisation (périphériques de sortie) et des dispositifs d'interaction avec le système (périphériques d'entrée). Cela sort du cadre de cette thèse. Cependant, nous effectuons une énumération très succincte des différents dispositifs d'entrée/sortie. Le lecteur pourra se reporter à [Dum99] pour une étude plus détaillée.

Les dispositifs de visualisation

La technologie d'écran la plus répandue est celle des tubes cathodiques qui équipent nos moniteurs d'ordinateurs et nos téléviseurs. Il y a quelques années, des systèmes de visualisation plus spécialisés sont apparus. Ils permettent d'afficher des objets en trois dimensions, et non plus à plat (*technique de vision stéréo*), de superposer de l'information virtuelle à de l'information réelle (*technique de réalité augmentée*). Ces différentes techniques de visualisation sont dites non-immersives : l'utilisateur garde contact avec le monde réel.

Au contraire, d'autres techniques permettent de plonger l'utilisateur dans le monde virtuel 3D : ces techniques sont dites immersives. On peut entre autre citer les techniques à base de casques d'immersion et les *Caves* où l'utilisateur est entouré d'écrans géants, créant ainsi artificiellement son immersion.

Les dispositifs d'interaction

L'environnement virtuel doit également fournir à ses utilisateurs un moyen pour interagir avec le système. Dans le domaine de l'informatique bureautique, la souris s'est imposée comme le dispositif d'interaction indispensable. Cependant, ce périphérique à deux degrés de liberté montre ses limites lors de son utilisation dans des environnements 3D. Des périphériques d'entrée plus originaux et plus adaptés à la troisième dimension voient le jour pour le grand public comme par exemple la *SpaceMouse* de Logicad 3D ou le *FreeD* de Pegasus. D'autres, plus onéreux

16. Les licences d'utilisation sont disponibles sur le site `http://www.mpegla.com`.

et plus perfectionnés comme par exemple le *Phantom* de Sensable, sont réservés à un usage professionnel. Des dispositifs d'interaction sont plus ou moins contraignant pour l'utilisateur et peuvent le gêner dans son activité. C'est notamment le cas des différents dispositifs à retour d'effort de type "exosquelette".

1.2 Le Travail Coopératif Assisté par Ordinateur

Le Travail Coopératif Assisté par Ordinateur (TCAO), traduction du terme anglais *Computer Supported Collaborative Work* (CSCW), est un domaine de recherche apparu pour la première fois en 1986 [KG86]. La pluridisciplinarité de ce domaine de recherche rend difficile sa définition. Néanmoins, nous retiendrons la définition axée sur l'informatique de Ellis [EGR91], qui a été traduite par Karsenty de la façon suivante [Kar94] : "*Système informatique qui assiste un groupe de personnes engagées dans une tâche commune (ou but commun) et qui fournit une interface à un environnement partagé*". Quant à lui, le terme de "Collecticiel", traduction du terme anglais *groupware* [Lév90], est couramment utilisé pour parler des logiciels de TCAO.

L'essor du collecticiel est très lié à l'évolution des matériels informatiques (développement des ordinateurs personnels/portables, des réseaux informatiques, performance du matériel audio et graphique). L'omniprésence de l'ordinateur dans notre société (son rôle principal étant de supporter l'activité humaine) et la nature collaborative de l'activité humaine expliquent l'intérêt porté aux collecticiels [Wex93]. Les conférences scientifiques qui relèvent des domaines liés au collecticiel sont nombreuses. De plus, le collecticiel fait l'objet de nombreuses publications dans des magazines grand public.

1.2.1 Classification des collecticiels

La classification du collecticiel en fonction des dimensions spatiales et temporelles [EGR91] est une des plus répandues (voir la figure 1.4).

	même lieu	*lieux différents*
même moment	salle de réunion	téléconférence, chat
moments différents	tableau d'affichage	email, forum de discussion

FIG. 1.4 – *Classification espace/temps des collecticiels*

La première dimension, *l'espace*, correspond à la distance séparant les utilisateurs : soit les membres de la réunion se trouvent dans la même pièce, soit les membres de la réunion sont situés dans des lieux éloignés (des bâtiments, des pays ou bien même des continents différents).

La deuxième dimension, *le temps*, correspond au type d'interaction entre les utilisateurs. L'interaction entre les membres du groupe peut s'effectuer en même temps, en direct : toute action de l'un des membres du groupe est immédiatement transmise aux autres membres. Il s'agit de collaboration *synchrone*. Au contraire, les membres du groupes peuvent agir en temps différé : les actions sont espacées dans le temps et un membre du groupe est capable d'observer la somme des interventions qui se sont effectuées avant son arrivée. Il s'agit alors de collaboration *asynchrone*.

1.2.2 Le TCAO synchrone

En utilisant la classification précédente, nous nous intéressons dans ce travail aux collecticiels permettant de mettre en relation directe des personnes ou des groupes de personnes : c'est le domaine du TCAO synchrone. Et plus particulièrement, nous nous intéressons aux collecticiels permettant à des utilisateurs physiquement éloignés de pouvoir coopérer afin de réaliser un objectif particulier.

La médiatisation d'une activité coopérative synchrone impose de respecter des contraintes humaines sous peine d'échec. Au delà de la technique, le travail coopératif synchrone impose aux collecticiels de respecter certains concepts fondamentaux afin que les tâches coopératives se déroulent correctement et que le collecticiel soit accepté par les utilisateurs. Nous allons maintenant passer en revue ces différents concepts sur lesquels doit s'appuyer tout collecticiel. Le lecteur trouvera une étude plus détaillé sur ces différents concepts dans [Sau98] et [Dum99].

1.2.2.1 La téléprésence

Le terme d'avatars est utilisé pour désigner la représentation quelconque d'un utilisateur. Les différents interlocuteurs doivent être vus dans l'interface sous forme de vignette fixe (image) ou animée (vidéo), ou sous forme 3D. C'est ce que l'on appelle la téléprésence. Des travaux ont mis en avant les qualités d'une représentation sous la forme d'avatar 3D piloté par les mouvements de l'utilisateur (yeux, gestes, expressions faciales) [BF93][BGB+95].

Avec son avatar, chaque intervenant doit également posséder un moyen de faire de la dési-

gnation à distance. Cet artefact doit être visible dans les interfaces des autres interlocuteurs : c'est ce que l'on appelle un télépointeur [PA94].

1.2.2.2 L'activité de groupe

La parole, les gestes (volontaires ou réflexes), le regard et les expressions du visage sont autant de vecteurs de communication nécessaires pour mener à bien une tâche collaborative. Ils permettent de renseigner un utilisateur sur ce que font, ce que pensent les autres. Le travail à plusieurs nécessite de connaître le point de vue et les actions des autres interlocuteurs. Un collecticiel doit être capable de rendre ces différentes informations permettant aux différents participants d'une activité coopérative de comprendre les actions des autres et pouvoir interagir.

1.2.2.3 Le concept de bureau virtuel

Le concept de regrouper spatialement les documents nécessaires à une tâche particulière afin de recentrer l'attention des participants est issue du projet *Rooms* de Xerox [HC86]. À chaque tâche est associé un bureau virtuel qui est dédié au traitement de celle-ci. Les participants peuvent naviguer entre les différents bureaux virtuels, un seul étant affiché à la fois. Les différentes personnes devant travailler sur une même tâche sont donc présentées dans le même bureau virtuel : ceci permet de montrer les interactions entre personnes et documents, ou entre les personnes uniquement.

1.2.2.4 L'activité centrée autour de la tâche

C'est un concept qui va de paire avec le concept de bureau virtuel : il vise à déterminer comment les documents ainsi que les outils nécessaires à la tâche doivent être organisés dans le bureau virtuel. Les participants doivent se concentrer autour de plusieurs documents partagés. Le bureau virtuel contient uniquement l'essentiel afin de faciliter le traitement de la tâche et de supprimer les tâches parasites. Il est recommandé de placer les documents et outils nécessaires à la tâche de façon centrale afin de renforcer l'impression de collaboration.

Il n'existe pas une unique représentation pour le bureau virtuel : il en existe une par participant. Bien que la tâche soit commune, chaque utilisateur a sa propre représentation mentale pour la réaliser et l'environnement doit être capable de s'adapter. C'est pourquoi, chaque participant

doit être libre d'organiser les documents nécessaires à la tâche dans sa propre représentation du bureau virtuel comme il le souhaite. Le collecticiel doit être capable de supporter des organisations différentes d'un utilisateur à un autre tout en maintenant la cohérence des documents partagés.

1.2.2.5 Perception et soutien de la conscience mutuelle

Le TCAO exige, qu'à tout moment, chaque participant soit capable de savoir ce qui se produit, ce qui change, où et par qui. C'est ce que l'on peut appeler le concept du "tout visible".

1.2.2.6 L'animation interactive temps réel

C'est en fait la fusion de deux concepts. Tout d'abord, les données nécessaires à un utilisateur à un instant donné doivent être parfaitement visibles, les autres peuvent être affichées de façon dégradée donnant juste une idée globale de leur contenu. Ensuite, l'interface présentée à l'utilisateur doit être "fluide". Il faut éviter les ruptures brusques dans ce qui est présenté afin de minimiser la charge cognitive de l'utilisateur. Certains systèmes graphiques 2D décomposent la réduction d'une fenêtre à l'état d'icone en une animation linéaire, au lieu de fermer brutalement la fenêtre et de créer un nouvel icone.

L'animation interactive temps réel permet également d'avoir un système réactif nécessaire pour supporter l'activité synchrone entre les différents participants : les différentes animations, réponses aux actions des participants, doivent être immédiates et linéaires.

1.3 De la vidéo aux environnements virtuels 3D collaboratifs

De nombreux collecticiels utilisent la vidéo comme support à la collaboration. L'utilisation de la vidéo n'est pas sans poser de problèmes tant du point de vue technique que du point de vue de l'utilisateur. Les dernières avancées techniques, comme celles réalisées dans le domaine de la synthèse d'images 3D, offrent de nouvelles perspectives pour développer de nouveaux collecticiels.

1.3.1 Les outils audiovisuels

De nombreux collecticiels disponibles à l'heure actuelle sur le marché utilisent des techniques et du matériel classiques, essentiellement basées sur la vidéo : caméras, télévisions ou projections sur grand écran. Ce sont le plus souvent des systèmes de visioconférence : ils sont équipés de caméras pour filmer les différents interlocuteurs, et de micros pour capter la voix. Ils permettent de mettre en relation plusieurs personnes ou plusieurs groupes de personnes.

Dans une simple réunion à deux personnes (un face-à-face), une caméra correctement positionnée devant chaque utilisateur permet d'utiliser correctement le regard. Au contraire, quand le nombre d'interlocuteurs augmente, il devient impossible d'utiliser la caméra pour regarder quelqu'un ou quelque chose en particulier. Le support vidéo sert uniquement à faire de la téléprésence.

Des outils, plus aboutis et permettant de résoudre ces problèmes liés à l'utilisation de la vidéo, ont été développés : on peut entre autres citer les *Hydra Units* [Bux92], *Magic* [OMIM94] ou le *mur de téléprésence* récemment proposé par France Télécom R&D [17].

1.3.2 Les outils audiovisuels informatisés

Les différentes techniques audiovisuelles ont été transposées à l'informatique. Il s'agit des applications informatiques de travail collaboratif utilisant une caméra vidéo (webcam), principalement pour faire de la téléprésence. Ces outils proposent en plus du support vidéo de partager des applications informatiques classiques (logiciels de traitement de texte, tableurs, etc) pour travailler à plusieurs sur un même document (cf. figure 1.5). On parle alors de *desktop conferencing* : ce sont des réunions à plusieurs, chacun étant dans son bureau devant sa machine. Un grand défaut de ce type de logiciel est la latence introduit dans l'interaction par le fait de manipuler à distance : l'application s'exécute sur un unique poste et les utilisateurs interagissent avec l'application à distance réduisant alors l'interactivité de l'application partagée.

Le produit le plus connu dans ce domaine est le logiciel NetMeeting de Microsoft [18] (disponible gratuitement). De nombreux autres produits en reprennent les concepts, comme par exemple les

17. `http://www.rd.francetelecom.fr/fr/galerie/navig_mur.htm`

18. `http://www.microsoft.com/windows/netmeeting`

produits de Placeware[19] ou bien encore les produits de collaboration proposés par les différents éditeurs de CAO comme SmartTeam de Dassault Systèmes[20].

1.3.3 Les environnements virtuels 3D collaboratifs

L'utilisation de la vidéo comme support aux collecticiels montre ses limites : elle ne permet pas aux intervenants d'avoir naturellement conscience de la présence et de l'activité des autres. De plus, elle nécessite des moyens techniques importants et coûteux (en terme de matériels et de réseau à haut débit). Si l'on imagine difficilement d'autres alernatives dans le cas de la visioconférence entre plusieurs groupes de personnes, d'autres solutions sont envisageables dans le cas du *desktop conferencing*, comme par exemple les environnements virtuels 3D collaboratifs.

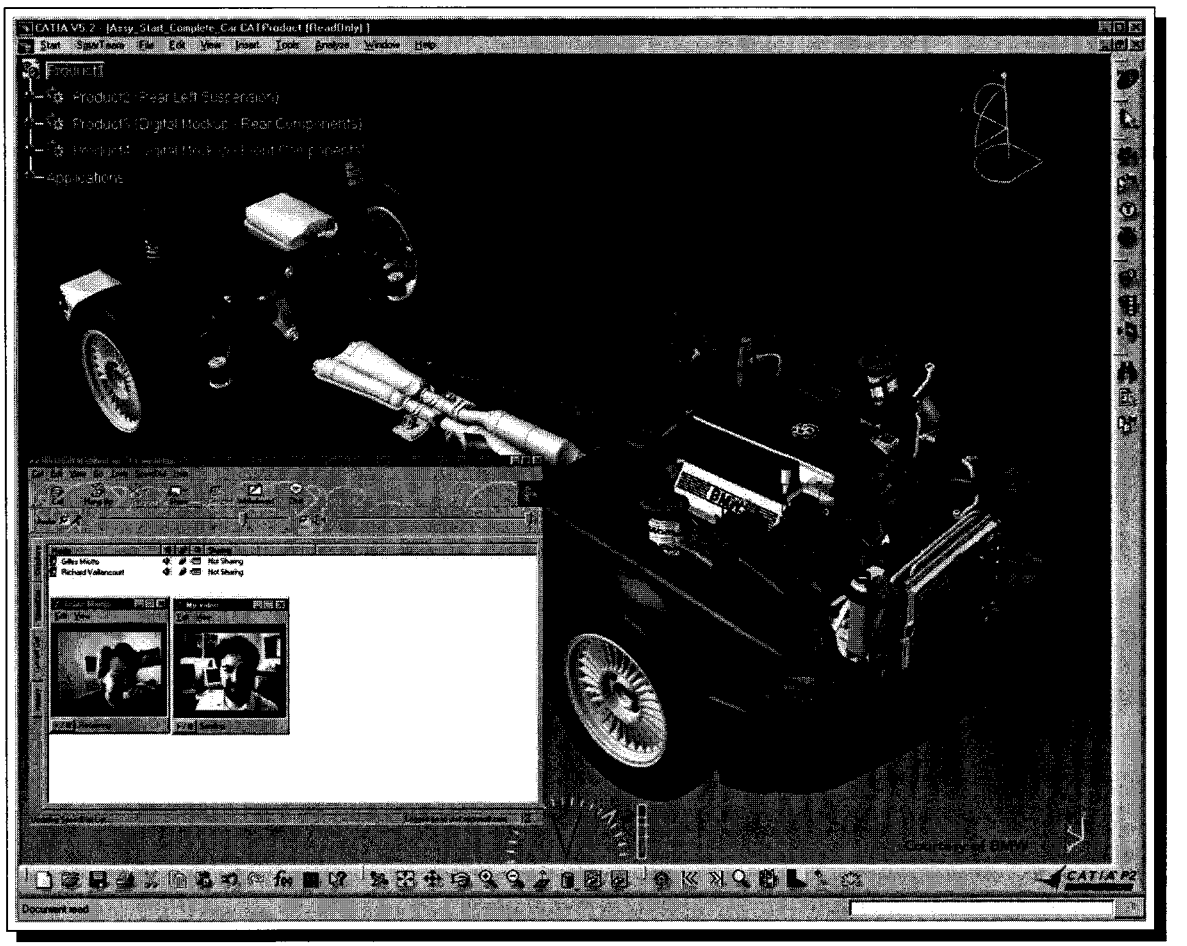
Les progrès effectués ces dix dernières années par le matériel graphique 3D et le développement des réseaux informatiques ont permis la montée en puissance du concept d'EVC 3D. Les EVC 3D fournissent un excellent support au TCAO synchrone. Un environnement virtuel 3D collaboratif est un système d'environnement virtuel permettant à plusieurs utilisateurs, distants géographiquement, d'évoluer dans un monde 3D commun. Les utilisateurs sont souvent représentés par des avatars (2D ou 3D) au travers desquels ils peuvent interagir avec le monde et ses différents objets et communiquer les uns avec les autres.

Chaque utilisateur est devant son poste sur lequel s'exécute le terminal de visualisation du monde virtuel. Pour des raisons de rapidité d'affichage, chaque poste dispose en local d'une copie (complète ou partielle) de la description du monde virtuel. L'utilisateur se déplace dans le monde virtuel, interagit avec les objets et les utilisateurs présents. Les différentes actions d'un utilisateur sont transmises aux autres utilisateurs afin de maintenir cohérent l'état du monde partagé, donnant ainsi l'illusion d'être et d'évoluer dans un même monde.

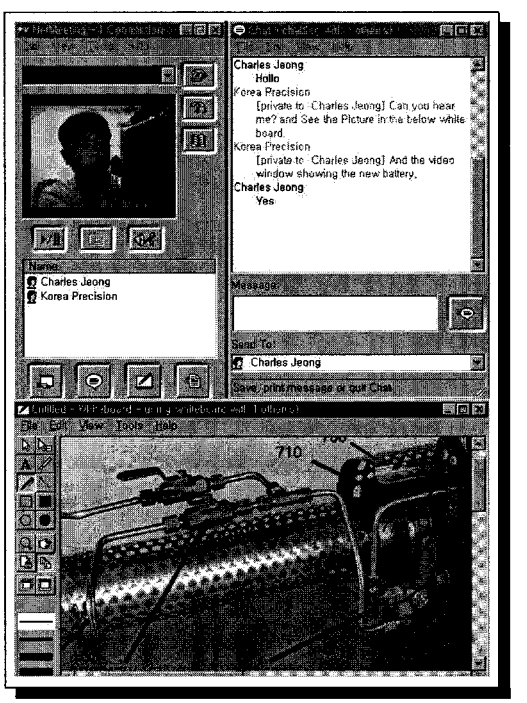
Nous allons maintenant présenter les différents domaines de recherche concernant l'aspect utilisateur des EVC 3D. Les aspects technologiques seront abordés dans le chapitre suivant (cf. 2.1, page 41).

19. `http://www.placeware.com`

20. `http://www.smarteam.com`



(a) Partage d'application



(b) Tableau blanc

FIG. 1.5 – *Outils audiovisuels informatisés*

1.3.3.1 Rendu 3D temps réel

Le rendu 3D temps réel, un élément clé d'un EVC 3D, ne représente plus vraiment un enjeu dans le développement des EVC 3D. En effet, les cartes graphiques 3D actuelles, dont les performances, déjà élevées, ne cessent de croître à la sortie de chaque nouveau modèle, permettent d'obtenir une excellente qualité d'image. Elles se programment soit via des API de haut niveau, comme Performer ou Java3D, soit via des API bas niveau comme OpenGL ou DirectX.

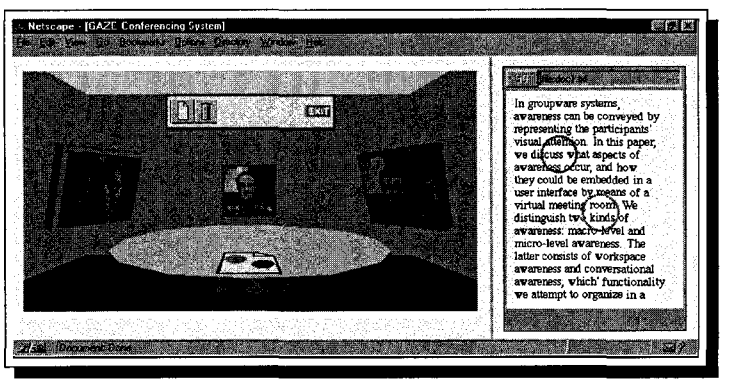
Du point de vue l'utilisateur, le TCAO 3D ne demande pas de réalisme au niveau des graphismes 3D. Le niveau de représentation 3D doit être suffisant pour permettre aux utilisateurs de comprendre les différents objets qu'ils manipulent sans pour autant que le niveau de détails soit élevé.

1.3.3.2 Représentation des utilisateurs et compréhension de l'activité

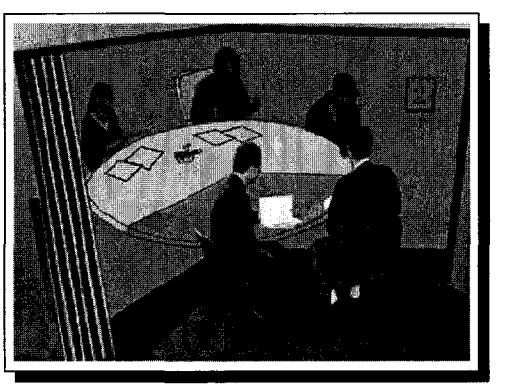
La représentation des utilisateurs détermine comment un utilisateur perçoit les autres dans le monde virtuel. C'est un vecteur de communication entre les utilisateurs du système. En effet, c'est au travers de la représentation des autres qu'un utilisateur doit naturellement avoir conscience de la présence et de l'activité des autres. Cet aspect est primordial dans la conception d'un EVC 3D puisque c'est un facteur qui va conditionner l'utilisation de l'EVC 3D. La représentation des utilisateurs dans un environnement virtuel 3D est un domaine de recherche à part entière.

Certains projets utilisent une approche hybride mêlant vidéo et environnement virtuel 3D, comme par exemple dans [Ver99][SK01] (cf. figure 1.6). Le monde est synthétique et les différents intervenants sont présentés sous la forme de vignettes vidéo ou bien encore d'avatars 3D sur lesquels on plaque une texture vidéo. Cependant, là encore, nous sommes confrontés aux problèmes classiques de l'utilisation de la vidéo, comme par exemple celui de la restitution du contact visuel, auquel nous ajoutons des problèmes d'ordres technologiques. En effet, de telles approches nécessitent beaucoup de ressources en terme de calcul et de matériels : captures d'images temps réel, détourage de visage temps réel, *texture mapping* de vidéo temps réel.

D'autres approches utilisent une représentation purement synthétique : les utilisateurs sont représentés sous la forme d'avatars 3D. Dans la littérature on trouve plusieurs types d'avatars 3D, de la simple représentation en utilisant des cubes comme dans [GB95a], à des représentations plus proches de la réalité en utilisant des corps articulés (clonage réaliste) [CH93a][CNT+97]. L'utilisation des avatars 3D permet d'améliorer la téléprésence entre les interactants renforçant



(a) Le système de vidéoconférence GAZE [Ver99]



(b) Concept de vidéoconférence immersive [SK01]

FIG. 1.6 – *Utilisation de la vidéo dans un environnement virtuel 3D*

ainsi le sentiment de collaboration. Cependant, des problèmes technologiques se posent pour atteindre un niveau de réalisme acceptable, comme par exemple le rendu facial réaliste (cheveux, visage ressemblant), l'animation faciale, le comportement autonome.

La grande majorité des systèmes d'EVC 3D offrent la possibilité à leurs utilisateurs de dialoguer de façon textuelle et vocale. Cependant, la communication naturelle est bien plus riche : les expressions faciales, les mouvements des lèvres et les gestes jouent également un rôle important dans la communication humaine. L'utilisation des avatars est un excellent vecteur de représentation de la communication naturelle. Mais, un compromis est à trouver pour ne représenter que ce qui est réellement nécessaire aux utilisateurs pour comprendre l'activité.

1.3.3.3 Interaction

L'interaction est un problème majeur dans une interface de TCAO. Les utilisateurs doivent être capables d'agir, de la façon la plus simple, sur les différents documents. Pour agir dans un environnement 3D, il existe plusieurs possibilités, certaines basées sur l'utilisation de nouveaux périphériques d'interaction 3D alors que d'autres continuent à utiliser des périphériques 2D. La transposition naturelle de l'utilisation courante des deux mains dans les tâches réelles incite à utiliser une interaction à deux mains.

Il parait intéressant de proposer une interaction 3D permettant de se rapprocher d'une interaction naturelle. Il faut essayer de faire en sorte que l'utilisateur reproduise dans l'interface certains comportements du réel réduisant ainsi la phase d'apprentissage.

Plusieurs indices visuels statiques ou dynamiques permettent à l'utilisateur de comprendre l'interaction. Il faut cependant donner uniquement les informations de retour nécessaire à la compréhension de la tâche afin de ne pas surcharger l'interface.

1.4 Conclusion

Dans ce chapitre, nous avons présenté les environnements virtuels 3D et les différentes techniques informatiques permettant leur conception. Nous avons également présenté le TCAO synchrone et les différents concepts à respecter afin de permettre aux utilisateurs de collaborer dans les meilleures conditions. À l'heure actuelle, les outils de TCAO disponibles se restreignant aux outils de visioconférence informatisés montrent leurs limites lorsqu'il s'agit de mener une activité

coopérative. Nous pensons que les prochains outils de TCAO peuvent tirer profit de la réalité virtuelle 3D. Tout particulièrement, les environnements virtuels 3D peuvent être un excellent support au TCAO synchrone. C'est dans ce but que l'équipe GRAPHIX du LIFL propose une nouvelle interface pour le travail coopératif tirant partie de la troisième dimension.

Nous avons abordé les différentes techniques couramment utilisées pour implémenter la "partie visible" (description 3D, périphériques d'entrée/sortie, etc) d'un logiciel d'environnement virtuel 3D et qui peuvent être utilisées pour développer un EVC 3D. Dans ce chapitre, nous avons également présenté les EVC 3D sous un aspect utilisateur. Nous avons vu les différentes contraintes humaines qu'impose la médiatisation d'une activité coopérative synchrone. Lors du développement d'un EVC 3D, ces différentes contraintes sont à respecter afin de proposer un outil utilisable et permettant à ses utilisateurs de travailler dans les meilleures conditions. Les différents choix technologiques dans un EVC 3D doivent répondre aux besoins des utilisateurs.

Dans le chapitre suivant, nous allons aborder les différents aspects techniques liés aux communications réseau permettant de supporter l'activité collaborative entre des utilisateurs distants géographiquement, c'est ce que nous appelons la plate-forme de communication. Ensuite dans un autre chapitre, nous présenterons différents systèmes d'EVC 3D susceptibles d'être utilisés pour le développement de notre plate-forme.

Chapitre 2

Aspects technologiques des communications dans un EVC 3D

Sommaire

Dans ce chapitre, nous présentons les différents aspects techniques permettant de supporter la distribution géographique des utilisateurs d'un EVC 3D. Dans un environnement virtuel 3D multi-utilisateurs, la plate-forme de communication a une place centrale puisqu'elle permet de faire transiter de l'information entre les différents participants distants géographiquement. Pour donner l'illusion du monde virtuel partagé et pour assurer la cohérence du monde, les différents évènements qui se produisent sur un poste doivent être transmis aux autres postes participant à la session virtuelle. Nous allons présenter les différents aspects que comportent la plate-forme de communication d'un EVC 3D, ainsi que les différentes façons de la concevoir.

Dans un premier temps, nous introduirons différents concepts que l'on retrouve dans la majeure partie des plate-formes de communication existantes et qui permettent de supporter les communautés virtuelles que l'on trouve sur Internet. Nous identifierons certains besoins spécifiques au TCAO permettant aux utilisateurs de réaliser une tâche coopérative. Dans un second temps, nous présentons les différentes façons de concevoir la plate-forme de communication : architecture, mécanisme de dialogue entre les machines.

2.1 Technologies des EVC 3D

2.1.1 Scalabilité

La transmission des évènements locaux aux autres instances de l'EVC 3D peut engendrer un énorme trafic réseau : le nombre de messages possibles dans un monde où n utilisateurs évoluent en même temps est de l'ordre de n^2. Pour supporter la scalabilité[21], le trafic réseau devrait idéalement rester à peu près constant ou croître linéairement avec le nombre d'utilisateurs.

Il existe plusieurs techniques permettant de réduire le trafic réseau et d'augmenter la scalabilité du système. Un point clé de la scalabilité est le partitionnement du monde virtuel [Saa99]. Il consiste à partitionner le monde virtuel en plusieurs régions (également appelées *zones*, *rooms*, *cellules* ou encore *locales*) [Fun95][MZP+94][WAB+96][BWA96] permettant ainsi de filtrer les informations envoyées à chaque utilisateur. À partir d'un graphe de visibilité de cellules, le système est capable de fournir uniquement les informations susceptibles d'intéresser un utilisateur

21. Le terme "scalabilité" est un abus de langage et est une francisation du terme anglais *scalability*, mais n'existe pas dans la langue française. Ce terme représente la capacité d'un système à supporter une augmentation de ses contraintes, nous pouvons le traduire par l'expression "*passage à l'échelle*". Par exemple, un système est dit "scalable" en terme de nombre d'utilisateurs si ses performances de fonctionnement sont aussi bonnes avec une dizaine d'utilisateurs connectés qu'avec une centaine.

donné dans une cellule donnée, ie. les informations provenant des utilisateurs présents dans la même cellule ou des sous-cellules (cellules visibles depuis la cellule courante de l'utilisateur). Le partitionnement de l'espace virtuel est intéressant dans le cadre d'un espace virtuel important réunissant un très grand nombre d'utilisateurs (on parle alors d'environnement virtuel massivement multi-utilisateurs).

Le modèle spatial d'interaction (*spatial model of interaction*) introduit par Benford dans [BF93] et mis en œuvre dans DIVE [CH93a][Hag97] et MASSIVE [GB95a][GB95b] vise à réduire la charge réseau en fonction du centre d'intérêt de l'utilisateur. Dans la majorité des cas, en utilisant la notion d'auras un utilisateur n'a pas besoin de recevoir les informations de tous les autres utilisateurs. Partant du fait qu'un utilisateur n'interagit à un instant donné qu'avec un nombre limité d'utilisateurs, l'idée est d'envoyer à cet utilisateur uniquement les informations qui peuvent potentiellement l'intéresser. Par exemple, prenons le cas de deux utilisateurs très éloignés l'un de l'autre, il n'y a alors pas de contact visuel direct entre ces deux utilisateurs. C'est pourquoi, il n'est pas nécessaire qu'ils s'échangent des informations les concernant tant qu'ils ne sont pas plus proche.

Nous verrons dans la suite de ce chapitre que d'autres facteurs, comme le degré de cohérence choisi et l'architecture de la plate-forme de communication, ont une influence directe sur la scalabilité.

Bilan

La scalabilité est un critère important dans les EVC 3D comme les communautés virtuelles 3D que l'on trouve sur Internet. C'est ce qui détermine le nombre d'utilisateurs que le système est capable d'accepter en même temps. Cependant, nous souhaitons proposer un système où les utilisateurs peuvent opérer, ensemble et à un grain fin, sur des documents. Pour cela, comme nous le verrons dans le chapitre suivant, nous nous plaçons dans le contexte de réunion de petits groupes. Ainsi, dans le cadre de notre activité, la scalabilité n'est pas un enjeu primordial. De plus, nous verrons également dans le chapitre suivant que l'organisation de l'interface de notre système de TCAO ne permet d'accueillir qu'un nombre réduit de participants. Pour la réalisation de la tâche coopérative, les participants sont virtuellement placés dans une pièce unique et sont en interaction directe et constante les uns avec les autres.

2.1.2 Distribution des données

La distribution des données s'attache à déterminer la façon dont la base de données décrivant le monde virtuel est organisée et dont elle est mise à la disposition des utilisateurs. [Ban96] propose la classification suivante :

- *Base de données centralisée*, elle est stockée à un seul endroit, sur un serveur qui est en charge de gérer les requêtes provenant des clients,
- *Base de données dupliquée à cohérence faible*, chaque participant dispose sur sa machine d'une copie de la base de données (copie complète ou partielle), les différents clients s'échangent à intervalles réguliers des messages de mise à jour permettant de conserver la cohérence,
- *Base de données dupliquée à cohérence forte*, c'est une base de données dupliquée mais synchronisée à tout instant,
- *Bases de données distribuées*, elle est partitionnée sur plusieurs serveurs.

Une base de données centralisée est une solution très simple à mettre en oeuvre mais uniquement viable pour les environnements virtuels basés texte comme les MUD [MZ97]: les clients très minimalistes accèdent au travers du réseau à la base de données du monde virtuel. Au contraire, dans le cadre des mondes virtuels 3D, les clients sont beaucoup plus évolués et doivent être capable d'effectuer le rendu du monde 3D en temps réel pour assurer l'interactivité du système. On entend par *temps réel* la génération (calcul) des images à une fréquence d'au moins 10 images par seconde, la limite basse acceptable pour considérer que l'animation est "confortable" à l'œil étant de 25 images par seconde. Comme mentionné dans [MF98], une approche centralisée n'est pas envisageable avec un monde virtuel 3D : il n'est pas concevable que chaque participant accède à la description du monde 3D via le réseau pour effectuer le rendu de chaque image. D'une part cela générerait un trafic réseau bien trop important, et d'autre part les accès réseaux introduisent un temps de réponse non négligeable et incompatible avec une fréquence d'affichage d'au moins 10 images par seconde. Dans un EVC 3D, il est préférable d'adopter l'approche base de données dupliquée : chaque participant dispose en local sur sa machine d'une copie de la base de données décrivant le monde 3D. Le fait de distribuer les données du monde introduit de nouveaux concepts comme le maintien de la cohérence, la gestion des accès concurrents dont nous parlerons par la suite.

Dans un EVC 3D à grande-échelle réunissant beaucoup d'utilisateurs, comme par exemple une ville ou un champs de bataille, la quantité de mémoire nécessaire pour stocker entièrement la description du monde 3D est beaucoup trop importante. C'est pourquoi, chaque client ne

possède qu'une copie partielle de cette description. Le système est capable de lui fournir, à la volée, les différentes parties qui lui sont nécessaires pour continuer sa progression dans le monde. Des algorithmes de prédiction doivent être mis en oeuvre pour que les opérations de chargement dynamique soient transparentes pour l'utilisateur [MPB03]. Dans des EVC 3D de très petite taille, comme par exemple ceux constitués d'une unique pièce, il n'est pas nécessaire de mettre en place un tel service : chaque participant dispose en local d'une copie complète de la description du monde virtuel.

Bilan

Une approche centralisée pour la base de données ne permet pas d'obtenir l'interactivité nécessaire à un système de TCAO synchrone : en effet, les modifications s'effectuent sur le serveur au travers du réseau introduisant ainsi de la latence dans les actions de l'utilisateur. Les approches à base de données dupliquée permettent au système d'être réactif : chaque utilisateur interagit sur les objets locaux et transmet les modifications aux copies distantes. Dans un environnement de petite taille, on peut considérer que chaque participant dispose d'une copie complète de la base de données.

Dans une interface de TCAO, par respect pour l'activité de l'utilisateur et la représentation mentale de l'activité propre à chaque utilisateur, il est préférable de laisser chaque utilisateur organiser son interface de travail comme il le souhaite. Ainsi, contrairement aux communautés virtuelles 3D où il n'existe qu'un graphe de scène unique, dans un système de TCAO, chaque utilisateur dispose de son propre graphe de scène constitué par l'assemblage hiérarchique des objets contenus dans la base de données dupliquée.

2.1.3 Cohérence du monde 3D

Dans un EVC, la base de données du monde virtuel partagé est susceptible d'être modifiée par chaque utilisateur. Comme nous venons de le voir, chaque utilisateur dispose en local d'une copie de la base de données du monde virtuel 3D. Afin de ne pas perturber les utilisateurs dans leurs activités dans l'EVC 3D, le système doit assurer la cohérence du monde partagé, ie. à un instant donné, la base de données du monde virtuel 3D doit être dans le même état chez tous les utilisateurs.

On distingue plusieurs niveaux de maintien de la cohérence : la *cohérence forte* consiste à assurer qu'à chaque instant tous les utilisateurs ont la base de données du monde virtuel partagé

dans le même état, ie. toutes les actions de chaque utilisateur doivent être transmises immédiatement aux autres utilisateurs en utilisant un canal de communication fiable (comme par exemple TCP/IP). Les conflits entre les participants doivent être interdits ou bien des mécanismes doivent être mis en oeuvre pour résoudre ces conflits. La *cohérence relâchée* permet de réduire la charge réseau en retardant l'envoi des actions de l'utilisateur dans le temps. Toutes les mises à jour sur la base de données ne sont pas transmises, seules quelques actions de l'utilisateur sont transmises à distance. De plus les communications réseaux se font en utilisant un protocole non fiable (comme par exemple UDP/IP).

Un des aspects cruciaux dans le maintien de la cohérence est le nombre de personnes impliquées dans le processus et le degré de cohérence nécessaire (cela dépend bien sûr du type d'application et des différents algorithmes utilisés). Le but est de réduire le nombre d'utilisateurs sur lesquels s'applique le processus de maintien de la cohérence afin de diminuer la complexité du problème. La notion de cohérence se trouve donc très liée à la notion de scalabilité. En effet, les techniques de partitionnement, permettant de réduire le nombre d'utilisateurs qui entrent directement en interaction avec un utilisateur donné, ont un rôle important à jouer dans le processus de maintien de la cohérence.

Les applications de simulation à grande-échelle, comme NPSNET [MZP+94], préfèrent utiliser une cohérence relâchée afin de réduire les communications réseaux, des algorithmes de *dead-reckoning* permettant d'extrapoler les positions des entités en fonction de leur dernière position, de leur vitesse et du temps écoulé depuis la dernière réception d'informations. Au contraire, les applications de TCAO, comme par exemple DIVE [CH93a], nécessitent une cohérence forte afin de permettre aux utilisateurs de coopérer dans de bonnes conditions. Le manque de cohérence peut entraîner des incompréhensions nuisibles à l'activité coopérative. Cependant, une cohérence forte peut très souvent entrer en conflit avec le besoin d'interactivité. L'utilisation d'un protocole fiable comme TCP/IP plutôt qu'un protocole non fiable comme UDP/IP entraîne un surcoût. Ainsi, pour conserver l'interactivité du système de TCAO, il est nécessaire de relâcher la cohérence pour certaines actions de l'utilisateur.

Pour cela, dans [KH00], Kessler propose d'utiliser plusieurs canaux de communication en classant les différents messages échangés entre les participants de la façon suivante :

- les message de mises à jour : un canal de communication permet d'échanger, en utilisant un protocole non fiabilisé (comme par exemple UDP/IP), des informations entre les machines pour maintenir une cohérence relâchée des données partagées. Des techniques comme le *dead reckoning* permettent d'éviter d'envoyer un message à chaque mise à jour. Ce canal est souvent assimilé à un flux de données,

- les messages de commandes et d'évènements : un canal de communication permet aux participants de s'échanger, en utilisant un protocole fiabilisé (comme TCP/IP), des informations ponctuelles (par opposition à la notion de flux). Les messages "commande" impliquent une réponse de la part des machines distantes, au contraire des messages "évènement" qui correspondent uniquement à l'envoi de messages de contrôle.

Bilan

Dans une application de TCAO, il est nécessaire d'avoir une cohérence forte. Les différentes valeurs partagées de la base de données doivent être maintenues cohérentes. Cependant, une cohérence forte peut très souvent entrer en conflit avec le besoin d'interactivité. L'utilisation d'un protocole fiable, comme TCP/IP, plutôt qu'un protocole non fiable, comme UDP/IP, entraîne un surcoût. Il faut donc être capable de relâcher la cohérence pour certaines actions comme par exemple dans la plupart des manipulations d'objets. En effet, dans de telles situations, le résultat final de l'action est plus important que les modifications intermédiaires. Cela sous-entend que les valeurs sont transmises en valeur absolue et non en valeur relative.

2.1.4 Persistance

Le nombre de participants dans un EVC 3D évolue au cours de la session. Des participants quittent la session, alors que d'autres sont susceptibles d'entrer dans le monde virtuel 3D après que des modifications sur l'état initial du monde aient eu lieu. Le système doit être capable de fournir aux nouveaux arrivants le monde dans son état courant. De même, lorsque tous les utilisateurs ont quitté le monde virtuel 3D, il peut être nécessaire de conserver l'état dans lequel se trouve le monde virtuel pour fournir à leur prochaine visite le dernier état connu du monde virtuel.

Les techniques de partitionnement de l'espace virtuel 3D ont un rôle à jouer dans la gestion de la persistance : lorsque plus aucun utilisateur n'est présent dans une zone, le système doit être capable de sauvegarder l'état courant des objets partagés présents dans cette zone afin de fournir aux utilisateurs entrant dans la zone la version courante des objets partagés.

L'implémentation de la persistance dépend de la topologie de l'architecture de communication choisie. Comme nous le verrons par la suite, certaines architectures de communication supportent très facilement la persistance, alors qu'avec d'autres, il est très difficile de l'implanter.

Bilan

Dans une application de TCAO, les utilisateurs doivent être capables de sauvegarder les différents documents qu'ils ont produits au cours de la session de travail coopératif pour pouvoir les ressortir par exemple à la prochaine réunion ou bien pour retravailler dessus en dehors de la réunion de travail. De même, on peut imaginer que la session de travail nécessite l'entrée de nouveaux arrivants (comme par exemple, la nécessité de la présence d'un expert pour réaliser une opération particulière) : le système doit donc être capable de fournir l'état courant des documents partagés aux nouveaux arrivants.

Lorsque les participants quittent la réunion de travail, il n'est pas nécessaire que la gestion de la persistance soit transparente comme c'est le cas dans les mondes multi-utilisateurs présents sur Internet. D'une part, les utilisateurs sont placés dans une pièce unique et d'un point de vue ergonomique, on peut très bien envisager qu'un des participants soit chargé de sauvegarder les documents produits. Comme nous le verrons dans la suite de cette section, les documents produits seront alors sauvegardés sur le serveur de TCAO.

2.1.5 Gestion des accès concurrents

Dans un EVC 3D, plusieurs participants peuvent vouloir accéder en même temps à la même donnée partagée. Des actions concurrentes peuvent donner lieu à des incohérences de l'état du monde partagé. Pour certaines applications comme les communautés virtuelles 3D, de telles incohérences peuvent être tolérées temporairement, cependant pour des applications de TCAO, ces incohérences peuvent avoir comme conséquences chez les différents participants une incompréhension de ce qu'il se passe dans le monde virtuel, perturbant ainsi l'activité collaborative. Le système doit être capable de résoudre les conflits d'accès assurant ainsi la cohérence du monde virtuel 3D. Dans [GB94], Greenberg a identifié deux modèles de résolution des conflits :

- *l'approche optimiste* : avec ce type de mécanisme, on part du principe que le participant A va être le seul à manipuler la donnée partagée. Le système diffuse les modifications aux interfaces distantes. Les autres participants sont libres d'accepter ou de refuser les modifications. Si un autre participant B était déjà en train de modifier la même donnée partagée, les participants A et B vont s'envoyer mutuellement des refus de modifications. Suite à la réception d'un refus, toutes les modifications effectuées avant la réception du refus doivent être annulées afin de remettre la donnée partagée dans son état original. C'est l'approche adoptée par CIAO [SYW99],

- *l'approche pessimiste* : elle est basée sur le principe de l'exclusion mutuelle, quand un participant modifie une donnée partagée, aucun autre participant ne peut la modifier. Ce mécanisme utilise un mécanisme de verrous, à chaque donnée partagée on associe un verrou. Quand un participant désire modifier une donnée partagée, il doit au préalable demander le verrou associé. Une fois les modifications effectuées, il peut céder le verrou à un autre utilisateur pour qu'il puisse à son tour effectuer des modifications sur cette donnée partagée. C'est l'approche utilisée dans DIVE [CH93a].

L'avantage de la première approche est de proposer un système réactif. En supposant que la latence du réseau est égale à L, le temps de réponse local est égal à 0 (ie. l'utilisateur manipule immédiatement la donnée partagée), les autres participants sont notifiés après un temps L, l'émetteur de la modification est prévenu de l'acceptation ou du refus de la modification après un temps égal à $2L$. Le principal inconvénient de cette approche optimiste est la confusion qu'elle génère. En effet, un utilisateur manipule les données partagées uniquement au travers de tentatives de modifications. L'utilisateur peut être perturbé par les refus tardifs de modifications et la procédure d'annulation. Un autre inconvénient, d'ordre technique, est la complexité d'implémentation d'une telle approche : le système doit être capable de mémoriser toutes les opérations afin de pouvoir revenir en arrière.

La deuxième approche propose un système moins réactif que l'approche optimiste. En effet, toujours en supposant que la latence du réseau est égale à L, quand un utilisateur souhaite modifier une donnée partagée, le temps de latence du système (le temps entre la demande et le début des modifications) est égal à $2L$ (un délai de L pour envoyer la demande de verrou et un délai de L pour la réception de la réponse). Contrairement à l'approche optimiste, l'utilisateur, une fois le verrou obtenu, manipule directement la donnée partagée, les autres utilisateurs ne peuvent y accéder pour la modifier. De plus, aucune opération d'annulation ne vient le perturber dans sa manipulation. Lorsque le verrou est libéré, l'utilisateur étant le dernier à l'avoir détenu le conserve : on bénéficie alors d'une optimisation lorsque ce même utilisateur redemande le verrou, en effet, il n'y a pas besoin d'envoyer de requête sur le réseau, le verrou est acquis immédiatement (il est à noter qu'il ne faut pas qu'entre temps un autre utilisateur est demandé le verrou).

Bilan

Dans le cadre d'un système coopératif, et par respect des contraintes liées à l'utilisateur, il est préférable d'utiliser une approche pessimiste pour gérer les accès concurrents aux objets partagés du monde virtuel 3D. Une approche optimiste serait susceptible de générer des incompréhensions de la part des utilisateurs : le retour visuel peut ne pas correspondre aux actions de l'utilisateur

et le perturber dans sa compréhension de l'activité.

2.1.6 Les techniques spécifiques issues du TCAO

Dans le cadre d'un collecticiel, il existe des besoins spécifiques. Contrairement aux communautés virtuelles 3D, les personnes susceptibles de participer à l'activité collaborative, les outils nécessaires ainsi que leur fonctions respectives sont connus à l'avance. L'interface de travail est le support temporaire permettant aux utilisateurs de réaliser des tâches collaboratives. Elle permet à son utilisateur de percevoir l'activité, d'y participer et de faire progresser la tâche commune. Il existe une entité centralisée (que nous appelons serveur de TCAO par la suite) permettant de gérer l'ensemble de l'activité collaborative.

Le serveur de TCAO est en charge de conserver l'état de l'activité, l'état des outils, des documents utilisés, mais aussi dans quelle phase est l'activité ainsi que l'état des participants dans cette phase. L'activité collaborative est découpée en phases séquentielles. Une phase (également appelée tâche) correspond à une étape dans l'activité. Une phase regroupe des participants, des documents et des outils. Pour chaque phase, des règles décrivent les droits qu'ont les différents participants sur les documents et les outils. D'une phase à une autre, les participants peuvent changer. Au sein de l'EVC 3D, les utilisateurs réalisent une ou plusieurs tâches.

Le serveur de TCAO fournit également au terminal les différentes descriptions nécessaires à la construction de l'interface de travail. La description indique la pièce de travail, les objets présents, les contraintes spatiales (par exemple de positionnements de deux objets l'un par rapport à l'autre, les objets ayant une place fixe), les droits des utilisateurs sur chaque objet, etc. La description peut éventuellement être adaptée à chaque utilisateur. Une fois la description de la tâche collaborative chargée dans chacun des terminaux, les utilisateurs peuvent travailler ensemble. La description peut évoluer au fur et à mesure que l'activité se déroule, par exemple les utilisateurs peuvent ajouter de nouveaux objets dans l'interface.

Pour supporter l'activité coopérative, le terminal de travail doit être capable de gérer le groupe des personnes présentes. Cela permet de gérer les différents avatars ainsi que renseigner les utilisateurs sur les activités des autres (*qui fait quoi?*).

Dans le cadre d'une tâche, les utilisateurs ont besoin d'outils pour manipuler les différents documents. Il existe de nombreuses applications classiques 2D permettant déjà de manipuler des documents. Il pourrait être intéressant d'utiliser ces outils existants dans le cadre de l'interface de travail. Cependant, un gros travail est nécessaire pour adapter ces outils 2D à un monde 3D :

les mécanismes d'interaction et de présentation ne sont pas les mêmes. De même, d'un point de vue utilisateur, une telle solution n'est pas satisfaisante : bien que l'utilisateur connaisse déjà l'outil 2D classique, nous allons lui demander d'apprendre à utiliser une nouvelle application qui a la même finalité que celle existante. De plus, les utilisateurs risquent d'être troublés de ne pas retrouver les mécanismes 2D qu'ils connaissent déjà. Dans l'idéal, il serait intéressant de pouvoir réutiliser directement les outils classiques comme c'est le cas dans l'approche des outils audiovisuels informatisés (NetMeeting).

2.2 Plate-forme de communication

Dans un environnement virtuel 3D multi-utilisateurs, la plate-forme de communication a une place centrale puisqu'elle permet de faire transiter de l'information entre les différents participants distants géographiquement. Pour donner l'illusion du monde virtuel partagé et pour assurer la cohérence du monde, les différents évènements qui se produisent sur un poste doivent être transmis aux autres postes participants à la session virtuelle. Dans un premier temps, nous présentons les différentes topologies possibles pour la plate-forme de communication puis dans un deuxième temps nous présentons les différents mécanismes permettant son implémentation. Comme le propose M. Macedonia dans [MZ97], il faut distinguer la façon dont est géré l'environnement virtuel de la façon dont sont échangés les messages entre les terminaux.

2.2.1 Taxonomie des topologies de gestion de l'EVC 3D

Nous identifions deux méthodes pour gérer l'EVC 3D : une première méthode consiste à gérer le monde virtuel de façon centralisée, et une deuxième méthode consiste à gérer le monde virtuel de façon décentralisée.

Dans une architecture centralisée (également appelée "client/serveur"), un serveur central est en charge de gérer l'espace virtuel. À une architecture centralisée, on associe une base de données centralisée. Elle est placée sur le serveur. Chaque client communique au serveur les modifications qu'il souhaite opérer. Le serveur est en charge de valider les modifications et de prévenir les clients des modifications afin qu'ils modifient eux aussi la copie de leur base de données. Chaque client n'interagit jamais avec sa copie de la base de données qu'il possède, elle est uniquement utilisée pour le rendu graphique. Le serveur est alors un point central de passage des messages introduisant de la latence supplémentaire dans l'interaction. Très simple conceptuellement et facile à mettre en oeuvre, cette solution est utilisée dans la grande majorité des systèmes d'EVC

3D comme Community Place [HMRL95][LHMM97], DeepMatrix [RCRW99] et la solution multi-utilisateurs de Blaxxun[22], VSPLUS [Ara98] ou encore pLVE [BPPT01]. Très souvent dans les EVC 3D commerciaux, la partie cliente est fournie gratuitement et le serveur est une application payante. Un inconvénient d'une telle architecture est qu'un trafic important passe par le serveur qui peut alors devenir un point de congestion lorsque le nombre de participants devient trop important. De plus, un tel système n'est pas tolérant aux pannes : si le serveur tombe, c'est toute l'application qui tombe. C'est pourquoi certains systèmes proposent de multiplier le nombre de serveurs, d'une part pour réduire le trafic réseau au niveau de chaque serveur, et d'autre part pour offrir un système tolérant aux pannes. Cependant, le fait de multiplier le nombre de serveurs augmente d'autant plus le temps de latence des messages : c'est dû au nombre de points de passage pour chaque message. Des exemples d'utilisation de cette topologie d'architecture de communication sont BrickNet [SSP+95], NetEffect [DSM+97] ou encore RING [Fun95].

Dans une architecture décentralisée, chaque client est responsable de gérer l'espace virtuel. Lorsque qu'un client décide de modifier l'espace, le client diffuse aux clients distants les modifications (une phase de dialogue est éventuellement réalisée entre les clients pour effectuer des vérifications, comme par exemple s'assurer qu'il n'y ait pas de conflits d'accès à une donnée partagée). À une architecture décentralisée, on associe une base de données dupliquée. Chaque client possède une copie de la base de données et lorsque qu'un client désire effectuer des modifications sur le contenu de la base, il le réalise sur sa copie locale et diffuse les modifications aux clients distants afin qu'ils mettent à jour leur copie de la base de données. Des exemples d'utilisation de cette topologie d'architecture de communication sont DIVE [CH93a][CH93b], SIMNET [CDG+93] ou encore NPSNET [MZP+94] ou bien encore Mimaze 3D [GD98], un jeu sur Internet ne reposant pas sur un serveur central. Avec une gestion décentralisée, les algorithmes de gestion de la session virtuelle, de la résolution des conflits peuvent se montrer complexes. Le filtrage des informations est difficile à mettre en place : la mise en oeuvre du partitionnement de l'espace permettant de filtrer l'information et de réduire le débit réseau est inutile. De même, n'existant pas une unique base de données, la gestion de la persistance du monde virtuel 3D est complexe à mettre en oeuvre. Il existe des approches qui tentent de mettre en place des algorithmes de filtrage pour assurer la scalabilité des systèmes à topologie point à point. On peut citer Solipsis [KS02], une plate-forme basée sur une topologie point à point intégrant la notion de voisinage et de partitionnement géométrique de l'espace virtuel. En théorie, le système doit permettre de faire interagir un nombre illimité de machines.

22. `http://www.blaxxun.com`

Bilan

Comme nous venons de le voir, il existe deux façons de concevoir la topologie de la plate-forme de communication, chacune d'entre elles possédant ses avantages et ses inconvénients. Une topologie de gestion architecturée autour d'un serveur (ou à serveurs multiples) est très simple à mettre en œuvre : tous les algorithmes de gestion sont réalisés sur le serveur. Cependant, cette architecture de gestion montre ses limites en terme d'interactivité : en effet, la manipulation s'effectue au travers du serveur et est donc soumise à la latence du réseau. Une topologie décentralisée permet d'assurer l'interactivité du système : l'utilisateur manipule les données partagées en local et prévient les autres participants des changements qu'il est en train d'effectuer. Cependant, les algorithmes de gestion de ces topologies sont plus complexes à mettre en œuvre.

Dans la mesure où nous souhaitons proposer un système interactif avec lequel les utilisateurs peuvent coopérer, il est préférable que nous nous tournions vers une topologie décentralisée pour gérer l'EVC 3D.

2.2.2 Taxonomie des topologies de réseau de communication

Dans la section précédente nous avons vu les deux façons de gérer la session virtuelle. Dans cette section, nous présentons les topologies du réseau de communication (représenté de façon symbolique par le nuage réseau sur la figure 2.1) conditionnant la façon dont les évènements sont échangés entre les terminaux.

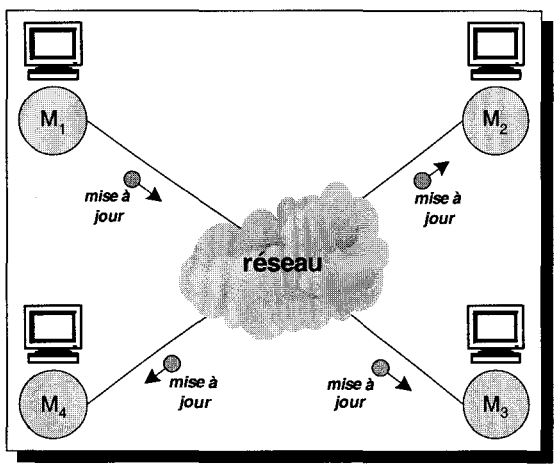
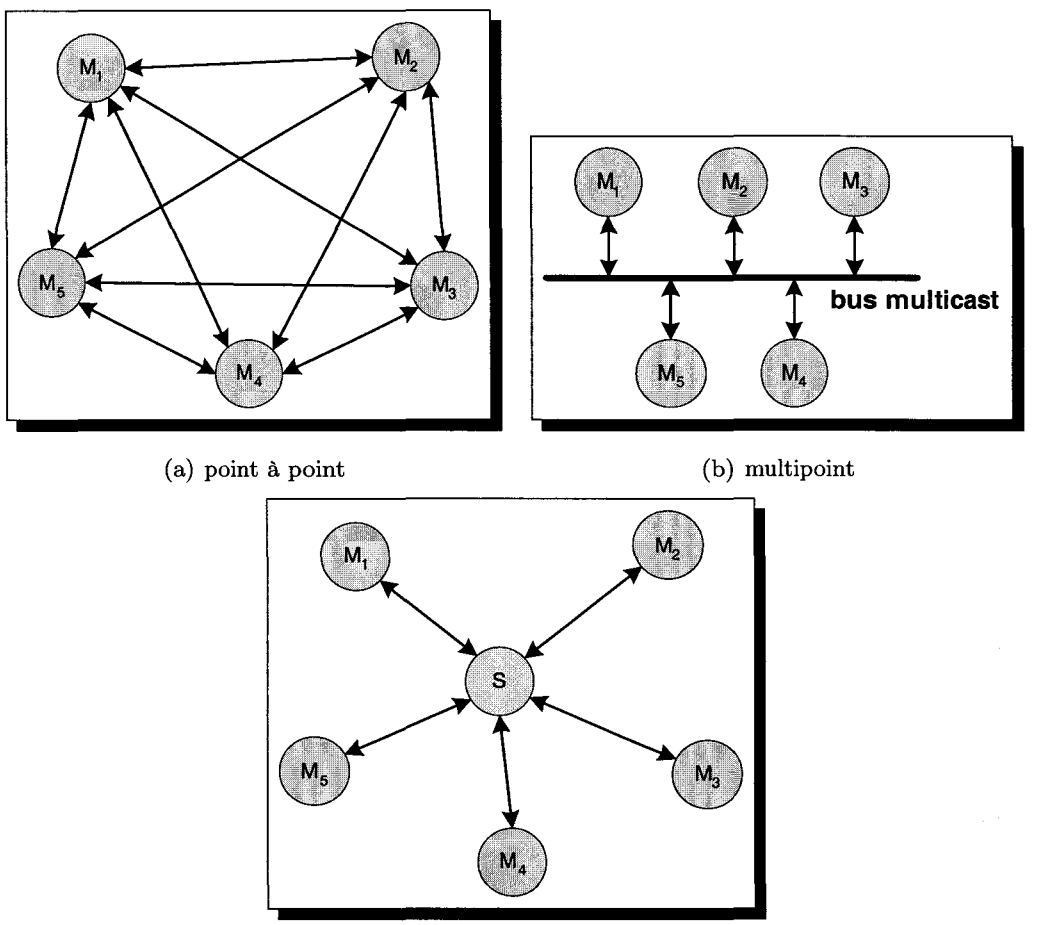


FIG. 2.1 – *Représentation schématique de l'architecture de communication d'un EVC*

Comme il est dit dans [Fun96][MZ97], nous avons trois topologies possibles pour l'architec-

ture du réseau de communications : communication point à point, communication multipoint, communication par pont de diffusion. Ces différentes topologies de réseau de communication peuvent être utilisées indifféremment de la méthode de gestion choisie.



(a) point à point

(b) multipoint

(c) pont de diffusion

FIG. 2.2 – *Taxonomie des topologies de réseau de communication*

2.2.2.1 Topologie point à point

Dans une topologie point à point, chaque participant communique directement les messages le concernant aux autres participants (cf. figure 2.2(a)). Un même message doit être envoyé $N-1$ fois (en supposant que l'on ait N participants), ce qui augmente considérablement le trafic réseau et réduit la scalabilité du système.

2.2.2.2 Topologie multipoint

Au lieu d'envoyer chaque message individuellement à tous les participants, et par conséquent d'augmenter artificiellement la quantité de données à transmettre, un message peut être envoyé sur une adresse multipoint (cf. figure 2.2(b)). Chaque message n'est envoyé qu'une seule fois, chaque participant étant à l'écoute de nouveaux messages sur l'adresse multipoint, ce qui réduit le trafic réseau en comparaison de la topologie point-à-point. Cette approche nécessite que l'infrastructure réseau supporte les flux multipoints.

2.2.2.3 Topologie à pont de diffusion

Plutôt que d'envoyer les messages directement aux autres participants (soit à destination de chacun en point à point, soit à destination du groupe soit en multipoint), une topologie à pont de diffusion se propose d'utiliser une passerelle pour relayer les messages entre les différents terminaux (cf. figure 2.2(c)). Chaque terminal envoie ses messages à la passerelle qui est en charge de les transmettre aux autres terminaux. C'est le cas lorsque l'on utilise un service de notification comme dans LivingSpace [HW99] : lorsque le service de notification reçoit un message de la part d'un client, il est en charge de le transférer aux autres clients.

Cette topologie de communication est souvent liée à l'architecture centralisée utilisée pour gérer l'environnement virtuel. Très souvent, le serveur utilisé pour la gestion et la passerelle pour diffuser les messages aux clients sont le même processus.

Cependant, cette topologie de communication peut être utilisée dans une architecture de gestion décentralisée utilisant des communications multipoint : la passerelle permet de servir de relais entre des sites non directement reliés par un canal multipoint comme c'est le cas dans Spline [WAB+96], DIVE avec le DiveBone [FS98] ou bien encore DWTP [BS98][Bro98].

Bilan

Cette séparation entre architecture de gestion et architecture de communication permet de ne pas lier l'EVC 3D à des contraintes techniques. Ce qui est important ce sont les aspects utilisateurs, ie. la façon dont est gérée la session virtuelle. Nous voulons offrir une architecture de gestion permettant au utilisateurs de travailler ensemble et d'agir de manière interactive dans l'interface. Les différentes topologies de réseau de communication présentées ci-dessus peuvent

être utilisées avec une architecture de gestion décentralisée. Le choix sur la topologie de réseau de communication reste encore ouvert. Nous effectuons le choix dans la suite de ce chapitre.

2.2.3 Techniques de développement

Nous venons de présenter plusieurs topologies de réseau de communication, nous présentons maintenant les méthodes possibles, d'un point de vue logiciel, pour réaliser l'échange d'informations entre différentes machines afin de maintenir cohérent l'état du monde virtuel. Nous disposons de deux paradigmes : soit les différents utilisateurs se partagent virtuellement une mémoire, soit ils s'échangent des messages. L'échange de messages s'effectue soit au niveau des processus (mécanisme de passage de message), soit au niveau des objets partagés de l'interface (mécanisme à invocation de méthode).

2.2.3.1 Mécanisme à mémoire partagée

Une solution pour communiquer des informations entre différentes machines est d'utiliser un mécanisme de mémoire partagée répartie (ce mécanisme est également connu sous le nom anglais de *Distributed Shared Memory* (DSM)) proposé pour la première fois par Li en 1986 [Li86]. Le principe d'une mémoire partagée répartie est de donner l'illusion d'une mémoire partagée à un ensemble de processus distribués sur un réseau de machines ne possédant pas de mémoire commune.

L'étude détaillée des mécanismes de mise en oeuvre d'une mémoire partagée répartie n'entre pas dans le cadre cette étude bibliographique. Le lecteur pourra se référer à la thèse de C. Dumoulin [Dum97] pour une étude plus détaillée. Nous allons donner un aperçu du fonctionnement d'une mémoire partagée répartie (appelée DSM dans la suite de ce document).

Le fonctionnement d'une DSM est basé sur un détournement des mécanismes de pagination mémoire pour faire en sorte que chaque processus voit le même contenu de la DSM. D'autres mécanismes permettent de détecter les accès à la DSM, aussi bien en lecture qu'en écriture. Lors d'une lecture d'une donnée présente dans la DSM, soit :

- le processus possède le droit de lecture de la page contenant cette donnée, la lecture se fait alors de façon classique,
- le processus ne possède pas le droit de lecture, un défaut de page est alors généré. La

DSM recherche alors le processus qui possède la version la plus à jour de la page pour la recopier depuis la mémoire locale du processus distant dans la mémoire locale du processus ayant généré le défaut de page. Le processus possède alors le droit en lecture et peut lire la donnée.

De même, lors de l'écriture d'une donnée dans la DSM, un mécanisme similaire est utilisé en y ajoutant les étapes permettant aux prochaines lectures de prendre en compte l'écriture qui a été faite.

Le comportement de la DSM lors d'un défaut de page est défini par le modèle mémoire. Chaque modèle mémoire met en œuvre un protocole de cohérence déterminant les traitements à réaliser lors d'un défaut de page de la DSM. Le premier modèle de mémoire, le plus naturel, est dit à cohérence séquentielle, c'est-à-dire que toute écriture est immédiatement visible de toutes les machines et que toute lecture renvoie la valeur de la dernière écriture. Ce modèle mémoire permet l'écriture de programmes sans tenir compte du fait que c'est une DSM. Cependant, le coût en temps à l'exécution d'une telle mémoire est assez élevé : la mise en œuvre d'un tel protocole de cohérence implique que lors d'une écriture d'une donnée dans une page mémoire, toutes les copies de cette page présente sur les différents sites soient invalidées avant de pouvoir réaliser effectivement l'écriture. Cela nécessite de diffuser un message d'invalidation à toutes les machines participants à la DSM et d'attendre la confirmation de chacune. D'autres modèles de cohérence ont été proposés de façon à améliorer les performances des DSM. La performance est obtenue au prix d'un relâchement des contraintes du protocole de cohérence et de certaines précautions à prendre lors de la programmation.

Le problème du faux partage

Le problème du faux partage se pose quand différents processus accèdent à une même page contenant différentes données.

Imaginons que chaque processus accède (en écriture) à une donnée distincte de celle accédée par les autres processus. Lors de l'accès, et afin d'éviter des problèmes de cohérence, la page est transférée vers le processus demandant l'accès. Si les processus accèdent régulièrement à des données distinctes contenues dans la page, la page se met à faire le va et vient entre les processus au gré des accès.

C'est un problème de faux partage car les données accédées n'ont pas de rapport entre elles. Si elles étaient placées dans des pages différentes, il n'y aurait pas de problème de va et vient. Le problème du faux partage peut être évité en plaçant les données distinctes accédées par différents

site dans des pages distinctes en effectuant un découpage de la DSM.

Bilan

Une mémoire partagée permet un partage aisé des informations et permet de masquer les différents problèmes techniques sous-jaçents ce qui peut être intéressant dans le cadre de notre activité. Lors du développement d'une application, les concepteurs se concentrent sur les objets présents dans l'interface et non sur les problèmes de maintien de la cohérence.

Cependant, comme l'explique C. Dumoulin [Dum97] dans sa thèse, il n'existe pas de "cohérence universelle" : un modèle de cohérence est adapté à un problème donné, mas peut ne pas convenir à un autre. L'utilisation d'une DSM à cohérence forte est simple et reste transparent du point de vue de l'application. La cohérence forte peut nuire aux performances de l'EVC 3D et à l'activité coopérative qui nécessite une certaine interactivité. Dans le cadre d'une DSM à cohérence relâchée, l'augmentation des performances de la DSM risque de nuire à l'application de TCAO qui, pour certaines données importantes, nécessite une cohérence forte : un manque de cohérence sur ces données pourrait troubler les utilisateurs dans la compréhension de l'activité. Il faut être capable d'utiliser plusieurs modèles de cohérence sur une même DSM afin d'ajuster le niveau de cohérence que nécessite des données présentes dans la DSM.

De plus, pour éviter le problème du faux partage, le partage doit être réalisé au niveau des données elles-même et non au niveau de la structure qui accueille les données comme les pages : dans une application de TCAO, plusieurs données partagées indépendantes doivent pouvoir être modifiées en même temps par des utilisateurs différents.

L'implémentation d'une DSM se fait à l'aide d'échanges de messages entre les différents processus participants à la DSM car il n'existe pas de mémoire physique commune. Dans la suite de ce chapitre, nous présentons le mécanisme à échange de messages.

2.2.3.2 Mécanisme bas-niveau : passage de messages

Dans la littérature, on trouve plusieurs plate-formes ad'hoc pour la communication entre les utilisateurs. Dans ces différentes plate-formes, la plate-forme de communication est développée en utilisant les techniques de programmation classiques (langage C ou Java et bibliothèque bas-niveau de fonctionnalités réseau *"sockets"*) pour envoyer et recevoir des messages réseaux.

Les différents messages entre les participants sont échangés suivant un protocole particulier. Très souvent chaque protocole est associé à une infrastructure réseau. On trouve plusieurs architectures comme DWTP (*Distributed World Transfer Protocol*) [BS98][Bro98], ISTP (*Interactive Sharing Transfer Protocol*) [WAS97], VRTP (*Virtual Reality Transfert Protocol*) [BZWM97], ou encore [FFH+99] dans le domaine universitaire, et Quake d'ID Software[23] dans l'industrie du jeu vidéo.

Toutes ces plate-formes de communication sont des propositions ad'hoc développées *from scratch* en utilisant directement de la programmation réseau de bas niveau. Les plate-formes de communication sont construites au dessus de deux canaux de communication : un canal fiabilisé et un canal non fiabilisé. Le protocole non fiable (comme UDP/IP ou RTP/IP) permet de réduire les temps de réponse du réseau et est utilisé pour les communications en "temps réel", comme par exemple la mises à jour de l'état des différents objets, les flux audio/video. Le protocole fiable (comme TCP/IP) est nécessaire pour les opérations de commande et de contrôle de l'environnement virtuel, comme les entrées/sorties d'avatars ou le *chat*.

Aujourd'hui, de plus en plus de concepteurs d'EVC 3D préfèrent se tourner vers des bibliothèques de communication développées par de tierces personnes de manière à automatiser et standardiser les processus de communication entre programmes distants ainsi que réduire les développements. On peut entre autres citer PVM (*Parallel Virtual Machine*)[24] [GBD+94]. OpenMask [Mar01][MAC+02] utilise la bibliothèque de programmation distribuée PVM permettant d'échanger très facilement (ie. sans aucune programmation au niveau du réseau) des messages entre les différentes machines. La bibliothèque PVM est chargée "d'empaqueter" le message dans un format indépendant du type de machine à laquelle il est destiné. Des fonctions de "désempaquetage" permettent à la machine réceptrice d'extraire le message. D'autres comme LivingSpace [HW99] utilise un service de notification de messages : les messages sont envoyés à un serveur qui est en charge de les redistribuer aux différents clients présents.

Ces différentes bibliothèques n'assurent que le transport des messages entre les machines, une grande partie du travail reste donc à la charge des concepteurs. En effet, les opérations comme le codage/décodage, ou bien le multiplexage/démultiplexage des messages restent à la charge des concepteurs d'EVC 3D. De plus, un autre inconvénient est que les concepteurs sont contraints aux canaux de communication offerts par la bibliothèque. L'utilisation de ces bibliothèques dans le contexte des EVC 3D sort de la fonction principale pour laquelle elles ont été prévues : par exemple PVM a été conçu à la base pour le calcul distribué où l'interactivité n'a aucune im-

23. `http://www.idsoftware.com`

24. `http://www.csm.ornl.gov/pvm/pvm_home.html`

portance. Par exemple, un seul canal, fiabilisé, est disponible assurant une cohérence forte. Cela pose des problèmes avec le besoin d'interactivité que requiert un EVC 3D. Pour relâcher la cohérence et réduire le coût des communications réseaux, des techniques d'interpolation sont mises en œuvre, comme c'est le cas, par exemple, dans SIMNET ou dans NPSNET.

Bilan

Tous ces différents systèmes à passage de messages manquent de généricité et sont donc difficilement extensibles : l'ajout de nouveaux types de données non pris en charge nécessite la modification du protocole (encodage, décodage, etc). Pour atteindre la généricité du mécanisme de communication, les concepteurs se retrouvent à développer un système que l'on pourrait très facilement comparer aux systèmes à invocation de méthodes à distance que nous allons détailler maintenant.

2.2.3.3 Mécanisme haut-niveau : invocation de méthode à distance

Le développement des techniques de programmation distribuée orientée objet peuvent profiter aux EVC 3D, réduisant les temps de développement de la plate-forme de communication, améliorant la généricité, et facilitant la maintenance. Cela permet aux développeurs de se concentrer sur les fonctionnalités propres à l'EVC 3D.

Dans la littérature, les intergiciels se sont développés, et à l'heure actuelle, les trois principaux sont : Java RMI de Sun Microsystems, CORBA proposé par le consortium de l'OMG et DCOM l'intergiciel propriétaire développé par Microsoft. Le grand avantage de CORBA par rapport à ses concurrents est qu'il se place dans un contexte normalisé et multi-langages.

2.2.3.3.1 DCOM

DCOM[25] est une plate-forme de communication propriétaire développée par Microsoft. DCOM (*Distributed-COM*) est une extension distribuée de COM, c'est à dire l'élargissement du concept COM (*Component Object Model*) sous forme d'objets distribués. COM était quant à lui une

25. `http://www.microsoft.com/com/tech/DCOM.asp`

évolution de OLE (*Object Linking and Embedding*), une technologie permettant à des applications Windows d'interagir entre elles. DCOM a été obligé de supporter toutes les contraintes de COM et ses lacunes de conception. Par exemple, il est impossible d'accéder directement à un objet particulier, car aucun mécanisme d'identification des objets n'existe. Il est uniquement possible d'accéder à des interfaces particulières (des services) et non à des instances. De plus, DCOM est une architecture distribuée fermée puisque sous contrôle de Microsoft, ce qui freine son utilisation.

2.2.3.3.2 Java RMI

Java RMI[26] (*Remote Method Invocation*) est une plate-forme de communication développée par Sun Microsystems et faisant partie intégrante de la plate-forme Java. A l'origine RMI fut développée pour ne faire communiquer que des programmes et objets Java entre eux. Ce qui caractérise Java RMI, c'est la possibilité d'effectuer des appels de méthodes sur des objets locaux et des objets distants en utilisant le même mécanisme. Il est ainsi possible de demander à des objets d'effectuer des actions, qu'ils soient ou non dans la même JVM (*Java Virtual Machine*) ou la même machine.

A l'heure actuelle, Java-RMI utilise IIOP (*Internet Inter-ORB Protocol*, protocole utilisé par CORBA que nous présentons dans le paragraphe suivant) comme protocole de communication. Ceci lui permet d'interagir et communiquer avec des objets CORBA qui ne sont pas forcément écrits en Java.

2.2.3.3.3 CORBA

CORBA (*Common Object Request Broker Architecture*) [OMG01d] est une norme ouverte de l'Object Management Group (OMG). La norme définit un ensemble composants logiciels permettant d'automatiser les tâches spécifiques liées à la distribution des objets sur le réseau : l'enregistrement des objets, leur localisation et leur activation ; l'encodage et le décodage des requêtes, des paramètres et des valeurs de retour ; l'invocation des méthodes.

Au centre de l'architecture CORBA, nous trouvons l'ORB (*Object Request Broker*) en charge de localiser un objet sur lequel l'on souhaite invoquer une méthode, de lui passer les paramètres,

26. `http://java.sun.com/products/jdk/rmi/`

puis d'invoquer la méthode, et enfin de retourner le résultat.

Le langage OMG-IDL permet de spécifier l'interface des objets, ie. de décrire les méthodes, et leurs paramètres, que l'on va pouvoir invoquer à distance. Un compilateur IDL génère à partir de la spécification OMG-IDL des souches (*stubs*) pour le coté client et des squelettes (*skeletons*) pour le coté serveur dans le langage de son choix (C, C++, Java, etc).

Le protocole GIOP (*General Inter-ORB Protocol*) est le protocole de haut niveau permettant de coder les messages qui couvrent la sémantique des échanges requête-réponse (demande d'invocation, message résultat, etc.). L'IOR (*Interoperable Object Reference*) permet de localiser sur le réseau les objets d'implémentation. Par exemple, un message GIOP correspondant à une demande d'invocation contient l'IOR de l'objet cible, la méthode à invoquer ainsi que la valeur des différents paramètres. Le protocole GIOP est conçu de manière à pouvoir fonctionner directement au dessus de n'importe quel protocole de transport. IIOP (*Internet Inter-ORB Protocol*) est une déclinaison de GIOP sur TCP/IP. Il spécifie comment les messages GIOP sont échangés sur les réseaux TCP/IP.

En complément de la norme CORBA, l'OMG a spécifié également un ensemble de services, un service étant caractérisé par un ensemble d'interfaces. Les services sont des outils pour faciliter l'utilisation courante des objets CORBA, comme par exemple, le service de nommage, le services de contrôle de le flux, le service d'évènements, le service vendeur, etc.

Le service de nommage [OMG01c] fournit un système de désignation pour retrouver les références d'objets (IOR) à partir de noms symboliques. Il peut être comparé à un annuaire de références où les applications serveurs peuvent enregistrer des références d'objets sous des noms symboliques, et où les applications clientes peuvent obtenir des références d'objets à partir des noms symboliques.

Le service de flux multimédia [MSS99][OMG00] (norme *Audio/Video Streams*) définit par l'OMG permet d'établir et contrôler des flux multimédia entre des producteurs et des consommateurs. Le contrôle des flux est réalisé à l'aide du mécanisme d'appel de méthode. Quant à elles, les données circulant sur les flux ne sont pas transportées par l'ORB. La norme introduit plusieurs interfaces pour le contrôle des flux :

- un périphérique multimédia (`MMDevice`) est composé de producteur/consommateurs élémentaires (`FlowDevice`),
- un flux (*stream*) est composé de plusieurs flots élémentaires (*flow*),
- les terminaisons d'un flux s'appellent des `StreamEndpoints`, et les terminaisons des flots

élémentaires s'appellent des `FlowEndpoints`.

- le contrôleur de flux (`StreamCtrl`) est en charge de gérer la mise en relation deux périphériques mutlimédia.

Lors de la mise en relation de deux périphériques multimédia, la norme propose d'établir une communication entre les deux périphériques afin que chacun d'eux configure ses entrées/sorties en fonction des attentes de l'autre périphérique.

La norme définit deux profils de contrôle des flux : un premier profil donne uniquement accès aux terminaisons des flux (`StreamEndpoints`). Néanmoins, les flots élémentaires (`FlowEndpoints`) peuvent être indirectement contrôlés via l'interface du flux (`StreamEndpoint`) auquel ils appartiennent en utilisant la notion de spécification de flot (`FlowSpec`) : une spécification de flot est une "adresse" assimilable à une URL qui est associée à un `FlowEndpoint`. Cette pseudo URL contient différentes informations comme le nom du flot associé, le protocole de transport utilisé, le type de données, etc. Pour plus de flexibilité dans le contrôle des flux, la norme définit un deuxième profil de contrôle autorisant le contrôle à la fois des `StreamEndpoints` ainsi que des `FlowEndpoints`. Ainsi avec ce deuxième profil, il est possible de connecter directement deux flots élémentaires ensemble.

Le lecteur trouvera en annexe une présentation détaillée de l'architecture CORBA et du service de flux multimédia (cf. annexe B, page 234).

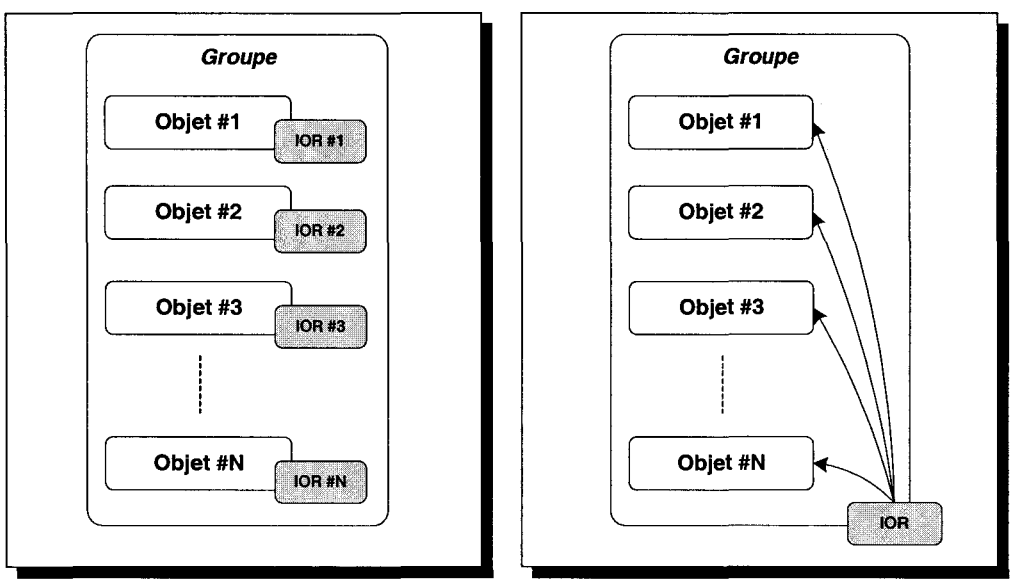
Communication de groupe avec CORBA

[HHBC92] introduit la notion de groupe "logique" et de communication de groupe : une donnée partagée constitue un groupe d'objets répliqués sur des machines différentes. Pour maintenir la cohérence au sein du groupe, Hagsand propose d'utiliser les dernières techniques disponibles (en 1992) en programmation distribuée pour implémenter la plate-forme de communication, préfigurant déjà ce qu'aujourd'hui l'on peut faire avec des intergiciels comme CORBA. Dès qu'un objet d'un groupe "logique" est modifié (la modification s'effectue par une invocation de méthode), la requête est transférée aux autres objets appartenant au même groupe "logique" en utilisant un protocole multipoint fiable. En 1992, la proposition de Hagsand avait été implémentée au dessus d'ISIS [BvR94], une boîte à outils pour le développement d'applications distribuées.

On peut imaginer reproduire ce mécanisme avec CORBA. Pour cela nous devons être capable de définir le principe des communications de groupe "logique" en CORBA. Une première approche est d'utiliser le protocole IIOP que tous les vendeurs d'ORB proposent dans leur implémentation. On peut définir un groupe comme étant l'union de N objets identiques, chaque objet possédant

sa propre référence. Au niveau de chaque client, on doit donc connaître les références de tous les objets présents dans le groupe "logique" (cf. figure 2.3(a)). Une telle approche est difficile à mettre en oeuvre et inadaptée quand le nombre d'objets partagés augmente.

Avec IIOP, une IOR désigne un unique objet, et pour constituer un groupe de N objets identiques on a besoin de N références différentes (chacune pointant sur un objet du groupe). En 1999, dans [GG99], Gransart a proposé un nouveau mécanisme permettant des communications de groupe "logique" en CORBA. Cette proposition a abouti en 2001 à une nouvelle spécification de l'OMG, MIOP (Multicast Inter-ORB Protocol) [OMG99][OMG01e], une déclinaison de GIOP au dessus de UDP/IP. Dans le principe, MIOP introduit la notion de référence (IOR) de groupe : une unique IOR désigne tous les objets identiques appartenant au groupe (cf. figure 2.3(b)). Une IOR de groupe est constituée d'un identifiant de groupe, d'une adresse multipoint et d'un numéro de port. Pour gérer les IORs de groupe, la norme propose deux mécanismes. Le premier est d'utiliser le mécanisme *corbaloc* fourni par la norme CORBA (corbaloc étant une représentation sous la forme d'une URL de l'IOR d'un objet). La deuxième solution est d'utiliser le gestionnaire de groupe multicast (MGM, *Multicast Group Manager*) : c'est un service centralisé permettant d'enregistrer de façon automatique les objets appartenant à des groupes et permettant également la gestion des adresses multicast afin d'éviter les conflits.



(a) en utilisant le protocole IIOP (b) en utilisant le protocole MIOP

FIG. 2.3 – *Communication de groupe avec CORBA*

Au niveau de la partie cliente, la norme MIOP propose de fragmenter chaque message GIOP correspondant à une requête sur le groupe d'objets (cf. figure 2.4). Chaque fragment du message GIOP d'origine est alors envoyé aux différents serveurs hébergeant un objet du groupe. Au niveau de la partie serveur, les fragments sont collectés afin de reconstruire le message GIOP d'origine

et d'exécuter la requête. La norme MIOP propose d'utiliser IP multipoint pour le transport des fragments (l'IP multipoint est présenté dans la section suivante de ce chapitre). Aucun mécanisme de fiabilisation de l'IP multipoint n'est proposé, ce qui pose des problèmes dans la reconstruction de message GIOP et la cohérence du groupe d'objets. Cependant pour les clients ne supportant pas le multipoint, la norme propose une passerelle multipoint (MG, *Multicast Gateway*) : les clients lui envoient leurs requêtes et la MG est en charge de les transmettre aux serveurs.

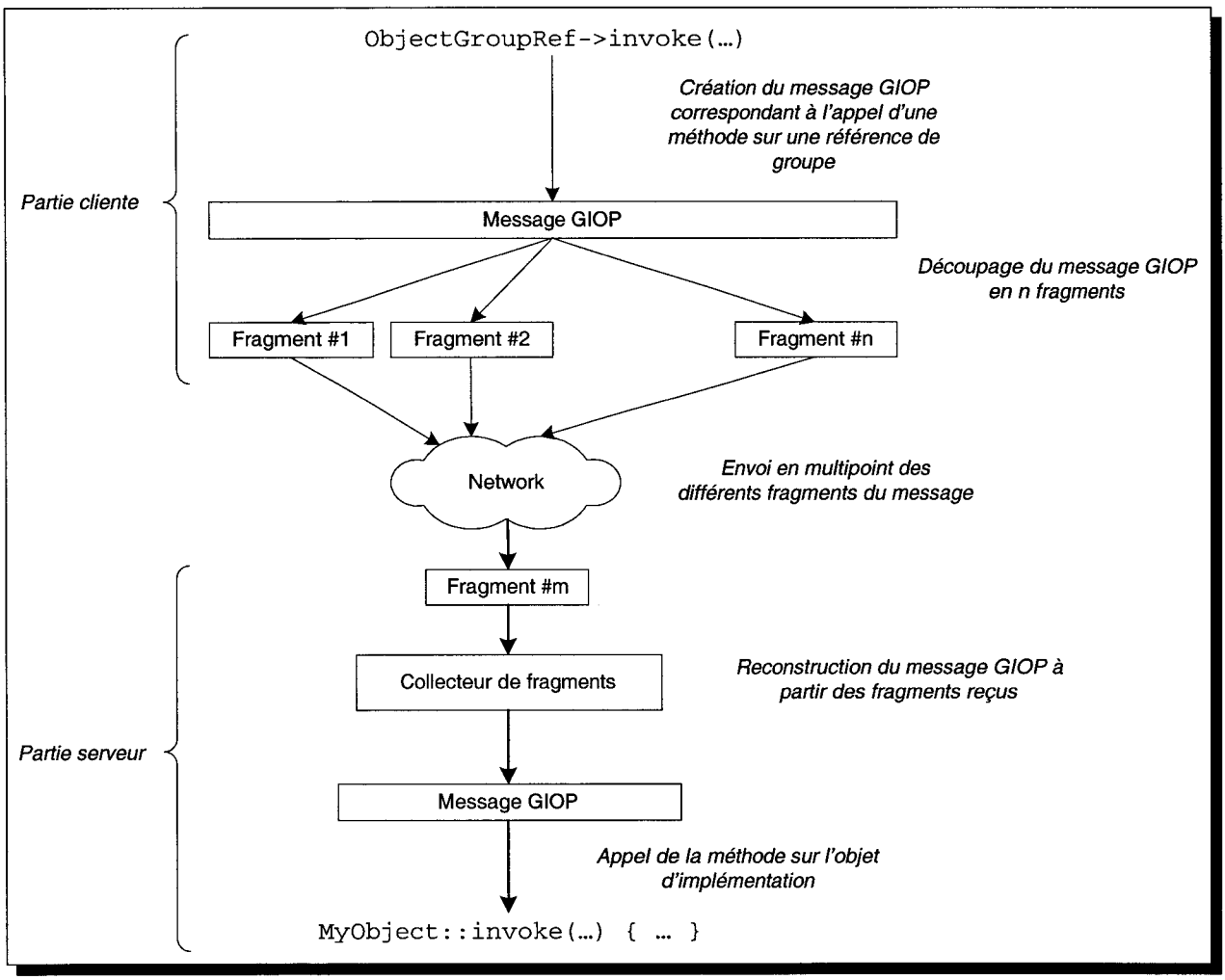


FIG. 2.4 – *Découpage des messages GIOP correspondant aux requêtes MIOP*

Bilan

Une approche utilisant une plate-forme de communication construite autour de l'infrastructure CORBA pour gérer la distribution de l'environnement virtuel offre plus de généricité qu'une approche par passage de messages. Cependant, l'approche consistant à utiliser CORBA de fa-

çon classique (un client/un serveur) n'est certainement pas la bonne solution pour respecter les contraintes liées aux besoins des EVC 3D, notamment en terme d'interactivité. La nouvelle norme proposée par l'OMG, MIOP, semble beaucoup plus adaptée qu'IIOP pour la gestion distribuée d'un EVC 3D. En effet, cette approche de groupe d'objets est naturelle et correspond exactement à la solution proposée par Hagsand : à chaque donnée partagée correspond un groupe logique d'objets au sein duquel il faut maintenir la cohérence. Cette solution revient à simuler une DSM sur laquelle le partage est placé au niveau des données elles-même. Nous devons être capable d'adapter le niveau de cohérence en fonction de la donnée partagée.

Une couche MIOP utilisant une couche de transport fiabilisé pourrait être utilisée comme canal à cohérence forte afin de gérer, par exemple, les entrées/sorties des participants ou les accès concurrents aux données partagées. La norme MIOP telle que l'a proposée l'OMG utilise le multipoint UDP/IP comme couche de transport. Or dans notre contexte de travail, nous préférons abstraire la couche de transport utilisée par MIOP : cela nous permettrait à la fois de conserver le paradigme de communication de groupe, et également de pouvoir s'adapter au contexte technique dans le cas où il n'est possible d'utiliser le multipoint. Pour le canal à cohérence faible, le canal d'animation temps réel, incompatible avec la relative "lourdeur" du protocole GIOP, on peut imaginer l'utilisation du service de flux multimédia défini par l'OMG.

Nous allons maintenant nous intéresser aux différentes techniques permettant de réaliser de la diffusion de messages à un groupe et peuvent être utilisées comme couche de transport à MIOP.

2.3 Techniques des communications de groupe

Dans le principe des communications de groupe, un émetteur désire émettre des données identiques vers de multiples destinataires. Nous allons maintenant détailler plusieurs techniques permettant d'effectuer cette opération.

2.3.1 Le modèle à diffusion

Une solution serait d'utiliser le modèle de diffusion (*broadcast*) IP. Ce type de diffusion est inadapté puisque le réseau local va être inondé par des messages de diffusion. De plus, les messages de diffusion restent confinés au réseau local et ne passent pas les routeurs IP.

2.3.2 Le modèle à pont de diffusion

Une autre solution serait d'utiliser un pont de diffusion : chaque client s'enregistre auprès du pont. Lorsqu'un client veut envoyer un message à destination du groupe, il envoie son message au pont de diffusion qui est en charge de le transférer aux autres clients enregistrés. Avec ce modèle, le pont de diffusion peut très vite être un point de congestion de l'application : tous les messages doivent passer par ce point central.

2.3.3 Le modèle multipoint

L'objectif du modèle multipoint (*multicast* en anglais) est de supporter les communications de groupe sans créer de surcharge au sein du réseau. Le modèle IP multipoint fut proposé pour la première fois en 1989 par Deering [Dee89]. L'IP multipoint est basé sur des adresses IP de classe D[27]. Chacune d'entre elles représente une adresse de groupe. On définit comme membre du groupe un participant désirant recevoir les données émises sur ce groupe et ayant préalablement effectué une procédure d'abonnement pour ce groupe. Il est à noter qu'un participant a besoin de s'abonner au groupe uniquement dans le cas où il souhaite recevoir des données : une source n'a pas besoin d'être membre du groupe afin d'émettre sur celui-ci. Au cours d'une session multipoint, de nouveaux membres peuvent s'abonner, tandis que d'anciens membres peuvent se désabonner.

2.3.3.1 Le réseau du MBone

Pour le moment, Internet (construit aujourd'hui au dessus d'IPv4) ne supporte pas le multipoint IP. Depuis une dizaine d'années, le MBone (*Multicast backBone*) s'est développé au dessus de l'Internet. Ce réseau virtuel expérimental de diffusion multipoint est constitué d'un ensemble de routeurs et de réseaux interconnectés dans le but de distribuer du trafic multipoint. Il est construit sur certains routeurs de l'Internet formant des îlots. Ces îlots sont interconnectés entre eux au moyen de liens virtuels appelés "tunnels". Les tunnels permettent au trafic multipoint d'utiliser les sections Internet ne supportant pas encore le multipoint. Pour cela, les paquets IP multipoint sont encapsulés dans des paquets IP point à point traditionnels. L'encapsulation se fait à l'entrée des tunnels et les paquets sont désencapsulés à la sortie des tunnels.

Des protocoles de routage spécifiques ont été développés permettant d'atteindre les membres

27. Les adresses IP multipoint s'étendent de 224.0.0.0 à 239.255.255.255, certaines étant réservées pour des utilisations précises, comme par exemple la communication entre les routeurs (cf. [RP94] pour la liste complète).

des différents groupes répartis sur tout Internet (construction d'arbres d'acheminement), d'économiser de la bande passante en n'acheminant les paquets multipoint que là où il y a des membres des groupes correspondants, et d'optimiser les échanges entre routeurs. La présentation des protocoles de routage multipoint déployés sur le MBone sort du cadre de cette thèse. Le lecteur peut se reporter à [Lét00] pour une présentation plus détaillée des différents protocoles de routage multipoint.

Actuellement sur le MBone circulent essentiellement des paquets multipoint pour des applications multimedias (visioconférence et audioconférence).

Cette situation de réseau virtuel est transitoire en attendant le support du multipoint en natif dans les protocoles de Internet (prévu dans la nouvelle version d'IP, IPv6) et le choix d'un protocole de routage approprié.

2.3.3.2 Gestion du groupe

Le protocole IGMP (*Internet Group Management Protocol*) [Dee89][Fen97] permet aux machines de déclarer leur appartenance à un ou plusieurs groupes auprès du routeur multipoint dont elles dépendent. Grâce à IGMP, un routeur est capable de savoir si un trafic à destination d'un groupe multipoint doit être diffusé sur un de ses sous réseaux. Les routeurs interrogent périodiquement le réseau local pour savoir si il y a toujours des membres actifs pour chaque groupe multipoint enregistré.

Construit au dessus d'IP, IGMP peut être comparé au protocole point à point ICMP (*Internet Control Management Protocol*)[28] et comprend essentiellement deux types de messages : un message d'interrogation (*Host Membership Query*), utilisé par les routeurs, pour découvrir et/ou suivre l'existence de membres d'un groupe et un message de réponse (*Host Membership Report*), délivré en réponse au premier, par au moins un membre du groupe concerné.

2.3.3.3 Protocole de transport de données multimédia "temps réel"

Le protocole RTP (*Real-time Transport Protocol*) [SCFJ96] fournit des fonctions de transport de données multimédia temps réel sur des réseaux multipoint ou unipoint. Sur le réseau du

28. Le protocole ICMP est un protocole qui permet de gérer les informations relatives aux machines connectées (routage, erreurs).

MBone, de nombreuses applications multimédia temps réel l'utilisent : conférence audio (comme par exemple RAT, *Robust Audio Tool*[29]), conférence vidéo (comme par exemple VIC, *VIdeoconferencing Tool*[30]), diffusion vidéo ou audio, simulation.

RTP ajoute aux données multimédia émises, des marqueurs temporels et des numéros de séquence permettant au(x) récepteur(s) de "rejouer" les données à la vitesse à laquelle elles ont été émises, ou encore d'ignorer les données arrivées en retard.

RTP permet de détecter les pertes de paquets et d'en informer la source via le canal RCTP (*Real-time Tranport Control Protocol*) associé, sans pour autant offrir un mécanisme de retransmission automatique des paquets perdus. RTCP assure le trafic de contrôle et permet un retour pour l'émetteur sur la qualité de transmission. En fonction des informations reçues par l'émetteur via le canal RTCP, l'émetteur peut adapter la bande passante pour limiter le trafic réseau des destinataires.

RTP permet également d'identifier le type de l'information transportée (*payload*). Plusieurs *payloads* ont déjà été définis : comme par exemple, pour des données vidéo au format H261 (RFC 2032), des flots vidéo au format JPEG (RFC 2035) ou encore des données vidéo au format MPEG1/MPEG2 (RFC 2038). Régulièrement de nouveaux *payloads* RTP sont proposés.

2.3.3.4 Fiabilisation

Le modèle IP multipoint s'appuie sur le protocole de transport UDP (*User Datagram Packet*). Rappelons que UDP se limite à garantir une qualité de service du *moindre effort*, n'offrant aucune garantie sur l'arrivée des paquets ni sur leur délai de livraison ou même sur leur ordonnancement.

Un protocole fiable assure que tous les sites reçoivent les messages émis. Il existe pour cela plusieurs techniques [LGLA98] :

- fiabilité orientée récepteur : lorsqu'un récepteur détecte un(des) paquet(s) manquant(s) dans la séquence, il envoie un `NACK` (*Negative Acknowledgement*) pour demander une retransmission du(des) paquet(s) manquant(s) (cf. figure 2.5(a)).
- fiabilité orientée émetteur : chaque paquet émis est soumis à un accusé réception, en l'absence d'acquittement `ACK` (*Acknowledgement*) d'un des récepteurs, l'émetteur retransmet

29. `http://www-mice.cs.ucl.ac.uk/multimedia/software/rat/`

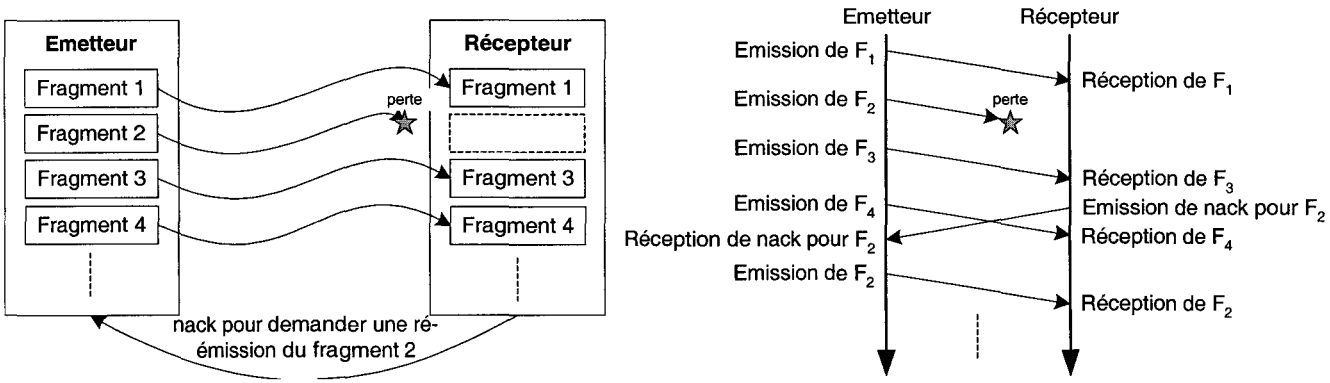
30. `http://www-mice.cs.ucl.ac.uk/multimedia/software/vic/`

le paquet (cf. figure 2.5(b)).

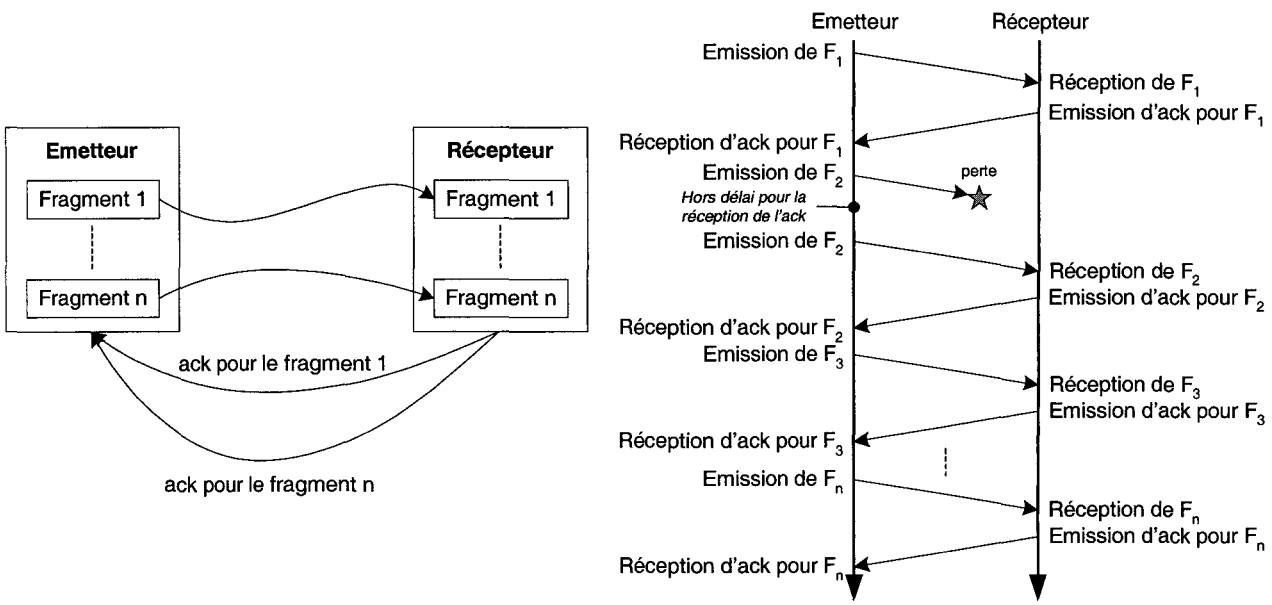
- fiabilité hybride : le récepteur envoie des NACK lorsqu'un trou dans la séquence est détecté. Un ACK est demandé pour certains paquets de la séquence (cf. figure 2.5(c)).

Une fiabilité orientée récepteur est intrinsèquement plus scalable que la fiabilité orientée émetteur, puisque les demandes de retransmission sont envoyées uniquement en cas de pertes. Cependant, l'émetteur n'est pas à l'abri d'une possible implosion de requêtes NACK. Il existe des techniques visant à réduire l'implosion des requêtes de retransmission. L'idée est simple : il est inutile de demander la retransmission d'un paquet si un autre récepteur l'a déjà fait. Les requêtes NACK sont transmises en multipoint à tous les membres du groupe. Un récepteur qui reçoit cette requête retarde alors éventuellement la sienne. Les retransmissions sont ensuite envoyées en multipoint. C'est l'approche adoptée par SRM (*Scalable Reliable Multicast*) [FJL+97]. De plus dans SRM, tout récepteur peut répondre à une demande de retransmission, ce qui implique également des délais aléatoires de retransmission pour réduire les retransmissions inutiles. LRMP (*Lightweight Reliable Multicast Protocol*) [Lia98] reprend les principes de SRM en améliorant le mécanisme de récupération des erreurs.Dans RAMP (*Reliable Adaptative Multicast Protocol*) [KZ96], l'émetteur collecte les demandes de retransmissions pendant une certaine période. À la fin de la période, le nombre de requêtes de retransmission est évalué pour estimer si les paquets manquants doivent être transmis en multipoint ou en point à point.

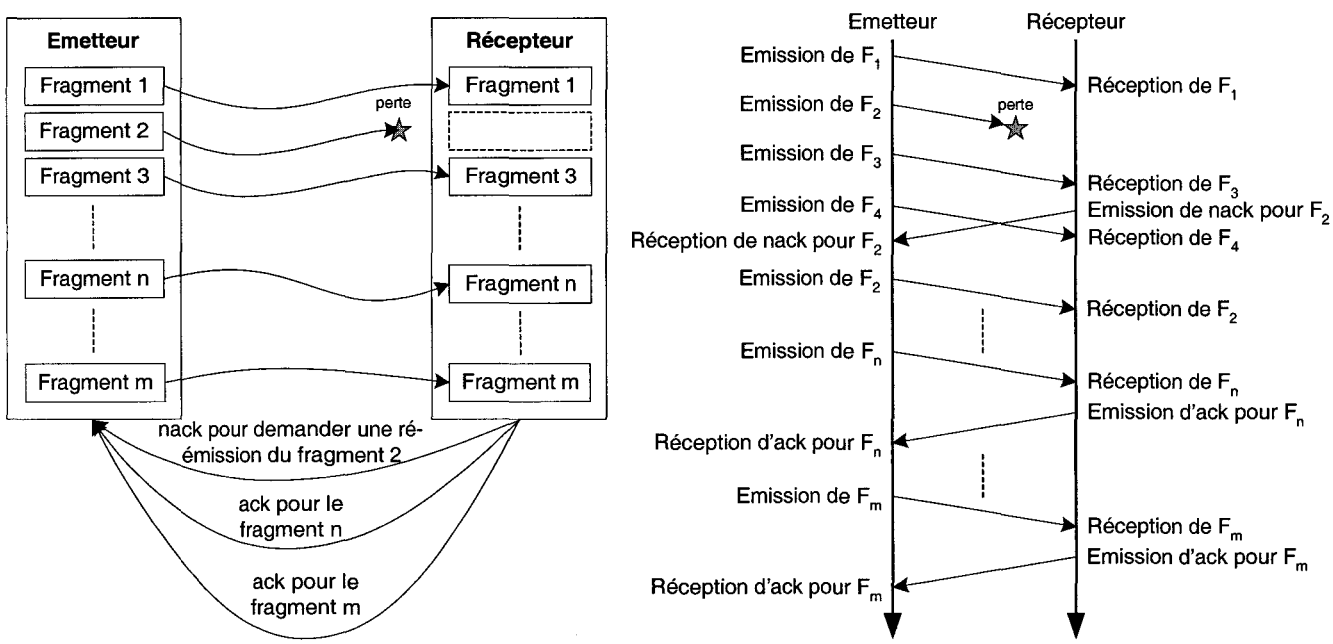
Les problèmes des protocoles multipoints à fiabilité orientée émetteur sont assez évidents. D'une part, l'émetteur doit avoir connaissance des membres du groupe pour gérer les retransmissions quand un ACK d'un récepteur n'a pas été reçu. D'autre part, l'émetteur est confronté, dans le cas de grands groupes, au phénomène d'implosion de messages ACK entraînant par conséquent une congestion du réseau. Des solutions basées sur un mécanisme de distribution ont été développées visant à réduire le phénomène d'implosion de message ACK. XTP (*Xpress Transfer Protocol*) [SDW92] propose une approche centralisée pour la retransmission et permet à un émetteur de maintenir la liste des membres du groupe (arrivée et départ de récepteurs). Cependant, XTP se trouve confronté à un problème d'inefficacité pour des groupes de grande taille. Des méthodes ont été proposées pour améliorer la scalabilité. Pour cela, le contrôle de fiabilité et de retransmission sont distribués. RMTP (*Reliable Multicast Transport Protocol*) [LP96] divise le groupe en sous-arbres hiérarchiques et un récepteur (DR, *Designated Receiver*) est désigné comme maître de chaque sous-arbre. Les récepteurs d'un même sous-arbre envoient leur message ACK au DR local qui s'occupe de l'envoyer à son supérieur dans la hiérarchie. Le DR s'occupe également de la retransmission si une branche n'a pas reçu le paquet. RMP (*Reliable Multicast Protocol*) [WKM94] se base sur une architecture en anneau : à chaque session un participant est désigné comme étant le maître de l'anneau et il lui incombe l'envoi du ACK à la source. Les autres participants deviennent tour à tour le maître.



(a) fiabilisation orientée récepteur



(b) fiabilisation orientée émetteur



(c) fiabilisation hybride

FIG. 2.5 – *Techniques de fiabilisation d'IP multipoint*

D'autres approches combinent les approches de fiabilisation orientées récepteur et émetteur comme dans TMTP (*Tree-based Multicast Transport Protocol*) [YGS95]. Les participants sont organisés en domaines avec un seul manager de domaine (DM) responsable de la récupération des erreurs et des retransmissions locales en utilisant une approche `NACK`. Les DM sont organisés en arbre dynamique. Lorsqu'un membre du domaine émet un paquet au domaine local, le DM qui lui est rattaché lui renvoie un message `ACK` confirmant ainsi la réception du paquet. À son tour, le DM local émet le paquet à destination de son DM parent et ses DM fils en utilisant une approche de type `ACK`.

Bilan

Il existe dans la littérature bien d'autres protocoles multipoint fiables, tous utilisant un des mécanismes de fiabilisation présentés précédemment. Cependant, afin de fiabiliser la couche de transport du protocole de communication de groupe MIOP, les différents protocoles classiquement utilisés sont inadaptés :

- un protocole basé uniquement sur `ACK` peut poser des problèmes de performances comme une surcharge de l'émetteur par des paquets `ACK`,
- un protocole utilisant uniquement des messages `NACK` n'est pas adapté à nos besoins. Un tel protocole n'est utilisable qu'avec des flux continus de données comme de la vidéo ou de l'audio : un paquet est toujours suivi par un autre paquet.

Les messages GIOP générés par MIOP sont de taille limitée : nous pouvons donc être amené à envoyer des messages dont la taille est inférieure à la limite de fragmentation (requête GIOP de petite taille). Dans la mesure où nous souhaitons offrir au protocole MIOP une couche de transport fiabilisée, lorsqu'un message occupent un seul paquet, avec un protocole `NACK`, il devient impossible de détecter la perte de ce paquet. La perte de ce paquet entraîne la perte de messages GIOP (ie. de requêtes de mise à jour des données partagées), et par conséquent, des incohérences dans le monde virtuel 3D.

2.4 Conclusions

Dans ce chapitre, nous avons présenté les différents aspects techniques, liés à la distribution géographique des participants et permettant d'assurer la cohérence du monde virtuel 3D partagé. On retrouve ces différents aspects dans la plupart des logiciels de communautés virtuelles que l'on

trouve sur Internet. Très souvent, ces communautés regroupent un grand nombre d'utilisateurs dans un espace virtuel très vaste, c'est pourquoi des facteurs comme la scalabilité, le découpage de l'espace virtuel ou la distribution des données sont importants.

Au contraire, dans une application de TCAO synchrone, ces différents aspects techniques prennent beaucoup moins d'importance. Une réelle coopération entre les utilisateurs implique un nombre réduit de personnes (environ quatre ou cinq personnes). Les principes de base du TCAO synchrone préconisent de placer le groupe de participants dans une pièce unique. Les différentes solutions techniques doivent permettre aux utilisateurs de coopérer et d'agir de façon interactive avec les objets présents dans l'espace virtuel.

Nous avons également introduit les deux topologies possibles pour la gestion de l'environnement virtuel. Une architecture décentralisée nous permet de conserver l'interactivité lorsqu'un utilisateur manipule un objet partagé : les modifications sont réalisées sur la copie locale et sont transmises aux terminaux distants afin de maintenir la cohérence de l'espace partagé. Ensuite, nous avons introduit la topologie de réseau de communication définissant comment sont échangés les messages entre les machines. Les systèmes existants utilisent le plus souvent une topologie à pont de diffusion pour l'échange des messages entre les machines.

La notion de groupe d'objets répliqués, comme le propose Hagsand et comme le permet MIOP, nous permet de réaliser facilement et de façon naturelle le maintien de la cohérence. À chaque donnée partagée, nous associons un groupe logique constitué des objets répliqués sur différentes machines : dès qu'un membre du groupe est modifié, les autres membres du même groupe sont prévenus des modifications. La norme MIOP permet très facilement de mettre en oeuvre cette solution, cependant pour permettre de s'adapter au contexte technique, il nous semble important d'abstraire sa couche de transport. Ainsi l'utilisation de techniques de diffusion à un groupe, autres que le multipoint, sera possible. L'utilisation d'une topologie point-à-point ou pont de diffusion est classique et toutes les technologies sont dès aujourd'hui disponibles : le canal à cohérence forte utilise TCP/IP, et le canal à cohérence faible utilise UDP/IP. L'utilisation du multipoint pose encore un problème, dans le sens où un mécanisme de fiabilisation, adapté au transport de requêtes GIOP/MIOP, doit encore être proposé. C'est pourquoi nous choisissons d'utiliser le multipoint comme couche de transport à MIOP et aux flots multimédia. Nous devons :

- proposer une couche de transport multipoint fiabilisé et adaptée au protocole MIOP. Cela définirait un canal à cohérence forte,

- adapter le fonctionnement du service de flux multimédia à une architecture de gestion décentralisée. Cela définirait un canal à cohérence relâchée.

Pour offrir un système simple à utiliser (du point de vue du concepteur d'applications), les différents aspects techniques doivent être masqués le plus possible : ainsi le concepteur se concentre uniquement sur le développement de la partie applicative. Dans le chapitre suivant, nous allons présenter différentes plate-formes d'EVC 3D de haut niveau permettant aux concepteurs de créer leurs applications. Nous allons étudier les mécanismes de haut niveau offerts par ces différentes plate-formes permettant de masquer la distribution des données et des utilisateurs afin d'offrir un système simple à utiliser pour le concepteur d'application. De plus, nous allons étudier ce que peuvent nous apporter ces plate-formes pour le développement de notre EVC 3D pour le TCAO synchrone.

Chapitre 3

Exemples d'EVC 3D

Sommaire

Les environnements virtuels 3D collaboratifs (EVC 3D) sont des systèmes de réalité virtuelle permettant à plusieurs utilisateurs, distants géographiquement, d'évoluer dans un monde 3D commun. Les utilisateurs sont souvent représentés par des avatars (2D ou 3D) au travers desquels ils peuvent interagir avec le monde et ses différents objets et communiquer les uns avec les autres.

Dans ce chapitre, nous allons présenter différents systèmes d'environnements virtuels 3D multi-utilisateurs complets. Nous pouvons cataloguer ces systèmes de la façon suivante. Dans une première catégorie, on trouve des systèmes purement descriptifs : ils permettent de concevoir un monde virtuel 3D en limitant le plus possible les aspects liées à la programmation. Les outils permettant de développer de tels mondes restent basiques et se limitent le plus souvent à un simple éditeur de texte. Dans une deuxième catégorie, on trouve des systèmes proposant une interface permettant de développer le monde virtuel, c'est ce que l'on appelle les outils Auteur. Contrairement à la catégorie précédente où le concepteur devait quand même "écrire" son application, un outil Auteur propose une bibliothèque de composants couvrant très largement les besoins les plus courants et la conception du monde virtuel se fait par l'assemblage de différents composants. Finalement, dans une dernière catégorie, on trouve les boîtes à outils : ce sont des bibliothèques de fonctions permettant à un programmeur de développer son monde virtuel 3D.

3.1 Les systèmes descriptifs

3.1.1 Les monde multi-utilisateurs en VRML

Une scène 3D VRML est décrite à l'aide d'un ensemble de noeuds organisés sous la forme d'un arbre. Le comportement de chacun des noeuds du graphe de scène est configuré par l'intermédiaire des champs du nœud. Ces différents champs permettent de caractériser la scène 3D. C'est sur cette remarque que repose le partage d'une scène 3D. On peut en donner la définition suivante : une scène 3D partagée correspond à la spécification d'un ensemble de champs permettant de caractériser la scène 3D et dont la valeur doit être cohérente dans les différentes instances de la scène 3D, i.e. dès qu'une des instances du monde 3D modifie la valeur d'un de ces champs, les changements sont transmis aux autres instances du monde 3D. Ces différents champs sont alors dit partagés.

Nous allons maintenant détailler quelques propositions permettant de décrire en VRML 97 des mondes multi-utilisateurs. Dans la littérature, on trouve différentes techniques permettant

au concepteur de décrire le monde partagé :

- la description du partage est placée dans le fichier VRML 97 décrivant la géométrie et la dynamique du monde. De plus, le partage est décrit en utilisant les différents mécanismes fournis par VRML 97, i.e. respectant la syntaxe du langage,
- la description du partage est placée dans le même fichier que le monde mais en utilisant une syntaxe VRML 97 étendue. De nouveaux mots clés sont alors ajoutés à la grammaire VRML 97,
- la description du partage est placée en dehors du fichier décrivant la géométrie du monde.

Les différentes propositions utilisent les fonctionnalités disponibles dans la norme VRML 97, comme par exemple, la BSI (permettant à des scripts embarqués au sein du graphe de scène d'interagir avec d'autres nœuds) et l'EAI (permettant à des programmes externes d'interagir avec le contenu du navigateur VRML 97). Comme nous le verrons dans cette section, l'EAI est un élément clé dans la définition de mondes multi-utilisateurs en VRML 97 puisqu'elle permet l'interface entre le réseau et le navigateur.

3.1.1.1 Première approche : description du partage en utilisant le langage VRML 97

La première approche est d'utiliser les différentes fonctionnalités offertes par le langage VRML 97 pour décrire le monde partagé.

3.1.1.1.1 Living Worlds

Le groupe de travail Living Worlds a défini un framework permettant la description de mondes multi-utilisateurs en utilisant VRML 97 [LW98]. Ce framework propose un ensemble de nouveaux nœuds permettant la description et le support des mondes multi-utilisateurs en VRML 97. Il utilise les différents mécanismes proposés par le standard VRML 97 comme les prototypes, les scripts, les applets Java et l'EAI. Ainsi, le framework proposé par Living Worlds reste entièrement compatible avec les différents browsers VRML 97 respectant le standard.

Les différentes applications développées avec l'extension Living Worlds sont à la fois "interpersonnelles" et "interopérables" :

- "interpersonnel" par le fait que l'application supporte la présence de plusieurs utilisateurs

dans un même lieu et les utilisateurs sont capables de s'échanger des objets,
- "interopérable" par le fait que le monde virtuel peut être conçu autour de plusieurs technologies multi-utilisateurs fournies par différentes sociétés. Le concepteur de l'application peut ainsi combiner les meilleurs composants multi-utilisateurs afin d'utiliser les fonctionnalités répondant à ses besoins.

L'architecture générale de l'extension Living Worlds présentée sur la figure 3.1 montre les différents composants permettant de partager la scène VRML 97 :

- le monde multi-utilisateurs est divisé en une ou plusieurs `Zones`. Une `Zone` permet de regrouper les différents éléments de la scène VRML 97 devant être partagés,
- une `Zone` contient un ou plusieurs `SharedObjects`. Chaque `SharedObject` représente une entité qui doit être partagée entre les différents utilisateurs présents dans le monde virtuel, i.e. dont l'état doit être synchronisé. Tout `SharedObject` peut avoir deux états : `Pilot` ou `Drone`. A un instant donné et pour un `SharedObject` donné, il existe un seul `Pilot` et plusieurs `Drones`, le `Pilot` doit communiquer tout changement aux `Drones`,
- chaque `SharedObject` contient plusieurs valeurs partagées, i.e. chacune de ces valeurs doit être synchronisée parmis les différents utilisateurs présents dans le monde. Avec le framework Living Worlds, on les appelle des `NetworkStates` : pour chaque type de base VRML 97 (`SFBool`, `SFColor`, etc), Living Worlds définit un nœud équivalent partagé (`NetworkSFBool`, `NetworkSFColor`, etc),
- la technologie multi-utilisateurs (`MUTech`) est le composant implémentant le comportement des différents éléments de la proposition (`Zones`, `SharedObjects`, et `NetworkStates`). Ce composant est en charge de gérer la distribution des évènements.

Living Worlds s'intéresse également à la sécurité du monde partagé. Dans chaque nœud (`Zone` et `SharedObject`), le contenu VRML 97 est isolé dans un nœud privé (`PrivateZone` et `PrivateSharedObject`) de façon à le protéger des attaques.

Living Worlds utilise le paradigme du mécanisme d'insertion pour transmettre à distance les évènements (cf. figure 3.2) : de nouveaux nœuds, chargés de propager à distance les modifications locales (événements), sont introduits dans le graphe de scène existant. Cela nécessite de la part du concepteur de modifier les routes existantes afin de tenir compte de ces nouveaux nœuds dits "de partage".

Outre la nécessité de modifier les chemins d'animation existants, le mécanisme d'insertion pose un problème. En effet, avec un tel mécanisme, il n'y a aucune cohérence entre les différents postes au niveau des capteurs de manipulation (`Sensors`). Étant les initiateurs de tout chemin

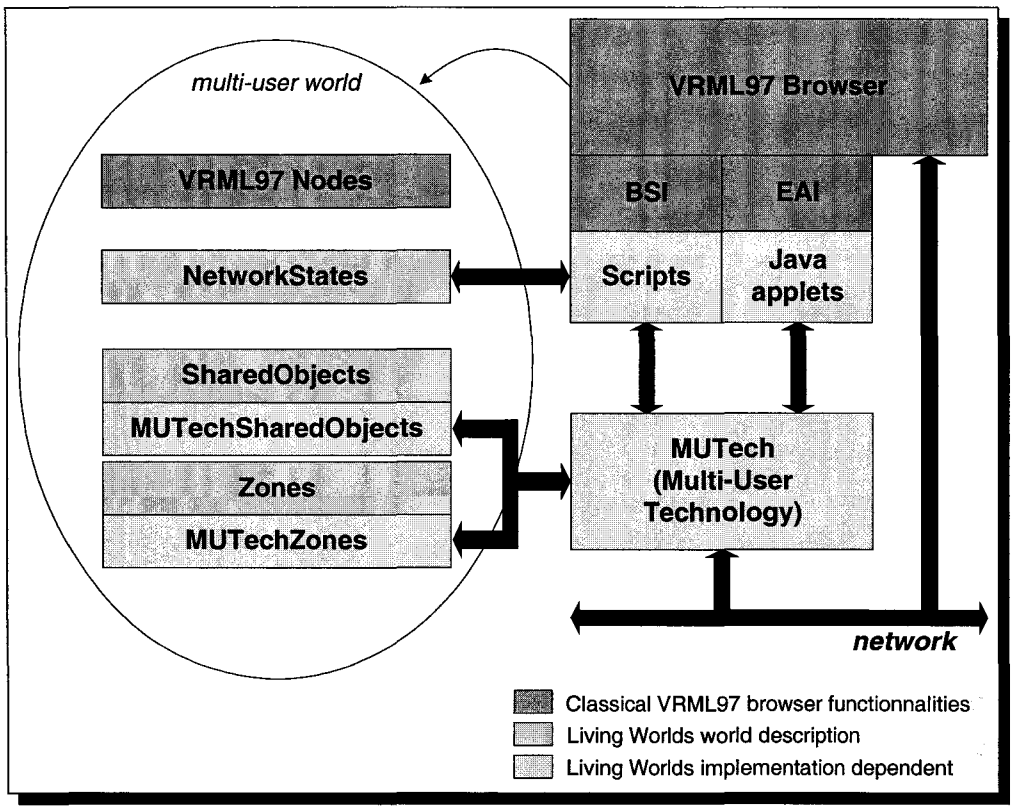


FIG. 3.1 – *Architecture de Living Worlds*

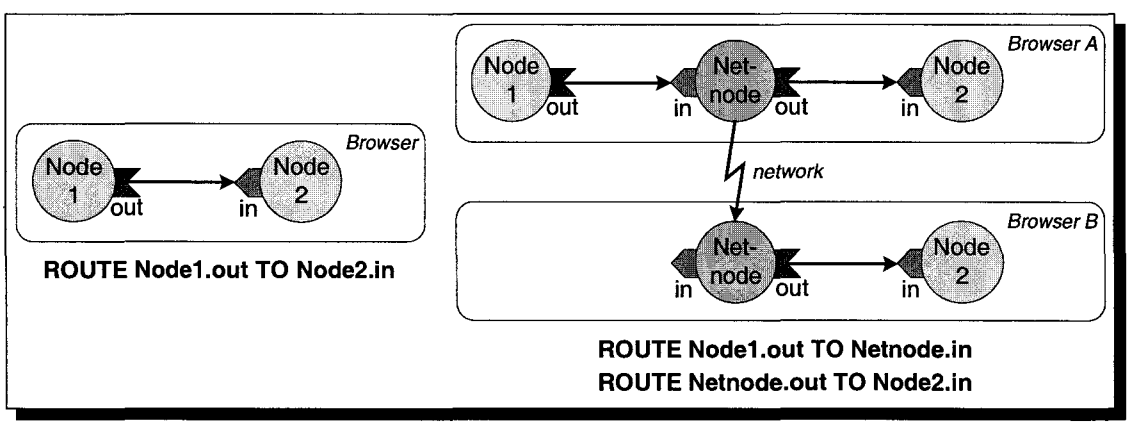


FIG. 3.2 – *Paradigme du mécanisme d'insertion : les routes existantes doivent être modifiées*

d'animation, leurs sorties (`EventOut`) ne peuvent être synchronisées avec ce mécanisme. L'état de chaque capteur de manipulation n'est donc pas cohérent avec l'état des variables qui lui sont reliées via le mécanisme de routes. Seul le poste à l'origine de la modification est cohérent ; sur les autres postes, cela va introduire des problèmes de compréhension de la part de l'utilisateur lors des futures manipulations (un saut brusque lors de l'animation pour retrouver un état cohérent entre la valeur de sortie du capteur et les variables qui lui sont connectées via le mécanisme de route). Il existe une solution technique présentée dans l'annexe C permettant de compenser le manque de cohérence entre les copies d'un même `Sensor` inhérent à l'utilisation du paradigme d'insertion : pour cela les `Sensors` sont placés en mode relatif et un script est en charge de réaliser le cumul, le noeud de partage étant inséré entre la sortie du `Sensor` et l'entrée du nœud `Script` assurant le cumul. Une telle solution reste complexe à mettre en œuvre et nécessite de bonnes connaissances en VRML 97.

La figure 3.3 montre un exemple de contenu multi-utilisateurs créé à l'aide du framework Living Worlds : une boite dont nous souhaitons partager le point de vue (cet exemple sera repris tout au long de ce chapitre pour illustrer et comparer les différentes propositions). Pour réaliser cet exemple, nous sommes partis d'un contenu VRML 97 mono-utilisateur auquel nous avons ajouté les informations permettant son partage. On remarque que la syntaxe proposée n'est pas facile à appréhender. De plus, cela illustre le mécanisme d'insertion : les différentes routes existantes doivent être modifiées afin de tenir compte des `NetworkStates` ajoutés au graphe de scène.

```
#VRML V2.0 utf8

# Some external definitions
EXTERNPROTO ...

# World description
DEF SO SharedObject {
 private PrivateSharedObject {
  state [ DEF R NetworkSFRotation {} ]
  visibleDefinition DEF T Transform {
   children [
    DEF CS CylinderSensor {},
    Shape {
     appearance Appearance { material Material { diffuseColor 1 0 0 }}
     geometry Box { size 1 1 1 }
    }
   ]
  }
}}

# Routes description
# removed: ROUTE CS.rotation_changed TO T.rotation
ROUTE CS.rotation_changed TO R.set_value
ROUTE R.value_changed TO T.rotation
```

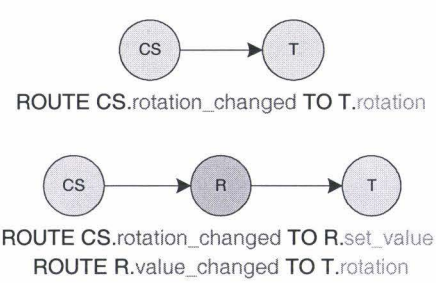


FIG. 3.3 – *Un exemple de contenu multi-utilisateurs conçu avec le framework Living Worlds (en gras, les parties provenant du contenu mono-utilisateur)*

A l'heure actuelle, il n'existe aucune implémentation complète des différents nœuds de partage

proposés par l'extension multi-utilisateur de Living Worlds. La proposition Living Worlds reste uniquement une technique de description de mondes multi-utilisateurs en VRML 97 : aucune plate-forme de communication ni aucun format de codage des données échangées sur le réseau n'y sont définis. Les implémentations (simplifiées) de Living Worlds que nous allons voir par la suite proposent un système complet incluant une plate-forme de communication.

3.1.1.1.2 Core Living Worlds

La proposition de Living Worlds est souvent jugée trop complexe à mettre en oeuvre et à utiliser [HW99]. D'une part, comme nous venons de le voir dans le paragraphe précédent, la conception du monde partagé et la syntaxe utilisée ne sont pas des plus simples. D'autre part, un gros travail d'abstraction afin de bien séparer les concepts de l'implémentation a été effectué, c'est pourquoi Living Worlds ne définit aucun protocole réseau au niveau du `MUTech`. Cela a eu pour conséquence le développement de différents blocs `MUTech` non interoperables entre eux. Sony propose Core Living Worlds [CLW98] qui regroupe un sous-ensemble des nœuds proposés par Living Worlds.

Core Living Worlds reprend les différents concepts proposés par Living Worlds et présentés précédemment : les prototypes `Zones`, `SharedObjects`, et `NetworkStates` sont toujours présents mais leurs interfaces ont été simplifiées. Pour assurer la transmission des évènements à distance, le paradigme du mécanisme d'insertion est également utilisé.

Sony a fait la démonstration de Core Living Worlds en l'implémentant dans son browser VRML 97 Community Place [HMRL95][LHMM97]. D'autres environnements virtuels multi-utilisateurs utilisent la proposition de Sony pour la description du monde partagé. DeepMatrix [RCRW99], VNet et VNet+ [WS] implémentent la proposition Core Living Worlds. Le visualisateur de monde 3D est un navigateur VRML 97 classique (Blaxxun Contact, Cortona ou CosmoPlayer) et l'interface proposée par l'EAI est utilisée pour faire le lien entre la couche réseau et le navigateur VRML 97. Tous trois utilisent une architecture client/serveur pour la plate-forme de communication. Elles sont toutes basées sur un mécanisme de passage de messages développé *from scratch*. Un protocole réseau a été défini dans VNet et repris dans DeepMatrix permettant l'encodage et l'échange de données entre le serveur et les clients : VIP (*VRML Interchange Protocol*) [Whi98] est un protocole simple et facile à mettre en oeuvre permettant, entre autres, d'envoyer des valeurs de champs VRML 97 sur le réseau. VIP utilise le protocole point à point TCP/IP pour les échanges de données avec le serveur. [RSL00][RDS03] propose MVIP (Multicast VIP), une adaptation du protocole VIP pour les réseaux multicast (non fiable).

LivingSpace [HW99] implémente également la proposition Core Living Worlds. Là où les autres propositions comme DeepMatrix ou VNet utilisent des solutions ad-hoc développées spécialement pour la communication entre les clients et le serveur, Living Space se distingue par l'utilisation d'un service de notification (*Keryx Notification System*). Lorsqu'un client modifie une donnée partagée, il prévient le service de notification qui est en charge de prévenir les autres clients de la modification.

3.1.1.1.3 La technologie de Blaxxun

Apparue en 1998, la technologie proposée par Blaxxun[31] est une des plus abouties du marché. La technologie de Blaxxun est utilisée par de nombreux sites de communautés virtuelles, on peut entre autre citer le site de *Cybertown*[32] (cf. figure 1.3, page 20) et le site du *2ème monde* de Canal+. Elle regroupe :

- un client (*Blaxxun Contact, BS Contact VRML*) capable de visualiser des scènes 3D VRML 97,
- un serveur de communauté virtuelle (*Blaxxun Community Server*) permettant de mettre en ligne des applications multi-utilisateurs et de gérer la communication entre les différents participants,
- un SDK (*Blaxxun Community Platform SDK*) permettant de créer une application sur "mesure".

Le navigateur proposé par Blaxxun est un des plus complets du marché. De plus, profitant des améliorations du hardware graphique, il intègre de nombreuses extensions à la norme VRML 97 permettant d'enrichir la scène (*multi-texturing*, *nurbs*, systèmes de particules). Ces différentes extensions sont intégrées dans la prochaine norme X3D (également connu sous le nom de VRML 3.0).

Tout comme Sony, Blaxxun a réfléchi à l'utilisation de la norme VRML 97 pour les mondes multi-utilisateurs. Ils ont défini de nouveaux nœuds VRML 97 permettant de décrire le partage d'une scène. Ces différentes propositions ont été intégrées à leur navigateur.

Leur proposition reprend les grandes lignes introduites par Living Worlds, à un tel point que

31. `http://www.blaxxun.com` et `http://www.bitmanagement.de`

32. `http://www.cybertown.com`

les propositions de Blaxxun et Sony se ressemblent beaucoup. On retrouve les mêmes concepts mais sous des noms différents (le prototype `BlaxxunZone` au lieu du prototype `Zone`, le prototype `SharedEvent` au lieu des prototypes `NetworkStates`, et l'on retrouve la notion de prototype représentant un `SharedObject`). Encore une fois la proposition multi-utilisateurs de Blaxxun repose sur le paradigme du mécanisme d'insertion. Contrairement aux `NetworkStates` de (Core) Living Worlds qui pour chaque type VRML 97 de base définissait un noeud embarquant le comportement réseau, Blaxxun propose un seul noeud `SharedEvent` pour tous les types VRML 97 de base : le noeud `SharedEvent` expose plusieurs champs, un pour chaque type VRML 97.

Blaxxun propose dans sa solution multi-utilisateurs différentes notions pour la gestion des évènements partagés (`SharedEvents`) : la gestion des accès concurrents, la distribution des évènements partagés à un groupe d'utilisateurs et non à toutes les personnes présentes dans le monde virtuel, etc. Avec le framework de Blaxxun, il est possible de nommer les `SharedEvents`. Pour accéder aux fonctionnalités comme la gestion de verrous, le nom des `SharedEvents` doit contenir une chaîne de caractères particulière, comme par exemple `"_LOCK_"` pour gérer les verrous, `"_S_GRP_"` pour l'envoi à un groupe de personnes, etc.

La figure 3.4 montre un contenu multi-utilisateurs créé avec le framework de Blaxxun. Le contenu VRML 97 est bien plus complexe que celui vu précédemment. En effet, profitant des fonctionnalités offertes par le framework multi-utilisateurs de Blaxxun, nous effectuons une gestion des accès concurrents concernant la rotation de la boîte 3D. Quelques explications doivent donc être apportées quant au fonctionnement de ce contenu VRML 97. Le `SharedEvent "Lock"` permet d'effectuer le contrôle d'accès à la rotation de la boîte 3D. Ce `SharedEvent` possède un nom particulier `"P_S_LOCK_REQUEST"`. La chaîne de caractères `"_LOCK_"` permet de spécifier au serveur qu'il faut gérer cet évènement comme un verrou, ie. le serveur reçoit des demandes de verrous via la propriété `set_string` du `SharedEvent` et renvoie le nom de la personne qui possède le verrou via la propriété `string_changed` ; d'autre part, la chaîne de caractères `"_S_"` permet de spécifier que la communication s'effectue seulement entre le client et le serveur, ie. le serveur ne propage pas aux autres clients les changements de valeurs de ce `SharedEvent`. Ensuite, un script permet d'effectuer la gestion du contrôle d'accès en local : à chaque manipulation du sensor, le script effectue une demande de verrou ; suivant la réponse du serveur, il propage ou non la valeur de la rotation du sensor à la rotation de la boîte 3D. Le mécanisme de verrou mis en place est un mécanisme associé à la donnée : à chaque modification, l'acquisition et la libération du verrou sont effectuées à chaque modification du champs. Cette approche convient pour des actions ponctuelles, mais est inadaptée à la manipulation fréquente d'une donnée (comme par exemple, déplacer ou faire tourner un objet).

Comme le montre l'exemple présenté ci-dessus, le développement de monde multi-utilisateurs

```
#VRML V2.0 utf8

# Some EXTERNPROTO definitions
...

# World description
DEF SharedZone BlaxxunZone {
 events [
  DEF Lock SharedEvent { name "P_S_LOCK_REQUEST" }
  DEF SharedRotation SharedEvent { name "SHARED_ROTATION" }
] }

DEF T Transform {
  children [
    DEF CS CylinderSensor {},
    Shape {
      appearance Appearance { material Material { diffuseColor 1 0 0 } }
      geometry Box { size 1 1 1 }
} ] }

DEF S Script {
  eventIn SFRotation set_rotation
  eventIn SFString set_lock
  eventIn SFRotation clear_lock

  eventOut SFString lock_requested
  eventOut SFString lock_cleared
  eventOut SFRotation rotation_changed

  field SFString lock
  field SFString myAvatarName
  field SFRotation rotation

  url "javascript:
    function initialize()
    {
      lock = '';
      myAvatarName = Browser.myAvatarName;
    }

    function set_rotation(v, t)
    {
      if (lock == '') { // request lock to the server
        rotation = v;
        lock_requested = myAvatarName;
      }
    }

    function set_lock(v, t)
    { // process the reply received from the server
      lock = v;
      if (lock == Browser.myAvatarName) { // lock is granted
        rotation_changed = rotation;
      }
    }

    function clear_lock(v, t)
    {
      if (lock == Browser.myAvatarName) { // release the lock
        lock = '';
        lock_cleared = '';
      }
    }"
}

# Routes description
# removed: ROUTE CS.rotation_changed TO T.rotation
ROUTE CS.rotation_changed TO S.set_rotation
ROUTE S.lock_requested TO Lock.set_string                 # from script to network
ROUTE Lock.string_changed TO S.set_lock                   # from network to script
ROUTE S.rotation_changed TO SharedRotation.set_rotation   # from script to network
ROUTE SharedRotation.rotation_changed TO T.set_rotation   # from network to local transform
ROUTE T.rotation_changed TO S.clear_lock                  # unlock
ROUTE S.lock_cleared TO Lock.set_string
```

FIG. 3.4 – *Un exemple de contenu multi-utilisateurs conçu avec le framework de Blaxxun (en gras, les parties provenant du contenu mono-utilisateur)*

avec ce framework requiert, de la part du concepteur, une excellente maîtrise du langage VRML 97 et l'écriture de nombreux scripts pour tirer pleinement parti de ses fonctionnalités.

La plate-forme de communication utilisée par Blaxxun est de type client/serveur. Elle permet d'échanger des messages entre un client et le serveur. Pour cela, elle propose un unique canal utilisant le protocole fiable TCP/IP. Le serveur est en charge de distribuer les messages aux différents destinataires en utilisant les informations contenues dans la description VRML 97 (en particulier, le nom du `SharedEvent`) et assure la gestion des verrous.

3.1.1.1.4 VSPLUS

Comme nous l'avons vu précédemment, la syntaxe proposée par Living Worlds peut apparaître très complexe. Dans VSPLUS [Ara98], Araki propose une alternative à l'utilisation de Living World. Le système de description du partage de données est beaucoup plus simple que celui proposé par (Core) Living Worlds. VSPLUS n'offre pas toutes les fonctionnalités de (Core) Living Worlds, mais il permet à un concepteur, non expert en programmation et dans l'utilisation de VRML 97, de créer facilement des mondes multi-utilisateurs. En effet, la principale motivation du système proposé est la facilité de création de contenus multi-utilisateurs à partir de contenus mono-utilisateur.

Tout comme (Core) Living Worlds, VSPLUS utilise le paradigme du mécanisme d'insertion : de nouveaux nœuds (appelés `NetNodes` dans le framework VSPLUS) sont ajoutés au graphe de scène mono-utilisateur. Chaque `NetNode` est en charge de transmettre aux copies distantes les évènements qu'il reçoit en entrée, de sorte que toutes les copies distantes génèrent les évènements de sorties. Les `NetNodes` peuvent être comparés au `NetworkStates` proposés par le framework Living Worlds : pour chaque type de base VRML 97 (`SFBool`, `SFColor`, etc), VSPLUS définit un nœud embarquant le comportement réseau (`NetSFBool`, `NetSFColor`, etc).

La figure 3.5 montre un exemple de contenu multi-utilisateurs créé avec le framework VSPLUS. On remarquera que la syntaxe est plus facile à appréhender que celle utilisée par Living Worlds, et par conséquent l'approche proposée par VSPLUS permet de décrire plus facilement un contenu multi-utilisateurs qu'avec Living Worlds. Cependant, comme VSPLUS utilise lui aussi le mécanisme d'insertion, le concepteur doit toujours modifier les routes existantes afin de prendre en compte l'ajout des différents `NetNodes` au graphe de scène. De plus, le problème de cohérence des `Sensors` est toujours présent et nécessite toujours de bonnes connaissance du langage VRML 97 pour être résolu (cf. annexe C, page 240). De plus, de base aucun mécanisme

permettant d'éviter les accès concurrents n'est proposé.

```
#VRML V2.0 utf8

# Some external definitions
EXTERNPROTO ...

# World description
DEF T Transform {
  children [
    DEF CS CylinderSensor {},
    Shape {
      appearance Appearance { material Material { diffuseColor 1 0 0 }}
      geometry Box { size 1 1 1 }
}]}

DEF R NetSFRotation {}

# Routes description
# removed: ROUTE CS.rotation_changed TO T.rotation
ROUTE CS.rotation_changed TO R.request_operate
ROUTE R.value_caught TO T.rotation
```

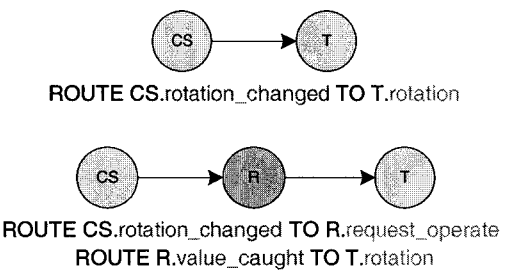


FIG. 3.5 – *Un exemple de contenu multi-utilisateurs conçu avec le framework VSPLUS (en gras, les parties provenant du contenu mono-utilisateur)*

VSPLUS utilise le navigateur VRML 97 Community Place de Sony pour afficher le monde 3D. La plate-forme de communication est de type client/serveur (cf. figure 3.6) : le serveur (Consistency Manager) est en charge de distribuer les messages de modification aux clients. Aucune information n'est disponible sur le(s) canal(aux) de communication disponible. L'interaction se fait via le serveur : le noeud 0 du client A génère un évènement E (évènement Pa) qui est envoyé au serveur via le noeud de partage N ; le serveur traite le message reçu (évènement Pb) et notifie tous les clients (y compris celui ayant généré l'évènement E) du message (évènement Pc) afin que chaque navigateur gère l'évènement E. Le *Consistency Manager* a également pour rôle de fournir aux nouveaux arrivants la version courante du monde partagé.

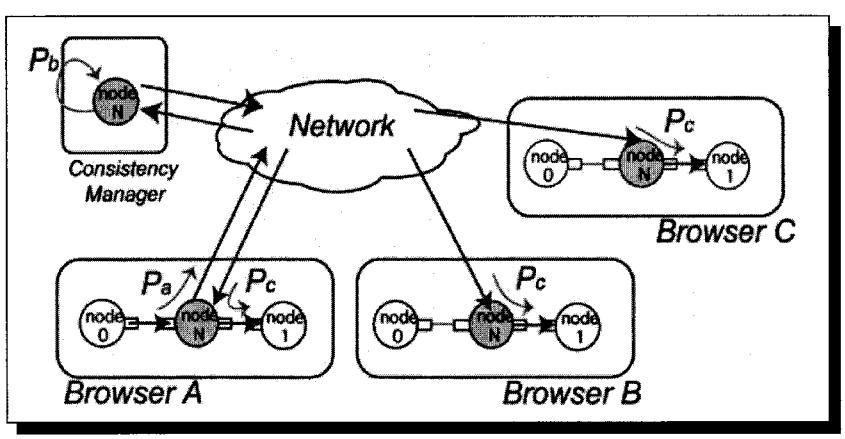


FIG. 3.6 – *Fonctionnement de la plate-forme de communication introduite dans VSPLUS [Ara98]*

3.1.1.2 Deuxième approche : description du partage par extension du langage VRML 97

Une deuxième approche pour la conception de monde multi-utilisateurs en VRML 97 est d'ajouter de nouveaux mots clés au langage VRML 97. Ces différents ajouts au langage permettent de décrire le monde partagé. Dans la norme VRML 97, le mot clé `ROUTE` permet de faire transiter des évènements au sein du monde 3D. Dans un monde partagé, nous avons besoin de maintenir la cohérence entre les différentes instances du monde, par conséquent les événements qui se produisent dans une instance doivent être transmis aux autres instances afin de produire les mêmes effets. Dans [CC99], Carson propose le concept de `ROUTE` partagée en ajoutant au langage VRML 97 les mots clés `SHARED_ROUTE` et `LOCKED_ROUTE`. Ces nouveaux mots clés reprennent la syntaxe du mot clé classique `ROUTE` de la norme VRML 97. Tout événement transitant sur une route partagée est transmis aux autres instances du monde. Ainsi, à distance, l'évènement reçu du réseau est également transmis au(x) destinataire(s) de la route partagée, de la même façon que si l'évènement avait été déclenché localement. Alors qu'une `SHARED_ROUTE` transmet les différents évènements sans effectuer de contrôle d'accès, une `LOCKED_ROUTE` effectue une vérification et une acquisition de verrou avant d'effectuer la transmission, cela permet d'éviter les conflits d'accès. Pour la gestion des verrous, un service centralisé, le *Gatekeeper*, est introduit dans l'architecture. Outre le service de gestion de verrous, le Gatekeeper offre également un service de démarrage : ce service propose aux nouveaux arrivants l'état courant du monde partagé.

La figure 3.7 montre un exemple de contenu multi-utilisateurs conçu avec la proposition de [CC99]. Le fichier VRML 97 originel est presque intact, les seules modifications sont la gestion des routes (remplacement du mot clé `ROUTE` par `SHARED_ROUTE`).

```
#VRML V2.0 utf8

# World description
DEF T Transform {
 children [
  DEF CS CylinderSensor {},
  Shape {
   appearance Appearance { material Material { diffuseColor 1 0 0 }}
   geometry Box { size 1 1 1 }
}]}

# Routes description
# removed: ROUTE CS.rotation_changed TO T.rotation
SHARED_ROUTE CS.rotation_changed TO T.rotation
```

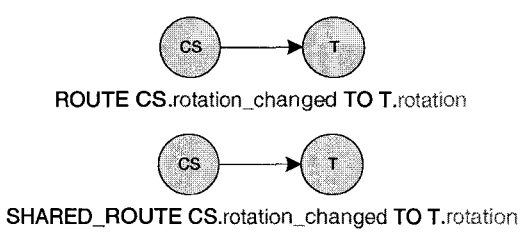


FIG. 3.7 – *Description d'un monde partagé avec la proposition de [CC99] (en gras, les parties provenant du contenu mono-utilisateur)*

Plutôt que d'opter pour le développement d'un navigateur VRML 97 étendu prenant en compte les extensions du langage, Carson a choisi d'utiliser un navigateur VRML classique pour visualiser le monde partagé : les différents fichiers VRML 97 "étendu" sont préprocessés par un

compilateur afin que le monde partagé soit compatible avec un navigateur VRML 97 classique. Le préprocesseur produit plusieurs fichiers de sorties :

- des fichiers de données 3D compatibles VRML 97, ils sont chargés par le navigateur VRML 97 classique,
- des fichiers sources Java implémentant les différentes routes partagées. Une fois ces sources compilées en applets, elles sont chargées par le navigateur HTML au chargement du monde partagé et communiquent avec le navigateur VRML 97 en utilisant l'EAI,
- des fichiers spécifiques permettant au *Gatekeeper* de connaître les différentes variables pour lesquelles il doit "espionner" leur état. Le *Gatekeeper* enregistre les modifications de ces variables ce qui lui permet de communiquer aux nouveaux arrivants l'état actuel de ces variables.

Un nouvel arrivant télécharge depuis un serveur web les fichiers de données 3D correspondant à la description du monde partagé tel qu'il était à l'origine. Ensuite, le nouvel arrivant se connecte au *Gatekeeper* via TCP/IP, ce dernier lui renvoie l'état courant des différentes valeurs partagées du monde, ce qui lui permet de synchroniser sa description du monde avec l'état courant du monde. Ensuite, en cours de session, la plate-forme de communication est de type multipoint : les communications entre les différents instance du monde s'effectuent par diffusion en utilisant le multicast IP non fiabilisé.

Le fait d'utiliser un préprocesseur pour le partage empêche tout partage dynamique pendant l'exécution de la session virtuelle : le partage est géré au moment de la compilation et de la génération des différents fichiers (applets Java et sources VRML 97). De plus, comme nous l'avons expliqué précédemment, il est toujours difficile de partager les sorties des *sensors* : cela nécessite l'utilisation de scripts et la configuration des *sensors* en mode relatif (cf. annexe C, page 240).

3.1.1.3 Troisième approche : description du partage avec un langage propriétaire

[BPPT01] propose de placer la description du partage en dehors de la description du monde. Le concepteur doit pour cela créer un fichier "`.sve`" contenant la description du partage : il doit identifier dans le fichier VRML 97 les routes devant être partagées, et, en utilisant une syntaxe propriétaire, différentes informations (champs de sortie et champs d'entrée de la route partagée) sont placées dans le fichier de partage. Des informations pour la gestion des verrous peuvent être également ajoutées, cela permet que les objets partagés ne soient manipulés que par un seul utilisateur à la fois.

La figure 3.8 montre un exemple de contenu multi-utilisateurs conçu avec la proposition de [BPPT01]. Le fichier VRML 97 originel est intact. Il est à noter que le champs *lock* permet de spécifier au système le verrou à utiliser - *la valeur -1 permet de désactiver la gestion de verrou pour l'évènement partagé, au contraire la valeur 0 permet de l'activer* -. En cours de session, si le champs *lock* a une valeur supérieure à 0, cette valeur correspond alors à l'identifiant de l'utilisateur qui possède le verrou. Comme avec le mécanisme de verrou proposé par la plate-forme technologique de Blaxxun, le mécanisme proposé ici est également associé à la donnée : à chaque modification de la donnée, il y a acquisition et libération du verrou.

```
#VRML V2.0 utf8

# World description
DEF T Transform {
 children [
  DEF CS CylinderSensor {},
  Shape {
   appearance Appearance { material Material { diffuseColor 1 0 0 }}
   geometry Box { size 1 1 1 }
}]}

# Routes description
ROUTE CS.rotation_changed TO T.rotation
```

```
node CS {
 lock 0
 SharedEventOut SFRotation rotation_changed
}

node T {
 SharedEventIn SFRotation rotation
}

ROUTE CS.rotation_changed TO T.rotation
```

FIG. 3.8 – *Description d'un monde partagé avec la proposition de [BPPT01] : sur la gauche le fichier VRML 97 décrivant le monde (aucune modification ne lui est apportée, en gras les parties provenant du contenu mono-utilisateur), sur la droite le fichier* `.sve` *de description du partage contenant les différentes routes à partager*

L'architecture de la plate-forme de communication est de type client/serveur. Les différents clients communiquent entre eux via le serveur (flux TCP/IP et UDP/IP). Le serveur offre différents services pour le support de la collaboration : un service audio permettant l'échange de données audio entre les clients, un service de chat permettant aux clients de s'échanger des données textuelles, et un service de gestion des objets partagés permettant aux clients d'assurer la synchronisation de l'état des objets 3D partagés présents dans le monde. Un protocole spécifique (très similaire à celui proposé dans VIP [Whi98]) a été mis en place et permet d'encoder les différentes données d'animation sur les objets 3D partagés. Du coté des clients, des applets Java reçoivent (émettent) des messages d'animation 3D du (vers le) serveur. La plate-forme repose sur un navigateur VRML 97 classique et les applets interagissent avec le navigateur en utilisant l'EAI (cf. figure 3.9). Contrairement à [CC99], la solution proposée ne nécessite pas l'utilisation de préprocesseur : des applets Java sont chargées d'interpréter le fichier de description de partage (`.sve`) et communique avec le contenu du navigateur VRML 97 classique en utilisant l'EAI.

À partir, des informations contenues dans le fichier `.sve` associé à une description VRML 97, les applets Java sont générées de façon automatique. Elles placent des observateurs sur les champs partagés en sortie (`SharedEventOut`) afin d'être prévenues des modifications opérées par l'utilisa-

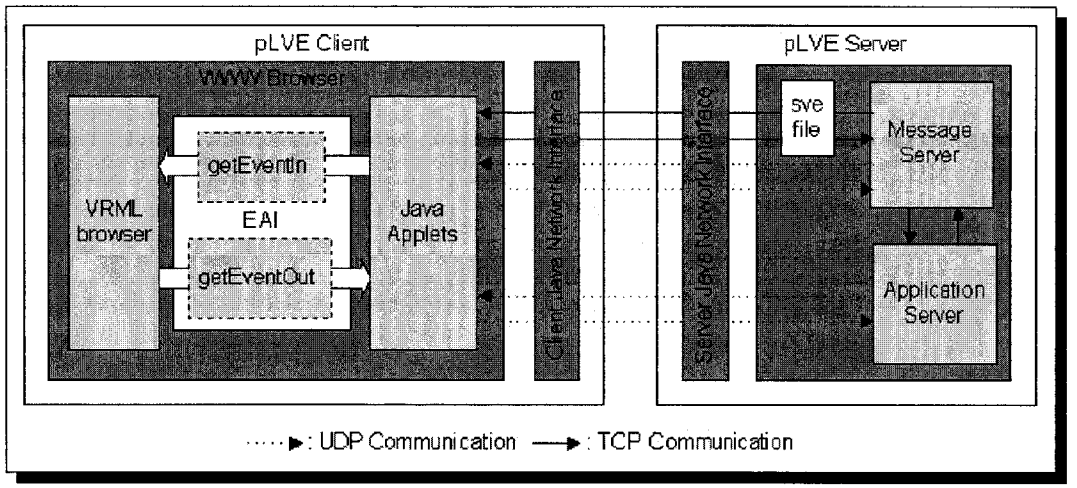


FIG. 3.9 – *Architecture de PLVE la plate-forme proposée par Bouras [BPPT01]*

teur. Après que l'applet ait vérifié qu'il n'y a pas de conflit d'accès, lorsqu'un champ de sortie est modifié, l'observateur de l'applet Java est prévenu, ce qui permet de transmettre la nouvelle valeur de ce champ au serveur. Le serveur est alors en charge de prévenir chaque client de la modification. À partir de la route partagée correspondante au champ de sortie modifié (`SharedEventOut`), l'applet Java est capable de modifier le champ d'entrée correspondant (`SharedEventIn`).

Cette approche ne permet pas de réaliser des mondes partagés dynamiques, ie. permettant l'ajout ou le retrait d'objets partagés : une phase de pré-compilation de la description du partage est nécessaire pour créer, de façon automatique, les différentes applets Java permettant de gérer le partage du monde.

Bilan

Nous avons présenté trois mécanismes permettant de concevoir des contenus multi-utilisateurs en VRML 97. Il nous parait intéressant de proposer de décrire, au moyen des fonctionnalités offertes par le langage VRML 97, tout ce qui concerne un objet 3D donné (sa géométrie, son interaction ainsi que la façon dont il est partagé) et de le placer dans un unique fichier. C'est pourquoi, nous préférons écarter les deux dernières approches. Cependant, nous pouvons regretter que les différentes propositions restent complexes à mettre en œuvre et à utiliser. En effet, des propositions multi-utilisateurs comme (Core) Living Worlds ou la solution de Blaxxun nécessitent d'excellentes compétences en VRML 97 de la part du concepteur. L'approche de VSPLUS est plus simple d'accès : le fait de placer la description du partage à l'extérieur de la description géométrique simplifie grandement la conception de contenus multi-utilisateurs. Néanmoins, toutes

ces propositions sont basées sur le même mécanisme de conception, le paradigme d'insertion : d'une part, le fait de casser les chemins d'animation existants implique un gros travail de la part du concepteur ; d'autre part, cela pose des problèmes de maintien de la cohérence au niveau des `Sensors`, ce qui n'est pas concevable dans une application de TCAO.

L'approche classique des mondes 3D multi-utilisateurs en VRML permet à plusieurs utilisateurs d'évoluer dans un univers virtuel 3D commun. Chaque participant dispose en local d'une copie du monde VRML mais la représentation du monde (ie. le graphe de scène) est unique, et chaque participant évolue dans ce graphe de scène commun. Cependant, comme nous l'avons dit précédemment, dans un système de TCAO, il est préférable de laisser l'utilisateur organiser son espace de travail comme il le souhaite : chaque utilisateur possède ainsi sa propre représentation mentale de la tâche à réaliser. Donc, dans un système de TCAO 3D, chaque utilisateur dispose de son propre graphe de scène. Un graphe de scène contient différents documents et le partage est placé au niveau de l'état de ces documents.

Les différentes plate-formes permettent de réaliser des communautés virtuelles 3D. Les utilisateurs peuvent dialoguer entre eux, ils y sont représentés sous la forme d'avatars 3D permettant de renforcer la sensation de présence. Cependant, les systèmes vus n'offrent que très peu d'interaction utilisateur/utilisateur et utilisateur/objets les rendant inutilisables comme support pour le TCAO. De plus, les plate-formes présentées utilisent une approche client/serveur ce qui réduit l'interactivité du système.

3.1.2 Le système MPEG-4

À la vue des possibilités multimédias actuelles, MPEG-4 [MP4] est l'évolution naturelle des précédentes normes du groupe de travail MPEG (*Moving Pictures Expert Group*). Elle répond aux besoins d'intégration des différents objets multimédias dans une même scène "audio-visuelle". La norme MPEG-4 permet l'encodage d'une scène multimédia dans le but de sa diffusion, mais offre également des mécanismes permettant d'interagir avec les différents objets diffusés. Elle permet la transmission (*streamed* ou *downloaded*) d'images et de sons, naturels ou synthétiques, d'objets 2D et 3D que l'on nomme AVOs (*Audio Visual Objects*).

MPEG-4 n'est pas qu'une simple définition de format de fichier mais une architecture logicielle complète pour la réalisation de mondes virtuels multimédia. Cette spécification repose sur :

- des méthodes de codage des AVOs,
- des méthodes de codage d'une scène multimédia complète composée d'AVOs agencés sous

la forme d'un graphe,

- le multiplexage et la synchronisation des différents flux de données contenant les AVOs,

- l'interface de communication entre les clients et le serveur MPEG-4.

L'architecture logicielle de la norme MPEG-4 peut être présentée comme un ensemble de trois couches (cf. figure 3.10) :

- la couche de distribution assure le transport des différents flux d'informations jusqu'à la couche applicative en faisant abstraction du type et du protocole du réseau sous-jacent. Elle comprend les fonctions de multiplexage et démultiplexage des différents flux élémentaires ainsi que le DMIF (*Delivery Multimedia Integration Framework*) permettant de gérer les flux. Le DMIF définit les interfaces de liaison avec le réseau, DNI (*DMIF Network Interface*), et avec l'application, DAI (*DMIF Application Interface*),

- la couche système donne accès aux mécanismes d'identification, de description et d'association des différents flux de données et permet la synchronisation entre les différents objets composant une scène (SL, *Synch-Layer*),

- la couche de compression effectue la compression/décompression de la scène et des différents objets la composant. La scène est représentée sous la forme d'un arbre de nœuds (comme en VRML), auquel ont été ajoutées des fonctionnalités pour la gestion des objets 2D et sonores. Le format XMT (*eXtensible MPEG-4 Textual*) permet de décrire une scène MPEG-4 en utilisant une syntaxe de type XML. La scène au format XMT est ensuite compressée sous un format binaire appelé BIFS (*BInary Format for Scenes*). La norme MPEG-4 définit également les protocoles *BIFS Update* et *BIFS Anim* qui permettent au serveur de prévenir les différents clients des modifications (ajout, retrait, changement d'état d'un nœud, etc) sur le graphe de scène afin de maintenir une vue cohérente du monde virtuel 3D pour tous les clients.

La norme MPEG-4 définit également un ensemble d'interfaces de programmation appelé MPEG-J permettant d'avoir un contrôle de la scène au travers d'une application Java. Les interfaces donnent accès au terminal, à la couche réseau, aux décodeurs et au graphe de scène. Pour la manipulation du graphe de scène, l'API proposée ressemble beaucoup à l'EAI définie dans la norme VRML 97.

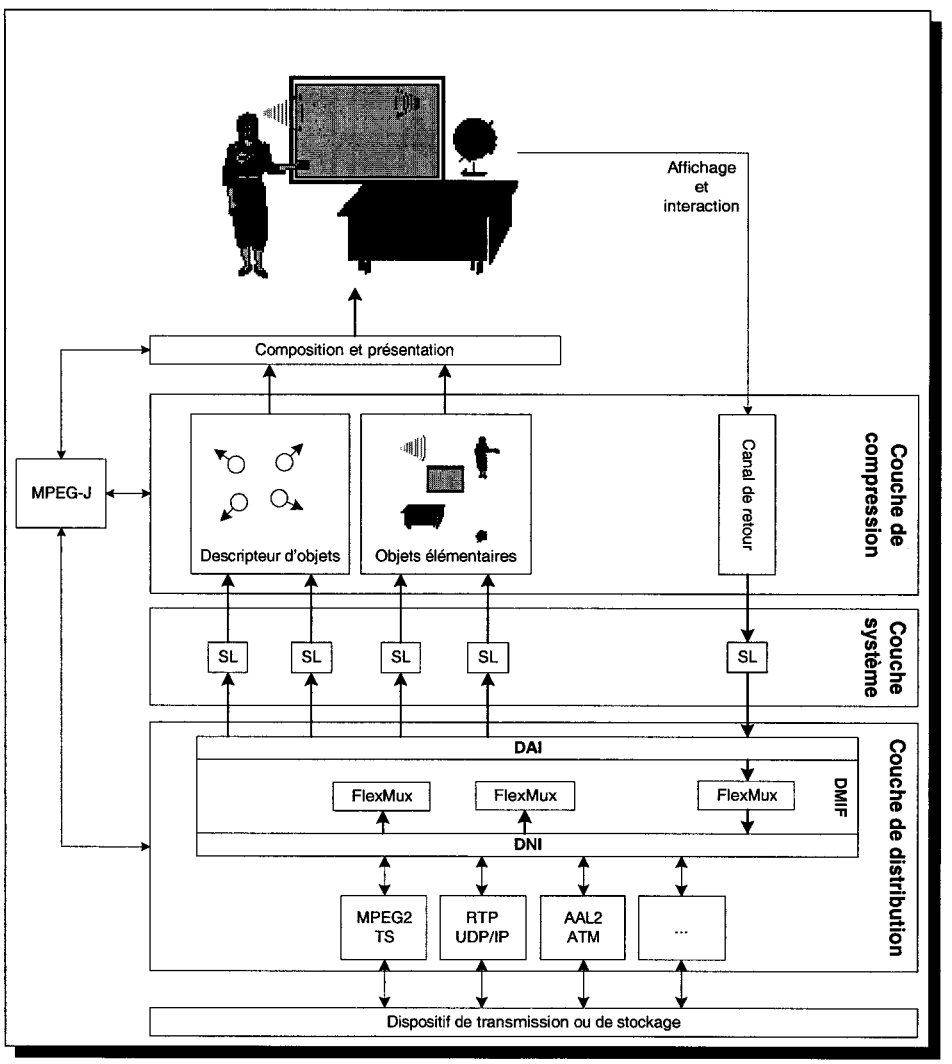


FIG. 3.10 – *Architecture d'un terminal MPEG-4*

Bilan

Cette nouvelle norme du MPEG semble, à première vue, s'adapter parfaitement à la création d'applications 3D intégrant différents documents multimédia. Cependant, elle a été conçue dans la suite logique de la norme de MPEG-2[33] et reste basée sur une approche de diffusion. La gestion de la scène est centralisée et un serveur est en charge de notifier les différents clients de l'évolution du monde 3D. En utilisant les notions vues dans le chapitre précédent, dans le principe, les communautés virtuelles développées au dessus de la plate-forme MPEG-4 reposent sur une base de données du monde 3D centralisée. Chaque client dispose d'une copie du monde 3D uniquement pour l'affichage. L'interaction se fait en envoyant des ordres de modifications au serveur via le canal de données (cf. figure 3.10), et le serveur distribue aux clients le résultat en utilisant les protocoles *BIFS Update* et *BIFS Anim*. Comme nous l'avons déjà dit précédemment, une telle approche réduit l'interactivité du système nécessaire à toute activité coopérative.

De plus, la norme reste relativement floue pour les personnes ne faisant pas partie du groupe d'experts. Les documents ne sont pas en libre accès. Il existerait[34] une extension multi-utilisateurs inspirée du framework Living Worlds et normalisée par le consortium MPEG-4, et qui serait implémentée dans le player MPEG-4 SoNG (*portalS of Next Generation*)[35].

3.2 Les outils Auteur

3.2.1 Muse

Créé en 1998, MuseCorp propose une plate-forme de développement de mondes 3D multi-utilisateurs, *Muse*[36]. Les mondes 3D de Muse peuvent être enrichis de différents types de médias classiques : audio, vidéo, documents 2D (présentation, page HTML, document flash, etc) et documents 3D. L'intégration de ces différents médias dans le monde 3D est complète : par exemple, alors que vous êtes dans le monde 3D, vous pouvez interagir sur ces médias (déplacer l'ascenseur d'une page HTML, remplir un formulaire web, faire une présentation Powerpoint, interagir avec les documents flash, etc). Plusieurs domaines d'applications peuvent être intéressés par la créa-

33. L'infrastructure MPEG-2 est utilisée par la télévision numérique.

34. L'utilisation du conditionnel étant de rigueur puisque aucune information officielle n'est disponible.

35. IST Project 10192 – *SoNG (portalS Of Next Generation)*, `http://www.song-portal.org`

36. `http://www.musecorp.com`

tion de mondes multi-utilisateurs avec la technologie Muse (cf. figure 3.11) : jeux, sites de vente sur Internet (tel que le site de vente *Amazon*), réunion de travail, etc. Plusieurs utilisateurs, représentées sous la forme d'avatars 3D, peuvent être présents dans le monde et interagir avec les différents éléments partagés, chaque modification effectuée par un utilisateur sur un élément partagé de son interface est transmise aux interfaces distantes. Cependant, seules les modifications effectuées sur les éléments pris en charge par Muse sont transmis.

Muse propose son propre visualiseur de mondes (*MuseLite*, disponible gratuitement). Sur son site, MuseCorp propose différents mondes 3D démontrant les capacités de sa plate-forme. Pour le développement d'applications, de composants spécifiques, Muse fournit un SDK (*Software Development Kit*). Le SDK fournit une interface pour le langage C++ et une interface pour le langage de Script TCL.

La technologie proposée par Muse est beaucoup plus aboutie que la solution proposée par Blaxxun : grâce aux fonctionnalités offertes par la plate-forme Windows de Microsoft et notamment la notion de composant *ActiveX*, le concepteur est capable d'intégrer à ses mondes 3D différentes applications existantes, telles que *Word*, *Powerpoint* ou un navigateur web. Là où Blaxxun se trouve confronté aux limites du langage VRML 97, Muse utilise pour sa part son propre format de données lui permettant d'enrichir beaucoup plus facilement les mondes 3D avec de nouveaux types de médias. Cependant, on regrettera le manque d'informations disponibles sur le format de fichier utilisé par Muse (données 3D, description du partage). De plus, le contenu des différents *ActiveX* (comme par exemple les ascenceurs de fenêtres ou le contenu d'un document *Word* ou *Powerpoint*) n'est pas synchronisé puisque cela n'est pas géré par Muse directement.

Avec des fonctionnalités techniques évoluées, Muse est bien plus qu'un logiciel de monde virtuel 3D. Le fait de pouvoir intégrer des applications 2D existantes rend Muse très attractif comme plate-forme de présentation virtuelle de documents multimédia (HTML, vidéo, powerpoint, etc).

Muse utilise une plate-forme de communication de type client/serveur : un serveur central est utilisé pour fournir aux nouveaux arrivants l'état courant du monde ; de plus, le serveur central est utilisé pour les communications entre les clients. De base, aucun contrôle d'accès aux données partagées n'est proposé : cela implique que deux utilisateurs peuvent manipuler en même temps une donnée partagée. L'architecture centralisée permet tout de même de conserver la cohérence du monde 3D. En effet, le serveur central traite les requêtes de modification les unes après les autres et diffuse les résultats. Cependant, les utilisateurs peuvent être perturbés par des incompréhensions dans leurs actions.

FIG. 3.11 – *Une présentation virtuelle avec la plate-forme Muse : en haut, la vue d'un spectateur (il visualise les transparents sur des écrans géants), et en bas, la vue du présentateur (il peut contrôler le défilement des transparents via une console de contrôle ; face à lui, les tribunes où sont présents les spectateurs), les transparents sont insérés dans le monde virtuel à l'aide d'un ActiveX Powerpoint*

3.2.2 Virtools

Virtools Dev[37] est une véritable plate-forme de développement pour la création d'applications interactives mixant la 3D temps réel, le son et la vidéo. Comme le montre la figure 3.12, Virtools Dev offre un IDE (Integrated Development Environment) qui peut être comparé aux autres environnements de développement classiques.

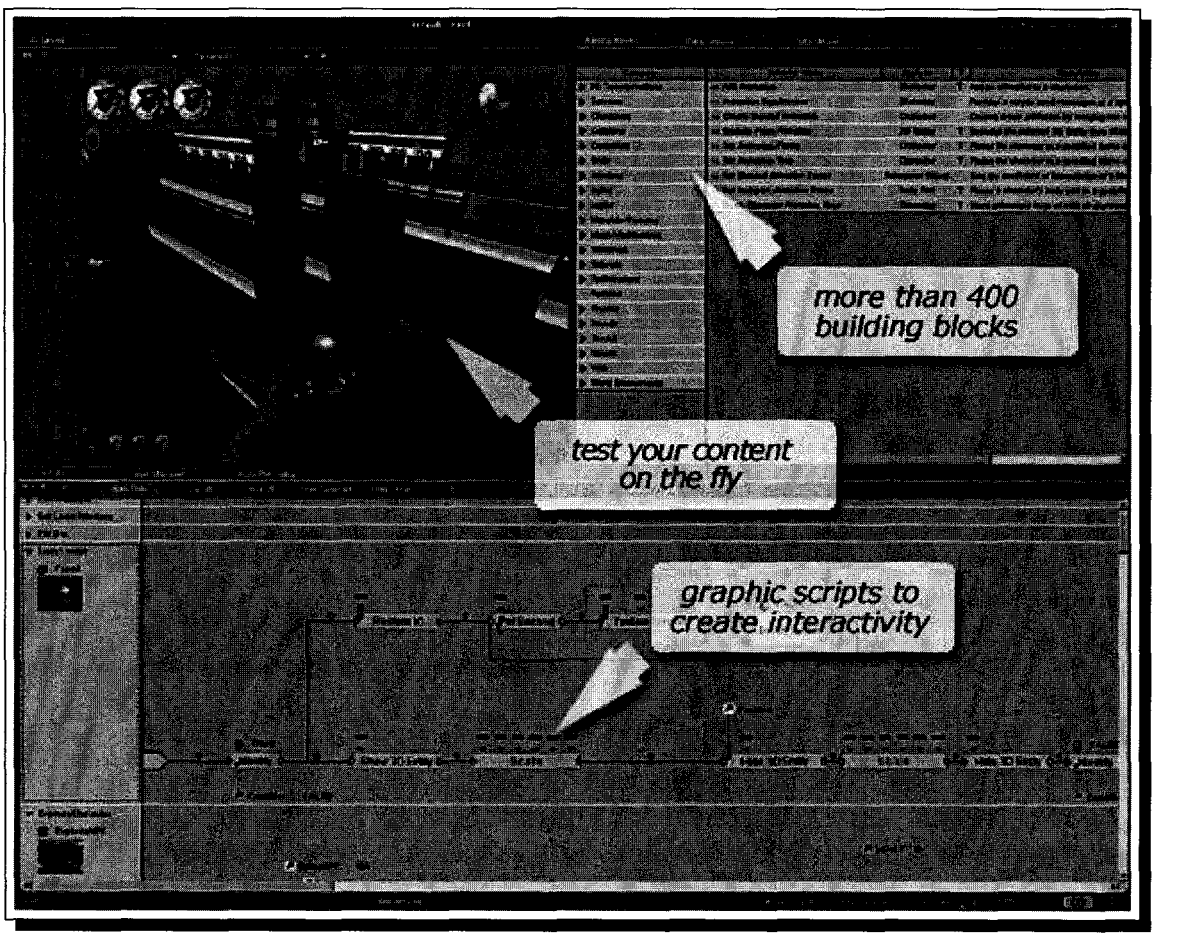


FIG. 3.12 – *Interface de développement Virtools Dev*

Tout le développement de l'application s'effectue au travers de cette interface. Le principe de "programmation" est simple : premièrement, il s'agit de placer dans le monde 3D différents objets 3D. Virtools n'est pas un modeleur 3D comme *3DS Max* ou *Maya* mais il permet d'importer dans l'interface de nombreux formats 3D. Virtools utilise un format propriétaire pour stocker les différentes données relatives au monde 3D (description géométrique, scripts d'interactions, etc). Ensuite le concepteur doit paramétrer la dynamique (interaction/comportement des objets) de son application : cela se fait à l'aide de Building Blocks (briques comportementales

37. `http://www.virtools.com`

prédéfinies, aux paramètres ajustables) que l'on assemble entre eux. Les Building Blocks (BB) peuvent répondre à des sollicitations d'autres BB ou aux valeurs de certaines variables. De base, de nombreux BB sont disponibles et couvrent un ensemble très large de besoins ; toutefois, si le concepteur a besoin de BB effectuant des traitements particuliers, il est capable de les développer en utilisant le SDK : bien entendu, l'utilisation de ce SDK nécessite des connaissances en programmation classique (ici, le langage utilisé est le C++).

Outre la maîtrise de l'interface de création - *qui n'est pas chose facile* -, la solution proposée nécessite des compétences en informatique minimales et permet donc à n'importe quelle personne (programmeur ou non) de développer très facilement son propre monde 3D interactif. Une fois la dynamique du monde 3D réalisée à l'aide de l'interface et différents BB, le concepteur exporte son application, Virtools Dev génère alors automatiquement le code de l'application.

Ensuite, tout comme pour VRML 97, les différentes applications développées peuvent être visualisées à l'aide d'un simple navigateur web auquel on a ajouté sous la forme d'un plug-in le player Virtools. Le moteur 3D de Virtools utilise les bibliothèques de rendu 3D bas-niveau (OpenGL ou Direct 3D).

Il est possible d'étendre les fonctionnalités du player en ajoutant plusieurs modules : on peut entre autre citer le *Physics Pack* permettant de gérer les propriétés physiques des objets (collisions entre objets, inertie des objets, gravité etc), l'*Artificial Intelligence Pack* ou bien le *Virtual Reality Pack* permettant de gérer de nouveaux périphériques d'entrée (comme par exemple les trackers 3D de type *flock-of-birds*) et de sorties (comme par exemple les casques de vision stéréo) ou le *Multi-user Pack*[38] permettant de créer des mondes multi-utilisateurs au dessus de la technologie Virtools.

La technologie multi-utilisateurs de Virtools propose une couche de haut niveau évitant au concepteur de manipuler directement les messages réseau : c'est ce que l'on appelle les *Distributed Objects* (DO). Un DO est un objet présent chez tous les participants et dont l'état est synchronisé (cf. figure 3.13). Le partage d'un objet avec la plate-forme Virtools se fait par la définition d'un ensemble de propriétés caractéristiques (ie. les DO). Là encore, aucune connaissance en programmation n'est requise, la définition des différents DO s'effectue au travers de l'interface de développement.

À plus bas niveau, pour synchroniser l'état du monde partagé, la plate-forme de communication utilise une technologie client-serveur : toutes les communications entre les clients doivent

38. `http://www.virtools.com/solutions/products/virtools_multiuser.asp`

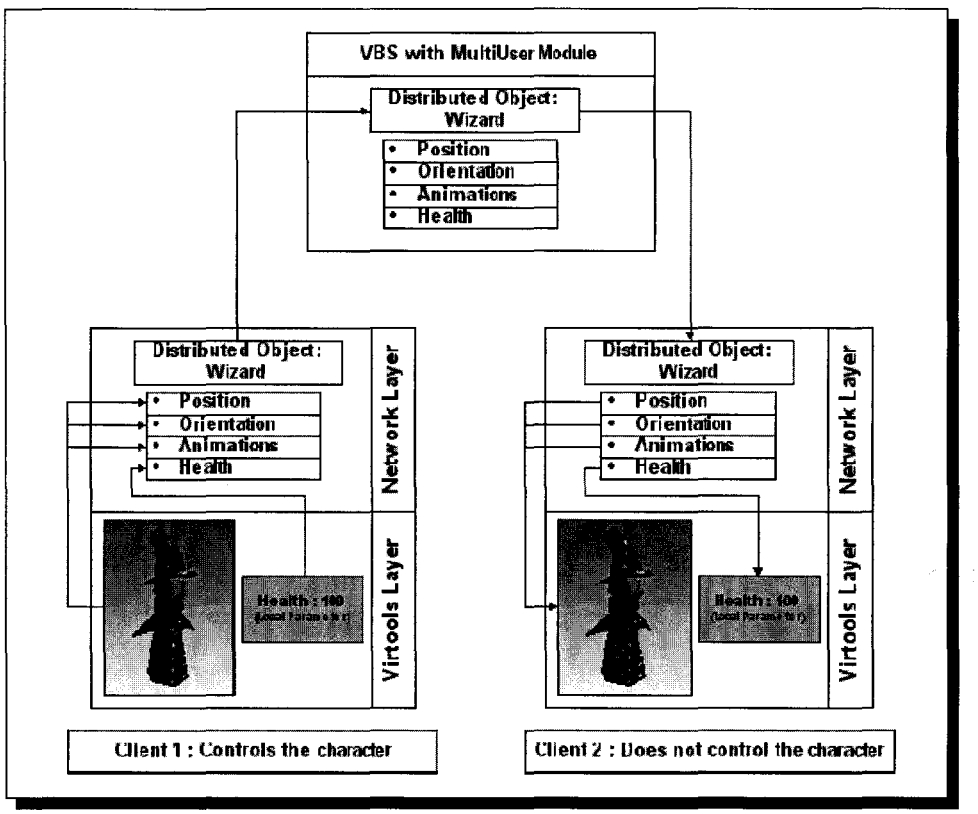


FIG. 3.13 – *Les Distributed Objects du multi-user pack de Virtools*

passer via le serveur. Cela permet de mettre en place, très facilement, un contrôle d'accès aux différents DO. Ainsi on a l'assurance qu'un DO n'est manipulé que par un seul utilisateur à la fois. Contrairement au mécanisme de verrou de Blaxxun qui propose de protéger la donnée partagée à chaque modification, le mécanisme proposé par Virtools est indépendant des modifications apportées aux données partagées, permettant ainsi d'éviter l'acquisition et le relâchement du verrou à chaque modification : le concepteur définit des verrous, ensuite des scripts sont chargés de l'acquisition des verrous. Une fois le verrou acquis, le système est libre de modifier la donnée jusqu'au relâchement du verrou. La plate-forme de communication permet de faire circuler différents types de messages entre les différents clients. Le concepteur du monde multi-utilisateurs peut choisir le mode de transport des différents messages :

- transport fiable : le message réseau est envoyé en utilisant le protocole TCP/IP assurant la réception du message et la conservation de l'ordre des messages.
- transport non fiable : le message réseau est envoyé en utilisant le protocole UDP/IP, la transmission est plus rapide mais l'on n'a la garantie ni de la réception, ni de l'ordre des messages.

3.2.3 Adobe Atmosphere

Depuis 2001, Adobe propose un système 3D interactif pour le web. Le système s'appelle *Atmosphere*[39]. Par les concepts proposés, le système d'Adobe se rapproche du système proposé par Virtools. Atmosphere se compose de différents éléments :

- *Adobe Atmosphere Builder*, un outil de création de mondes virtuels 3D Adobe Atmosphere,
- *Adobe Atmosphere Browser*, un navigateur 3D qui peut-être utilisé en *standalone* ou comme plug-in intégré dans un navigateur web classique,
- *Adobe Atmosphere Community Server*, un serveur de communication utilisant un protocole de communication spécifique (Atmosphere utilise une architecture client/serveur pour les communications).

Atmosphere utilise la technologie 3D propriétaire Metastream de Viewpoint[40] comme format de description de données 3D. Par conséquent, il bénéficie de leur technologie de *streaming* : le monde 3D n'a pas besoin d'être téléchargé entièrement pour être affiché, les différents éléments qui

39. `http://www.adobe.com/products/atmosphere`

40. `http://www.viewpoint.com`

le composent sont transmis au fur et à mesure (streaming) de la progression de l'utilisateur dans le monde. Le contenu 3D s'affine avec l'arrivée de nouvelles informations dans le flux, le navigateur reçoit en premier la géométrie, et ensuite les textures, graphiques, sons, etc. Grâce à des nouvelles techniques de compression (comme les *wavelet*), la quantité de données (géométries 3D, images et sons) à transmettre est très faible. De cette manière, cela permet à des usagers disposant d'une faible bande passante de visiter les mondes virtuels 3D Atmosphere. L'interactivité du monde est réalisée à l'aide de scripts (JavaScript).

Atmosphere utilise une architecture client/serveur pour la plate-forme de communication. Un serveur (*Adobe Atmosphere Community Server*) est en charge de gérer les entrées/sorties des participants et de faire communiquer les participants. Pour cela, un nouveau protocole de communication (*yacp*) a été développé pour Atmosphere se basant sur le protocole IRC. Il permet aux clients de communiquer avec le serveur : on distingue différents types de messages, des messages de chat et des messages permettant de mettre à jour la position des avatars dans le monde 3D. Aucune information n'est disponible sur le(s) canal(aux) de communication offert(s) par la plate-forme de communication pour l'échange des messages entre le serveur et les clients.

Avec la version actuelle d'Atmosphere, aucune interaction n'est possible avec l'environnement et les différents objets présents, ce qui, pour le moment, place cette plate-forme comme un logiciel de communauté virtuelle. Les utilisateurs peuvent uniquement se déplacer dans le monde 3D et dialoguer avec les autres utilisateurs qu'ils rencontrent, la 3D et les avatars sont utilisés comme support ludique à la communication (utilisation de postures prédéfinies).

Bilan

Les différents outils Auteur que nous venons de présenter proposent des fonctionnalités intéressantes, tout particulièrement le concept d'interface de développement qui facilite la tâche du concepteur et reste facile d'accès (après la maîtrise de l'interface). Cependant, comme nous l'avons déjà mentionné précédemment, là encore, il existe une seule représentation du monde 3D (ie. un unique graphe de scène) dont il faut synchroniser l'état en fonction des actions des utilisateurs. Il est difficile d'adapter les outils existants aux besoins spécifiques d'une interface de TCAO en terme d'organisation de l'espace de travail et d'interaction homme/machine et homme/homme. Les outils disponibles à l'heure actuelle ne sont pas ou peu extensibles au niveau de la présentation de l'interface et des mécanismes d'interaction, rendant leur utilisation très difficile dans le cadre d'une activité coopérative.

Ces outils Auteur restent cantonnés à la réalisation des mondes multi-utilisateurs sans réelle

coopération entre utilisateurs, comme les logiciels de communautés virtuelles sur Internet dont le but reste uniquement ludique. L'interaction entre les utilisateurs s'effectue essentiellement sous une forme textuelle. Il est à noter que la plate-forme Muse propose des fonctionnalités innovantes lui ouvrant d'autres portes que le simple logiciel support d'une communauté virtuelle sur Internet. L'intégration d'applications 2D existantes dans le monde virtuel 3D est un concept intéressant qui permet à ses utilisateurs de travailler ensemble à distance, comme par exemple dans le cadre de réunions de présentations virtuelles. Cependant, ce mécanisme montre ces limites : l'utilisateur se trouve confronté au problème d'interaction avec ces différents éléments 2D intégrés dans un monde 3D. La solution retenue est de basculer l'application existante au premier plan et d'interagir avec celle-ci avec un mécanisme classique 2D. L'inconvénient d'une telle approche est que l'utilisateur perd le contact avec les autres. D'autre part, Muse ne permet pas de synchroniser l'état interne de ces différents composants rendant difficile l'activité collaborative.

D'un point de vue ergonomique, le mode d'interaction des applications 2D n'est pas adapté à la 3D : ainsi, plutôt que de les intégrer dans l'environnement 3D, nous pensons qu'il est préférable de conserver leur fonctionnement 2D en proposant une interface de communication permettant à des applications "externes" (ie. s'exécutant en dehors du terminal de coopération) de manipuler les documents contenus dans le terminal. L'application externe est alors utilisée pour produire/modifier des documents, et le terminal de travail est utilisé pour coopérer autour de ces derniers.

3.3 Les boîtes à outils

3.3.1 DIVE

DIVE (*Distributed Interactive Virtual Environment*) [CH93a][CH93b] est un système d'EVC 3D développé par le SICS (*Swedish Institute of Computer Science*). Il permet à plusieurs utilisateurs, représentés sous la forme d'avatars 3D, d'explorer un espace 3D et d'interagir les uns avec les autres (cf. figure 3.14). DIVE se veut être une plate-forme pour le développement de monde virtuel 3D collaboratif [CH93b]. Le système fournit un ensemble de fonctions et le développeur doit spécifier un ensemble de fonctions *callback* permettant de gérer la dynamique de son application (interaction, animation, etc.). D'autre part, pour faciliter l'écriture de l'application, DIVE propose une interface de scripts (TCL) : les différents scripts sont déclenchés par des messages provenant de l'interaction utilisateur, de tops d'horloge ou encore de collisions.

L'environnement DIVE s'articule autour d'une base de données géométrique partagée. Chaque

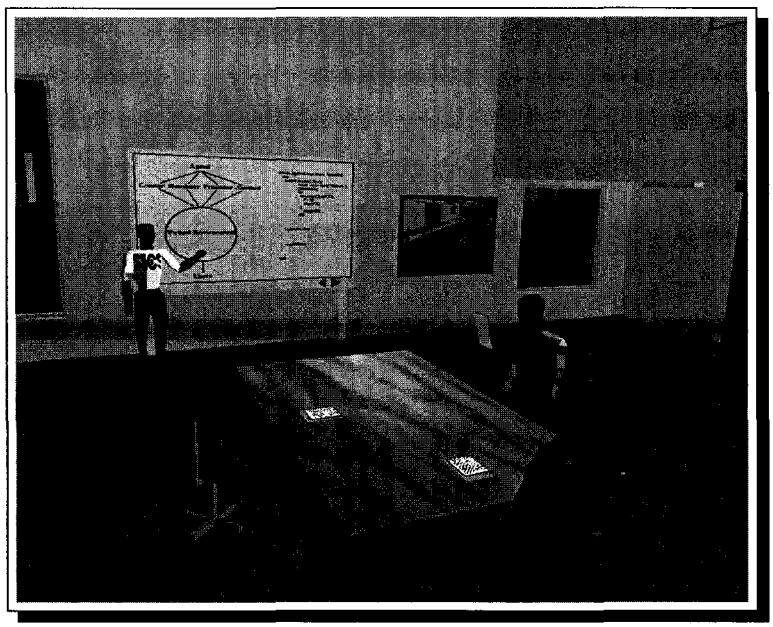


FIG. 3.14 – *L'EVC 3D DIVE développé par le SICS*

utilisateur dispose sur sa machine d'une copie de la base de données du monde 3D. Son état est maintenu cohérent par l'utilisation d'une plate-forme de communication ne reposant sur aucun serveur centralisé. Toute modification effectuée sur un objet partagé est transmise aux copies distantes du même objet. Dans DIVE, à chaque objet partagé, on identifie un groupe : chaque utilisateur dispose de sa propre copie de l'objet partagé et l'ensemble des copies constitue le groupe logique. Du point de vue de l'application, la réplication des données partagées est transparente : dès qu'un membre du groupe est modifié, les autres copies sont également prévenues du changement.

Les communications entre les différents clients se font en utilisant un canal de communication multipoint fiable. Une abstraction de haut niveau de la couche de communication est proposée dans DIVE permettant d'utiliser différentes couches de transport. Ainsi, DIVE propose l'utilisation de ISIS [HHBC92], une boîte à outils générique pour le développement d'applications distribuées, et SID (*SICS Distribution Package*) [Hag95], une couche de transport ad'hoc développée dans le cadre de DIVE. Deux versions de SID ont été développées : une première version utilisant un mécanisme à base d'acquittements positifs (`ACK`), posant des problèmes de scalabilité, et une deuxième version basée sur un mécanisme à base d'acquittements négatifs (`NACK`) visant à réduire les problèmes de scalabilité. SID est une bibliothèque de fonctions permettant le transport de messages en multipoint de façon fiable. La distribution des données combinée à la plate-forme de communication permettent de simuler une mémoire partagée et distribuée. Pour réduire la charge réseau, DIVE implémente le modèle spatial d'interaction (cf. 2.1.1, page 41). Un serveur est en charge "d'espionner" les différentes actions des différents clients afin de gérer

la persistance du monde virtuel et d'offrir une version actualisée du monde virtuel aux nouveaux arrivants.

DIVE propose également un mécanisme à base de verrou [CH93a] permettant une exclusion mutuelle. Le mécanisme utilise l'approche pessimiste que nous avons présentée dans le chapitre précédent (cf. 2.1.5, page 47).

3.3.2 WorldToolKit

WorldToolKit[41] proposé par Sense8 est une boite à outils existant sous de nombreuses plate-formes (Windows et Unix) regroupant un ensemble de fonctions permettant de développer très facilement une application de réalité virtuelle 3D en masquant les difficultés techniques. WorldToolKit autorise une grande variété de périphériques pour le rendu et pour l'interaction.

La bibliothèque regroupe un ensemble de classes (`Nodes`, `Lights`, `Sensors`, `Sounds`, etc). A l'aide des différentes classes et des fonctions fournies par la bibliothèque, le concepteur est capable de construire de façon logicielle le graphe de scène correspondant au monde virtuel, un graphe semblable à ce que permet de faire le langage VRML 97. Une fois le graphe de scène créé, le concepteur donne la main à la boucle principale de la bibliothèque, le système prend alors en charge l'évolution du monde virtuel 3D (interaction, affichage).

Il est à noter que Sense8 propose World Up une interface de développement pour la boîte à outils WorldToolKit. World Up propose une l'interface de développement très proche de celle offerte par Virtools.

Sense8 propose également World2World, une solution permettant le développement d'applications de réalité virtuelle multi-utilisateurs. Il étend les fonctionnalités de la boîte à outils de base en ajoutant des fonctions permettant la connexion/déconnexion au serveur, la déclaration de propriétés partagées, l'observation de celles-ci lorsqu'elles sont modifiées, la gestion des accès concurrents, etc. World2World intègre une plate-forme de communication ad-hoc basée sur une topologie client/serveur permettant d'échanger des messages entre les clients et le serveur.

[VDMKCS+99] propose un EVC basé sur WorldToolKit avec la particularité de proposer une plate-forme de communication de type client/serveur en s'appuyant sur l'intergiciel CORBA. Un serveur d'objets est en charge de stocker les positions 3D de chaque utilisateur et de les fournir

41. `http://www.sense8.com`

à chaque client. En utilisant une invocation de méthode, chaque client est capable d'entrer dans le monde virtuel partagé, d'obtenir la position des autres participants et enfin de quitter le monde virtuel. Cependant, une telle approche montre ses limites du fait du choix d'une topologie client/serveur et de l'utilisation du protocole IIOP inadapté aux communications de groupe (de par l'utilisation de TCP/IP et du point à point). Pour améliorer le système et réduire les temps de réponse, [VDMKCS+99] propose l'utilisation du multipoint sans donner de solutions.

3.3.3 SVE

Simple Virtual Environment (SVE)[42] [KBH00] est une bibliothèque permettant de créer des applications de réalité virtuelle 3D très facilement. SVE propose un ensemble de fonctions permettant de gérer la définition hiérarchique des objets dans l'environnement virtuel 3D et des fonctions de callback pour le rendu, l'interaction, etc.

Au dessus de SVE, Linebarger a défini *Shared Simple Virtual Environment* (SSVE) [LJK03] permettant de concevoir des mondes virtuels 3D multi-utilisateurs avec la bibliothèque SVE. Aucune information n'est disponible pour la conception des données partagées. Linebarger s'est intéressé à la conception de la plate-forme de communication de SSVE: il propose d'utiliser l'intergiciel CORBA. Plus particulièrement, il propose d'utiliser une combinaison de MIOP et IIOP pour supporter les communications entre les différentes instances du monde virtuel. Linebarger distingue deux canaux de communication véhiculant deux types de données. La couche IIOP (TCP/IP) est utilisée pour le transport des évènements significatifs de l'interface comme les entrées/sorties des utilisateurs, la gestion des verrous. La couche MIOP (IP multipoint non fiable) est utilisée pour mettre à jour fréquemment l'état des objets partagés (position, orientation). Cependant, dans SSVE, CORBA est uniquement utilisé pour passer des messages entre les machines (cf. figure 3.15). Le mécanisme mis en place ne propose pas la notion de données partagées et de groupes d'objets dupliqués au sens de Hagsand que nous avons présenté dans le chapitre précédent (cf. 2.2.3.3.3, page 62). L'approche de la plate-forme de communication de SSVE se rapproche plus de la philosophie d'un mécanisme par passage de messages : la technologie CORBA lui fournit une infrastructure de haut niveau pour l'échange de messages entre différents processus (les objets `SSVEApplicationThread` et `ORBThread` sur la figure 3.15). La plate-forme de communication permet également de gérer les accès concurrents aux objets partagés, cependant aucune information sur l'algorithme utilisé n'est disponible.

42. `http://www.eecs.lehigh.edu/GIVE/`

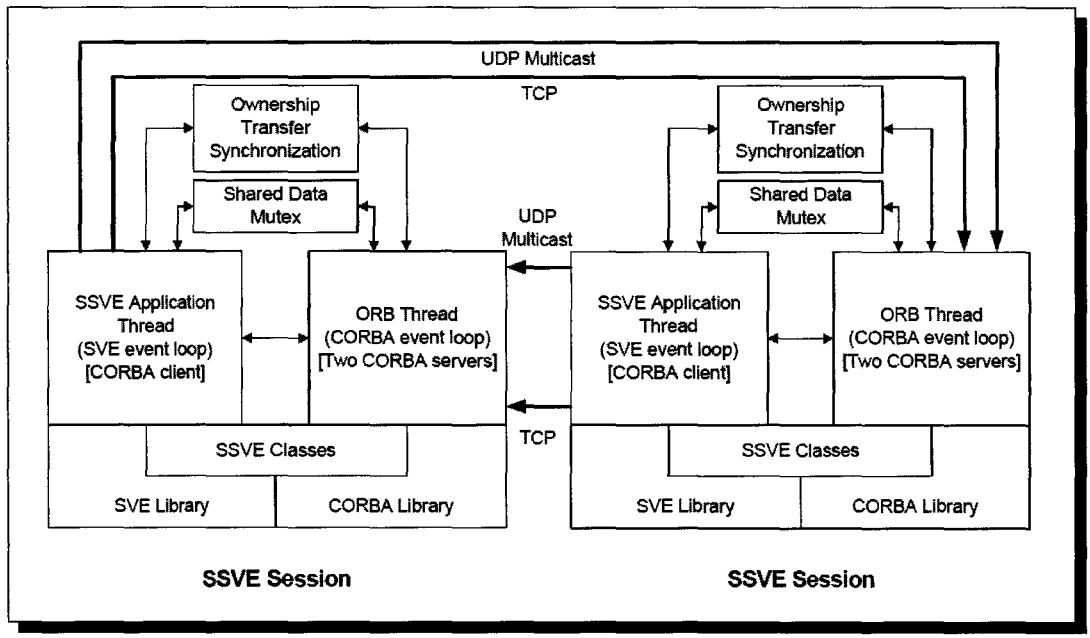


FIG. 3.15 – *Architecture de la plate-forme de communication de SSVE [LJK03]*

3.3.4 OpenMask

OpenMask[43] [Mar01][MAC+02], issu d'un développement de l'IRISA, est une boîte à outils pour le développement et l'exécution d'applications de simulation et d'animation en réalité virtuelle. Le noyau de la plate-forme OpenMask peut être vu comme un intergiciel fournissant une abstraction de l'environnement réel d'exécution (*multi-threadé*, distribué, etc). Le noyau permet d'accueillir les différents objets de simulation developpés par le concepteur de la simulation. Il offre également un ensemble de services de gestion de la simulation : service de nommage des objets, activation des objets, communication entre les objets suivant le modèle producteur/consommateur, gestion du temps. Dans le cadre d'une simulation distribuée sur un *cluster* de machines, OpenMask utilise une couche de communication basée sur la bibliothèque de programmation distribuée PVM afin de gérer la distribution des objets de simulation. De manière transparente, un mécanisme de *référentiel/miroirs*, très similaire au concept de *pilot/drones* introduit par Living Worlds, permet de masquer la distribution des objets de simulation : un objet de simulation référentiel s'exécute sur une seule machine et possède des miroirs sur les autres machines pour permettre la communication de cet objet avec les autres, indépendamment de l'endroit où il s'exécute (cf. figure 3.16). PVM permet de synchroniser l'état des miroirs avec celui du référentiel. Des "polateurs" (techniques de *dead-reckoning*) permettent de faire interagir des objets simulés à des fréquences différentes, et également, de limiter le débit réseau (PVM

43. `http://www.openmask.org`

utilise uniquement TCP/IP ce qui entraîne un surcoût à chaque communication).

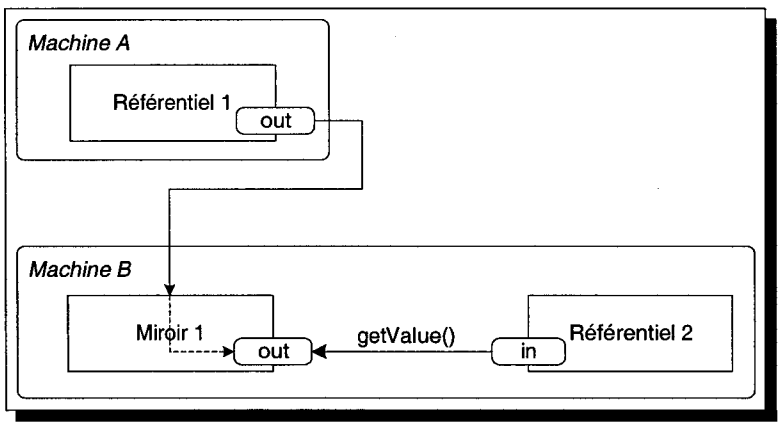


FIG. 3.16 – *Distribution des objets de simulation dans la plate-forme OpenMask suivant le modèle de référentiel/miroirs*

La plate-forme logicielle OpenMask offre également la possibilité de visualiser une scène 3D (au format Performer de SGI, un portage sur OpenSG[44] est en cours de réalisation), et permet à un ou plusieurs opérateurs d'interagir avec l'ensemble de la simulation. Le visualiseur 3D est un objet de simulation adaptable en fonction du périphérique de sortie (mur d'écrans, vision en stéréo, etc) communiquant avec les autres objets de simulation.

La version distribuée d'OpenMask permet de mettre en œuvre des applications coopératives en autorisant plusieurs participants à interagir au sein d'un même environnement 3D. La coopération est obtenue très simplement en instanciant un objet de visualisation interactive sur chacune des machines. Contrairement aux mondes massivement multi-utilisateurs, OpenMask vise plutôt les réunions coopératives de petit groupe. Pour supporter la coopération, les utilisateurs sont représentés sous la forme d'avatars 3D et des mécanismes d'interaction de haut niveau sont mis en œuvre afin de percevoir ses propres actions ainsi que celles des autres. Ces différents travaux ont été implanté dans le projet de plate-forme française de réalité virtuelle PerfRV[45].

Dans le cadre d'une application distribuée conçue avec OpenMask, comme nous l'avons mentionné précédemment, le mécanisme de distribution utilise le concept de référentiel/miroirs. Ainsi, l'interaction avec des entités non présentes en local s'effectue alors à distance au travers de l'objet de simulation miroir [DM96], ce qui introduit un délai dans l'interaction utilisateur. Dans le cadre du projet PerfRV, une application de "revue numérique" coopérative automobile a été mise en place entre deux sites distants (cf. figure 3.17), cependant il est à noter que le réseau utilisé

44. `http://www.opensg.org`

45. `http://www.perfrv.org`

est de type VTHD (*Vraiment Très Haut Débit*) et que donc la latence du réseau est minime. Aucune information n'est disponible sur la solution utilisée pour gérer les accès concurrents aux objets partagés de l'interface de travail (peut-être l'utilisation d'un objet de simulation spécifique auquel ce rôle aurait été attribué?).

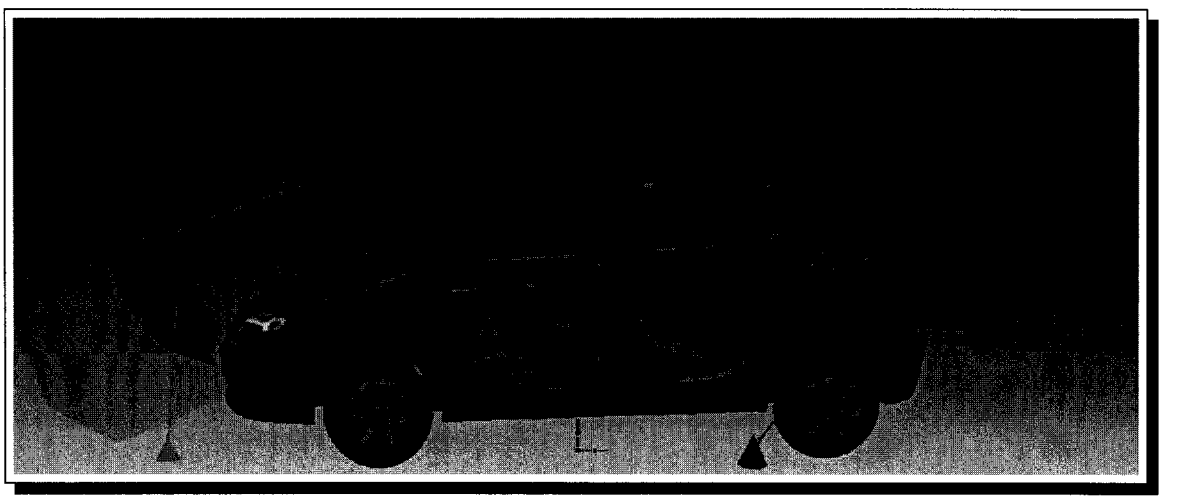


FIG. 3.17 – *Application de "revue numérique" coopérative avec la plate-forme logicielle OpenMask*

3.3.5 SIMNET, NPSNET, DIS et HLA

Conçu pour la DARPA (*Defense Advanced Research Projects Agency*), SIMNET [CDG+93], le premier environnement virtuel collaboratif 3D, est apparu au milieu des années 80. Cet EVC 3D est un système de simulation permettant aux militaires de mettre au point leurs tactiques durant leur période d'entraînement. Le principe de SIMNET reste très simple : chaque site participant à la simulation distribuée diffuse de manière non fiable à tous les autres la position et la vitesse des entités virtuelles qu'il simule, ainsi que la position et les effets des munitions tirées depuis les entités simulés. SIMNET s'appuie sur le protocole DIS (*Distributed Interactive Simulation*) [IST95][Loc95]. Ce standard IEEE propose une spécification du format des messages (appelés PDU, *Protocol Data Unit*) échangés par les différentes machines pendant la simulation. DIS propose uniquement le codage des messages, et reste indépendant du moyen de communication utilisé. Afin de réduire la charge réseau, DIS a été le premier à proposer la notion de *dead-reckoning*. SIMNET proposait une plate-forme de communication à diffusion inadaptée à la simulation distribuée à grande échelle.

NPSNET [MZP+94] s'inscrit dans la continuité de SIMNET en complétant ou étendant SIMNET. Les travaux de Macedonia ont visé à réduire la charge réseau de l'EVC 3D. Son approche a consisté à coupler un partitionnement spatial de l'espace virtuel, une division fonctionnelle des

entités simulées avec l'utilisation d'adresses multipoint. Les entités n'ont plus alors qu'à diffuser les informations qui les concernent sur les adresses de groupe multipoint auxquelles elles appartiennent directement. Le gestionnaire d'intérêt (AOIM, *Area of Interest Manager*) est en charge de déterminer les adresses multipoint à écouter.

Dans la continuité de DIS, le DMSO américain (*Defence Modelling and Simulation Office*) a proposé en 1996 le HLA (*High Level Architecture*) [HLA00]. Cependant, là où DIS n'était qu'un protocole réseau spécifiant le format des paquets échangés, HLA propose une architecture de haut niveau adaptée aux simulations militaires. L'architecture s'appuie sur :

- un ensemble de règles de conception,
- un méta modèle de description des fédérés, ie. des objets, (OMT, *Object Model Template*) présents dans une fédération (ie. une simulation) et la façon dont ils interagissent entre eux,
- la RTI (*Run Time Infrastructure*), une couche logicielle définissant un ensemble de services permettant de gérer la fédération et la communication entre les fédérés,
- une spécification d'interface qui décrit comment les fédérés interagissent avec la RTI.

De notre point de vue, HLA peut être comparée à l'architecture CORBA que nous avons présenté dans le chapitre précédent. En effet, nous retrouvons les mêmes composants : l'infrastructure d'exécution RTI peut être comparé à l'ORB et le langage OMT peut être comparé au langage IDL.

SIMNET, NPSNET et HLA permettent uniquement de réaliser des simulations : il n'existe qu'une seule instance de chaque objet simulé communiquant avec les autres objets via la RTI. Dans le cadre d'une simulation militaire, un utilisateur pilote un véhicule (représenté par un fédéré dans HLA) simulé sur l'ordinateur local. Il n'existe aucune interaction directe avec les autres objets simulés : lorsqu'un objet simulé localement a besoin d'un objet simulé distant, il utilise le miroir local de l'objet distant. Chaque objet simulé met à jour régulièrement ses différents miroirs distants. Une telle approche est inconcevable dans une application de TCAO où les utilisateurs sont susceptibles de manipuler directement et de façon interactive tous les objets présents dans l'interface.

De plus, HLA n'est pas destiné à simuler des environnements pouvant être modifiés de manière dynamique. Dans un premier temps, la simulation est préparée, ie. on met en place les différents fédérés, puis dans un second temps, la simulation est effectivement démarrée, et l'on suit l'évolution du système.

Bilan

Les boites à outils comme DIVE, WorldToolKit ou SVE permettent la réalisation de communauté virtuelle sur Internet. Ici encore, il est difficile d'adapter ces outils aux besoins spécifiques d'une interface de TCAO tant du point de vue de l'organisation de l'interface et des mécanismes d'interaction, que du point de vue des techniques de communication réseau. Pour faciliter l'utilisabilité de la boite à outils, les mécanismes de communications réseaux sont cachés au concepteur d'application.

Les boites à outils comme OpenMask, SIMNET/NPSNET ou HLA permettent la réalisation de simulation distribuée sur un ensemble de machines reliées en réseau. L'approche adoptée de référentiel/miroir est assimilable à une architecture client/serveur : un objet n'est simulé que sur une seule machine, et est représenté, sur les autres machines de la simulation, par un objet miroir mis à jour régulièrement. Dès lors que l'on place un utilisateur par machine, et que celui-ci souhaite manipuler un objet non présent localement, l'interaction s'effectue alors à distance, introduisant ainsi un délai dans l'interaction utilisateur.

3.4 Conclusions

Dans ce chapitre, nous avons présenté plusieurs systèmes permettant de concevoir des environnements virtuels 3D multi-utilisateurs. Nous devons distinguer deux niveaux de conception dans notre terminal de travail :

- le niveau du concepteur du terminal de travail coopératif,
- le niveau du concepteur d'applications coopératives utilisant le terminal de travail.

Pour la conception des applications coopératives, nous proposons d'utiliser un système descriptif, ce qui est simple d'accès même pour des personnes novices en programmation. Le concepteur crée les documents 3D qui peuvent être directement intégrés dans l'interface de travail. Il nous semble que le format VRML 97 soit le plus adapté pour décrire les différents documents 3D présents dans l'interface. En effet, le format VRML 97 permet de décrire la géométrie 3D et l'interaction correspondante dans un unique fichier. Nous avons présenté des extensions au langage VRML 97 qui permettent également de placer la description de partage dans le même fichier que la description de la géométrie et de l'interaction. Cependant, les extensions multi-utilisateurs proposées restent complexes à utiliser : du fait de l'utilisation du paradigme d'insertion, la réalisation de contenus multi-utilisateurs à partir de contenus mono-utilisateur déjà existants implique

un gros travail de la part du concepteur.

Pour la conception du terminal de travail coopératif, nous ne pouvons pas utiliser les différents systèmes présentés précédemment. En effet, toutes les plate-formes d'environnements virtuels 3D que nous avons vu permettent de créer des communautés virtuelles 3D sur Internet regroupant un grand nombre de participants. Chaque participant dispose en local d'une copie du monde virtuel. Le graphe de scène est unique, et chaque participant évolue dans ce graphe de scène commun. Dans un système de TCAO, chaque utilisateur possède ainsi sa propre représentation mentale de la tâche à réaliser, il est donc préférable de laisser l'utilisateur organiser son espace de travail comme il le souhaite. C'est pourquoi, chaque utilisateur dispose de son propre graphe de scène contenant les différents documents partagés. De plus, ces différents outils sont très peu extensibles en terme d'interaction : l'ajout de nouvelles fonctionnalités d'interaction, comme par exemple des indices visuels de retour (comme des boites englobantes autour des objets), n'est pas très trivial voir même impossible.

D'autres outils comme OpenMask, SIMNET, NPSNET et HLA sont avant tout conçu pour la simulation distribuée. Chaque objet simulé est présent uniquement sur une seule machine, chaque machine participant à la simulation, étant en charge de gérer (ie. simuler) un (ou plusieurs) objet(s). Pour gérer les dépendances entre un objet et d'autres objets distants, le concept de *réferentiel/miroir* est utilisé : un objet simulé possède des miroirs sur les autres machines, mis à jour régulièrement. Lorsque l'utilisateur souhaite manipuler un objet non simulé localement sur la machine, l'interaction s'effectue alors à distance, au travers de son miroir, ce qui a pour conséquence d'introduire de la latence nuisible à l'activité. Pour cette raison, il nous est difficile d'utiliser de tels systèmes de simulation distribuée comme support pour notre terminal de TCAO synchrone

Chapitre 4

Spin : un EVC 3D pour le TCAO synchrone

Sommaire

Dans ce chapitre, nous présentons en détail le projet Spin dans sa globalité. Dans une première partie, nous présentons la première version du terminal de travail, à la fois sous des aspects Interface Homme Machine (organisation de l'espace de travail, interaction homme-machine, représentation des intervenants, etc), et également, sous des aspects technologiques. Ce premier prototype a démontré la faisabilité d'une interface 3D pour le travail coopératif. Cependant, la réalisation d'applications coopératives avec le premier prototype Spin reste difficile, c'est pourquoi, une nouvelle version du terminal, Spin-3D, conservant les principes de l'interface 3D, doit être proposée. Dans une deuxième partie de ce chapitre, nous présentons les différents aspects technologiques de cette nouvelle version du terminal de travail coopératif, ie. la méthode de description du partage de données et la plate-forme de communication supportant la distribution des utilisateurs.

Le projet d'EVC 3D Spin est issu des réflexions menées depuis 1994 par l'équipe Graphix du LIFL (qui travaille sur la synthèse d'image temps réel) conjointement avec l'équipe Trigone du CUEEP (qui travaille sur le travail coopératif). Dans un premier temps, il visait à proposer de nouveaux concepts d'interfaces pour le travail coopératif synchrone. En effet, les interfaces 2D "classiques" de type WIMP (Window Icon Mouse Pointer, concept d'interface issu des travaux de recherche menés par Xerox), très performantes quand il s'agit de travailler seul devant son écran, montrent leurs limites lors de la réalisation d'une activité coopérative.

Le projet a été soutenu par le CNET (Centre National d'Etude des Télécommunications, renommé France Télécom R&D) dans le cadre d'un contrat intitulé "Interactions hommes-machines multidimensionnelles, multimédia et multiparties pour les services de groupe : conception et évaluation" et par la région Nord-Pas de Calais dans le cadre de Ganymède (contrat de plan Etat-Région) autour du thème "Communication Avancée", et plus particulièrement "Travail Coopératif Assisté par Ordinateur". De 1994 à 1999 de nombreuses actions ont été menées, dont le résultat est la réalisation d'un prototype d'interface tridimensionnelle et multimédia pour le travail de groupe.

De nombreux thèmes de recherche sont couverts par le projet :

- des domaines très techniques comme ceux des réseaux et des protocoles de communication, et des techniques de conception de logiciels,
- des interfaces homme-machines,
- des sciences humaines et cognitives,
- des sciences sociales et de l'analyse des organisations.

Nous nous intéressons à la partie synchrone de l'activité coopérative distante, i.e. la réunion.

Nous nous plaçons dans le cadre de réunions de petits groupes. D'un point de vue ergonomique, seules les situations de petit groupe (quatre ou cinq personnes) permettent une réelle coopération entre les utilisateurs. Les différents protagonistes de la réunion, éloignés géographiquement, sont placés devant leur machine exécutant le terminal Spin. Il s'agit de recréer virtuellement, au travers de l'interface, les conditions de la réunion réelle. Les utilisateurs du système doivent être capable d'interagir sur des documents afin de les présenter, de les modifier ou même d'en produire de nouveaux. L'interface doit fournir un maximum d'informations permettant à ses utilisateurs de comprendre l'activité, et oubliant ainsi la médiatisation de la réunion.

D'un point de vue équipement, nous nous plaçons dans le cadre de réunions non-immersives : l'utilisateur est dans son environnement de travail bureautique classique, et il ne doit pas être coupé des autres outils classiques de communication (comme par exemple le téléphone) et de l'environnement réel. De même, le matériel informatique permettant d'exécuter le terminal de travail coopératif Spin doit rester standard : les ordinateurs disponibles pour le grand public doivent être capables d'exécuter le terminal Spin à une vitesse d'au moins 10 images par seconde. En ce qui concerne l'interaction, nous évitons d'équiper "lourdement" l'utilisateur, notamment avec l'utilisation de capteurs de position du type "gants de données" : l'utilisateur doit rester libre de ses mouvements, par exemple pour répondre au téléphone, ou bien pour s'absenter quelques instants. Nous nous limitons donc à des périphériques adaptés à l'interaction 3D mais restant simples d'accès comme par exemple la *SpaceMouse*[46].

Nous allons maintenant présenter succinctement les différents concepts de l'interface de travail de Spin. Toutes les propositions reposent sur les concepts de bases du TCAO présentés précédemment (cf. 1.2.2, page 29) et ont été mises en oeuvre dans la première version de Spin. Ce prototype a permis de valider les différents concepts de l'interface de travail 3D.

4.1 Aspects Interface Homme Machine

4.1.1 L'interface utilisateur

A la vue des concepts de base de la médiatisation de l'activité coopérative synchrone identifiés précédemment (cf. 1.2.2, page 29), [Sau98] a formulé différentes recommandations pour la réalisation d'une interface 3D pour le TCAO synchrone comme par exemple la conservation du contact visuel, la conscience du regard, la présentation de l'interaction personne-personne et

46. `http://www.logicad3d.com`

personne-document, la qualité du canal sonore ou bien encore l'animation temps réel. Ces différentes recommandations ont permis de définir l'interface de Spin telle qu'elle est aujourd'hui.

4.1.2 L'organisation spatiale

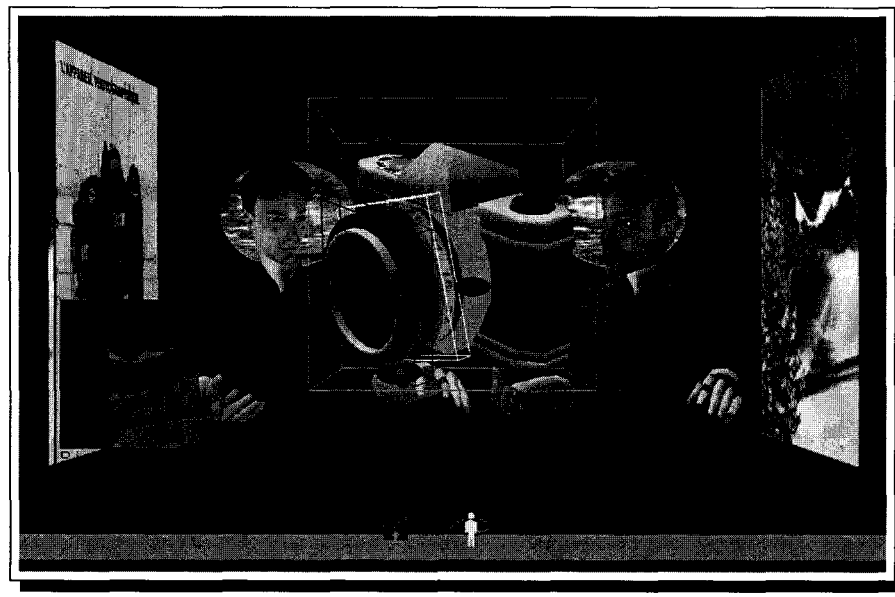
[Sau98] a proposé un modèle d'organisation spatiale basé sur une métaphore de table de réunion. Les documents et les différentes personnes présents lors de la réunion sont placés dans un monde virtuel 3D représentant une pièce unique. Les documents et les acteurs sont placés dans un même espace 3D renforçant ainsi la cohérence de la scène globale. A tout instant, chaque utilisateur a une vue globale sur l'activité et sur les documents de la réunion. Très facilement, il peut également "naviguer" d'un document à un autre.

Un découpage spatial de l'espace de travail est effectué (cf. figure 4.1) : une première partie, appelée le "bandeau", permet d'y placer les différents acteurs de la réunion ainsi que les documents nécessaires à l'activité coopérative. La seconde partie, la "table" placée au centre du bandeau, permet de placer de façon centrale le document intéressant l'utilisateur local (i.e. le document sur lequel il souhaite travailler). Chacune de ces parties de l'interface est appelée un *widget*.

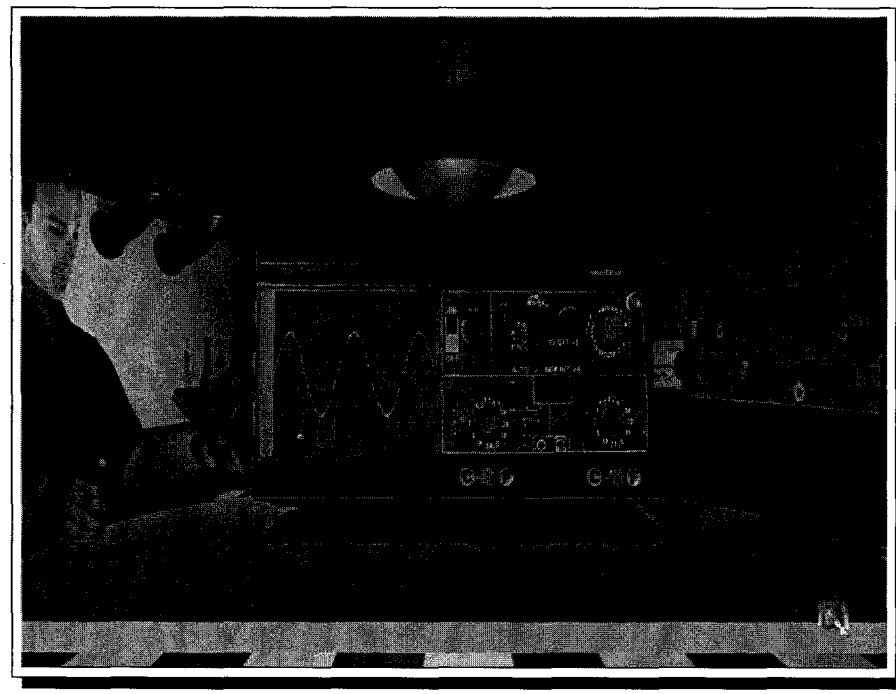
Le bandeau est composé de plusieurs "murs", chacun des murs contenant un document ou un avatar. Quand le nombre de murs composant le bandeau augmente, ils deviennent trop fins et par conséquent il devient difficile de les utiliser. C'est pourquoi l'interface offre la possibilité à l'utilisateur de pouvoir déformer le mur, situé face à lui, lui permettant ainsi d'utiliser beaucoup plus facilement le document qu'il contient (cf. figure 4.2).

L'utilisateur est capable de faire tourner le bandeau afin d'amener le mur de son choix face à lui : il est plus simple d'utiliser le document lorsqu'il est face à l'utilisateur. Cependant, les autres murs restent constamment visibles. Tout mur disparaissant sur la droite réapparaît automatiquement sur la gauche, et inversement (cf. figure 4.3).

Il est à noter que chaque utilisateur organise son espace de travail comme il le souhaite : il n'existe pas de cohérence globale au niveau de l'organisation de l'espace de travail. L'objet présent sur la table chez un utilisateur peut très bien être dans le bandeau d'un autre ; de même les objets du bandeau ne sont pas obligatoirement rangés dans le même ordre chez tout le monde.



(a) apprentissage de la manipulation d'un appareil photographique argentique



(b) apprentissage de la manipulation d'un oscilloscope

FIG. 4.1 – *L'organisation spatiale de l'interface*

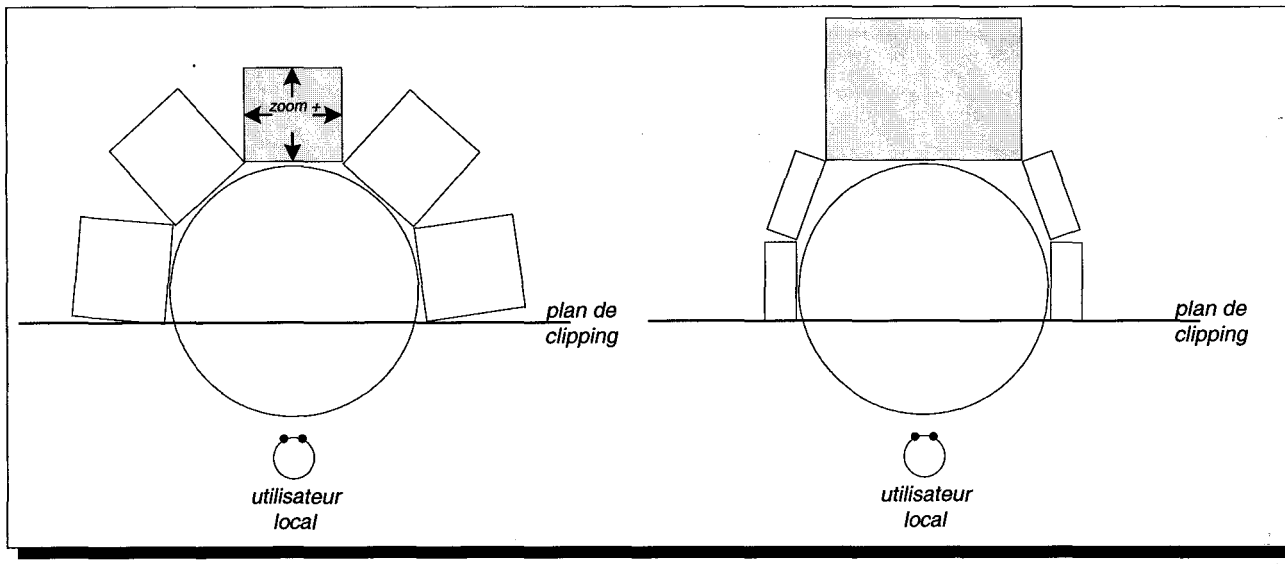


FIG. 4.2 – *Manipulation du bandeau : la déformation du mur central*

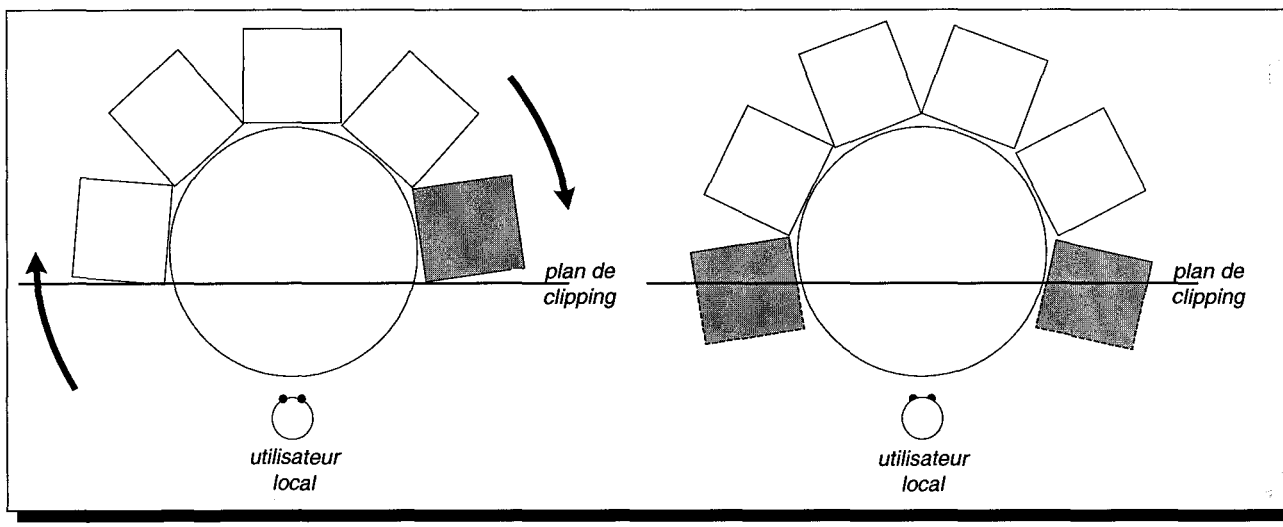


FIG. 4.3 – *Manipulation du bandeau : la rotation, tout objet disparaissant sur la droite réapparaît sur la gauche (et inversement)*

4.1.3 L'interaction homme-machine

[Dum99] a proposé un modèle d'interaction 3D pour l'interface définie ci-dessus. Les mécanismes de l'interaction de base reposent sur une interaction 3D utilisant deux périphériques (interaction *bi-manuelle*) :

- l'utilisation d'un premier périphérique isotonique, avec trois degrés de liberté, dans la main dominante pour déplacer un pointeur 3D dans l'ensemble de l'interface. Ce pointeur sert à désigner et sélectionner les documents,
- l'utilisation d'un deuxième périphérique isométrique dans la main non-dominante avec au moins trois degrés de liberté pour réaliser les manipulations sur les documents.

Le mécanisme d'interaction du terminal de travail Spin-3D est découpé en trois phases (cf. figure 4.4), la réunion de ces trois phases formant alors ce que nous appelons une action :

1. dans une première phase, l'utilisateur pointe l'objet qu'il souhaite manipuler à l'aide du périphérique isotonique ; ensuite après avoir cliqué sur ce dernier, l'objet est alors sélectionné
2. une fois l'objet sélectionné, l'utilisateur peut le manipuler à l'aide du périphérique isométrique,
3. un deuxième clic sur le périphérique isotonique entraîne la dé-sélection de l'objet, ie. l'arrêt de la manipulation et le retour à la sélection.

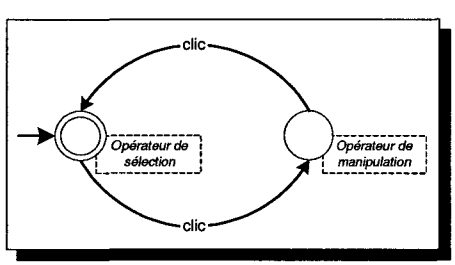


FIG. 4.4 – *Automate d'interaction de l'interface de travail*

L'utilisateur dispose de plusieurs indices visuels facilitant l'appréhension de la troisième dimension et l'interaction. Les ombres portées et des indices dynamiques comme par exemple les boîtes englobantes progressives permettent à l'utilisateur de placer plus facilement son pointeur par rapport aux documents (cf. figure 4.5). Des retours visuels permettent à l'utilisateur de comprendre l'activité des autres. Plusieurs évaluations ont été menées et ont permis de valider le modèle proposé [GTP99][DPD00].

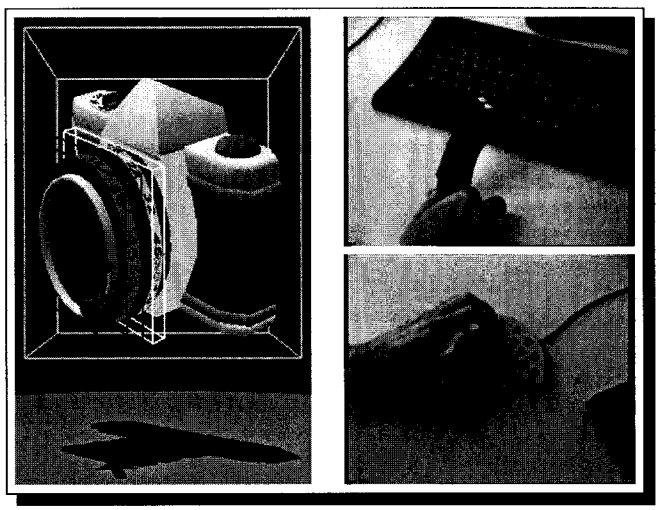


FIG. 4.5 – *Mécanisme d'interaction : les indices visuels (boîtes englobantes et ombres), des exemples de périphériques de désignation (en haut, l'OWL) et de manipulation (en bas, la SpaceMouse)*

A partir de ce modèle d'interaction de base, [Dum99] propose des primitives d'interaction de haut niveau comme par exemple les menus 3D ou l'explorateur de fichiers 3D (*conetree*) inspiré d'une proposition de Xerox (cf. figure 4.6).

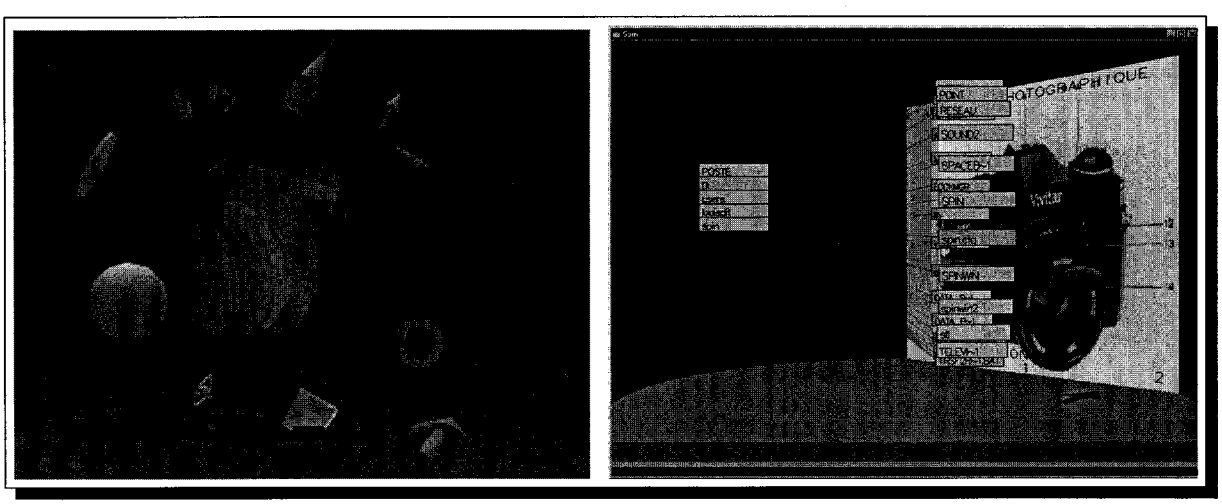


FIG. 4.6 – *Widgets évolués d'interaction : les menus 3D apparaissant autour des documents et l'explorateur de fichiers*

4.1.4 Le support de la téléprésence

[Dum99] envisage plusieurs solutions pour la représentation des acteurs de la réunion dans l'interface 3D : une photographie, une vidéo ou bien un avatar 3D. La solution retenue a été celle des avatars 3D : les différents acteurs sont représentés sous la forme d'avatars 3D placés dans le bandeau (cf. figure 4.1). Il est à noter que l'utilisateur local ne se voit pas dans l'interface, il voit uniquement les autres. L'interface permet de reporter à distance les différentes actions effectuées par un utilisateur. Ses interlocuteurs interpréteront ses actions par l'intermédiaire de son avatar : il permet de rendre le point d'intérêt de l'utilisateur grâce à la direction de son regard. Il dispose également d'un télépointeur, représentation distante de son pointeur local, lui permettant de désigner à distance des documents. D'un point de vue ergonomique, chaque utilisateur doit comprendre qui fait quoi dans l'interface. C'est pourquoi, lorsqu'un utilisateur sélectionne un objet, les utilisateurs distant sont prévenus par un changement de couleur de l'objet sélectionné : l'objet prend la couleur de l'utilisateur qui l'a sélectionné (chaque utilisateur possédant sa propre couleur). L'interface propose d'autres retours visuels d'interaction (*feedbacks*) permettant à l'utilisateur de comprendre l'activité. Notamment, lorsqu'un utilisateur cherche à manipuler un objet partagé déjà en cours de modification par un autre utilisateur, il est notifié qu'il lui est impossible, pour le moment, de sélectionner cet objet (cf. 4.2.2, page 124).

Dans les situations réelles de communication, le canal-verbal est utilisé de façon naturelle et inconsciente. Or, certaines informations, notamment celles liées à la gestuelle, sont très importantes. Grâce au fort potentiel de synthèse, ces différentes informations sont transmises via le vecteur de communication que sont les avatars 3D. [LM01] a identifié les données à entrer dans le système, le dispositif technique à mettre en œuvre et la façon d'animer les avatars 3D pour les rendre réellement communicants.

4.2 Aspects technologiques du premier prototype Spin

4.2.1 Description des données partagées

Dans la première version de Spin (que nous appellerons Spin-v1 par la suite), Dumas [Dum99] propose un mécanisme d'actions permettant de décrire l'interaction possible sur les différents objets de l'interface de travail. En 1994, quand le projet a commencé, VRML 1.0 était le standard normalisé pour l'échange de données 3D sur Internet. C'est également le format des différents documents 3D utilisés dans l'interface Spin-v1. De base, le langage VRML 1.0 ne propose pas de

mécanisme de description de l'interaction. C'est pourquoi, l'aspect interactif est géré séparément de la description géométrique, dans un fichier à part. En VRML 1.0, il est possible de nommer un noeud en particulier en utilisant le mécanisme de `DEF`. En utilisant ce mécanisme, le concepteur d'objet est capable de définir, dans un fichier autre que le fichier VRML 1.0 (un fichier possédant l'extension "`.def`"), le comportement des différentes parties de son objet (celles possédant un `DEF`), i.e. "l'action" à déclencher lors de la réception d'un message particulier (cf. figure 4.7). On définit une action comme une entité "intelligente" (comme par exemple un script) agissant sur un objet. Un mécanisme générique de gestion des actions a été mis en place : il permet au programmeur de développer ses propres actions. Dans Spin-v1, on ne peut manipuler un objet que si une boîte englobante apparaît autour de lui : Spin-v1 utilise les informations contenues dans le fichier d'actions pour déterminer si un objet est manipulable ou pas, i.e. un objet possédant un `DEF` et au moins une action sur ce `DEF` aura une boîte englobante.

La figure 4.8 illustre le mécanisme logiciel mis en oeuvre pour l'interaction, le mécanisme proposé est basé sur la gestion de messages (comme c'est le cas dans les interfaces 2D classiques Windows ou X-Window) :

- un message est retiré de la file d'attente de messages. Le message arrive sur l'objet sur lequel est placé le pointeur *(étape 1)*,
- Spin-v1 est en charge de retrouver l'action à déclencher en fonction du type du message *(étapes 2 et 3)*,
- l'action effectue des modifications sur le graphe de scène *(étape 4)*,
- si nécessaire elle propage le message aux interfaces distantes *(étape 5)* en plaçant le message dans une mémoire tampon. Cette mémoire tampon est vidée au fur et à mesure par Spin-v1.

C'est à partir de l'action spécifiée dans le fichier "`.def`" que le comportement partagé est également spécifié. En effet, l'implémentation de l'action peut transmettre à distance les messages qui lui ont été adressés afin de reproduire à distance le même comportement. Ce mécanisme est comparable à celui disponible dans *X11*.

4.2.2 Plate-forme de communication

Pour transporter les différents messages, une plate-forme de communication ad'hoc a été développée. Elle utilise une couche de transport construite au dessus d'un protocole multipoint non fiabilisé, par conséquent elle est donc sensible à la perte de paquets. Le manque de fiabilisation introduit des problèmes de cohérence entre les interfaces et par conséquent des problèmes de

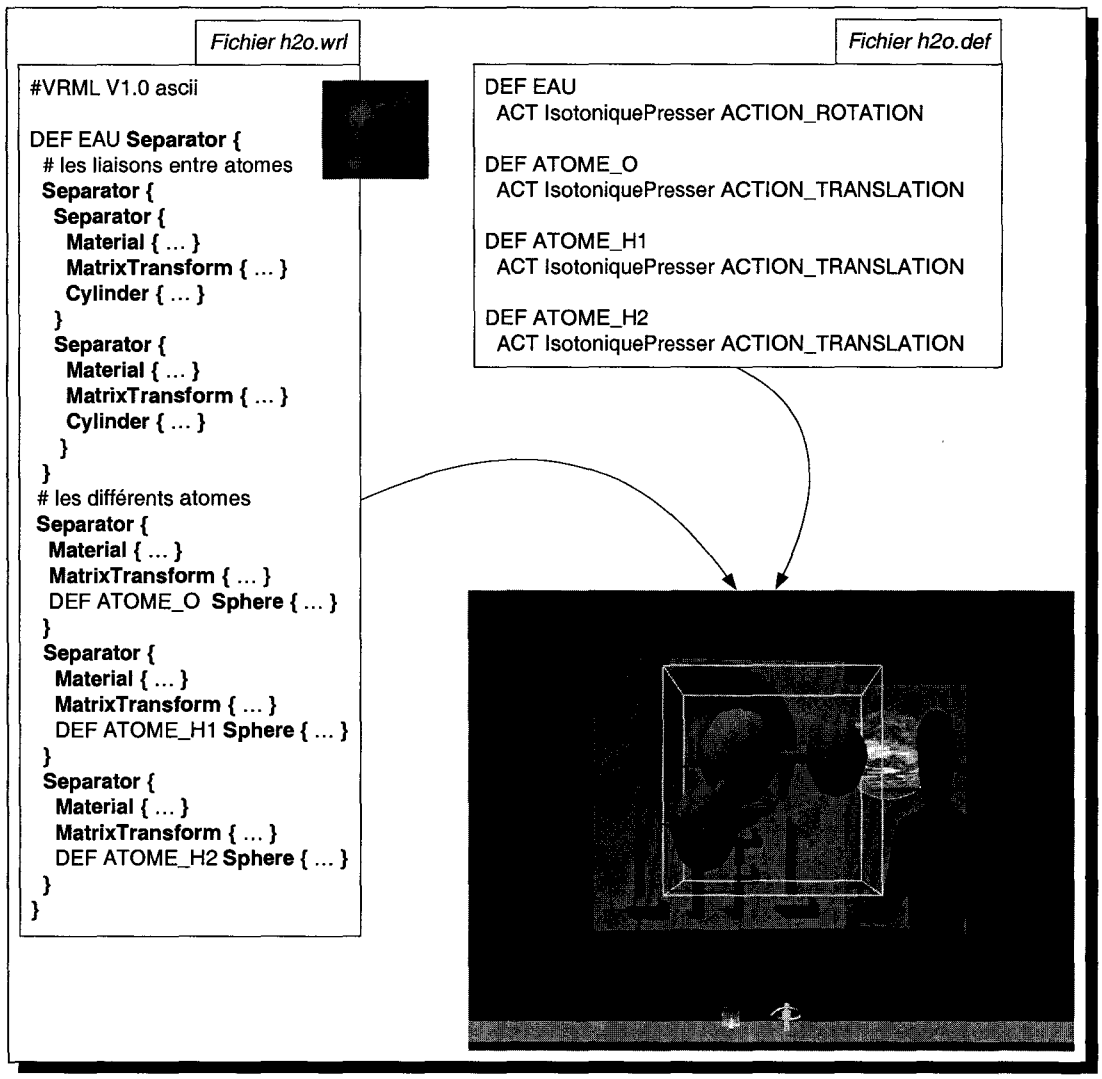


FIG. 4.7 – *Un document 3D au format VRML 1.0 (`h2o.wrl`) et son fichier d'actions associé (`h2o.def`) intégré dans l'interface Spin-v1*

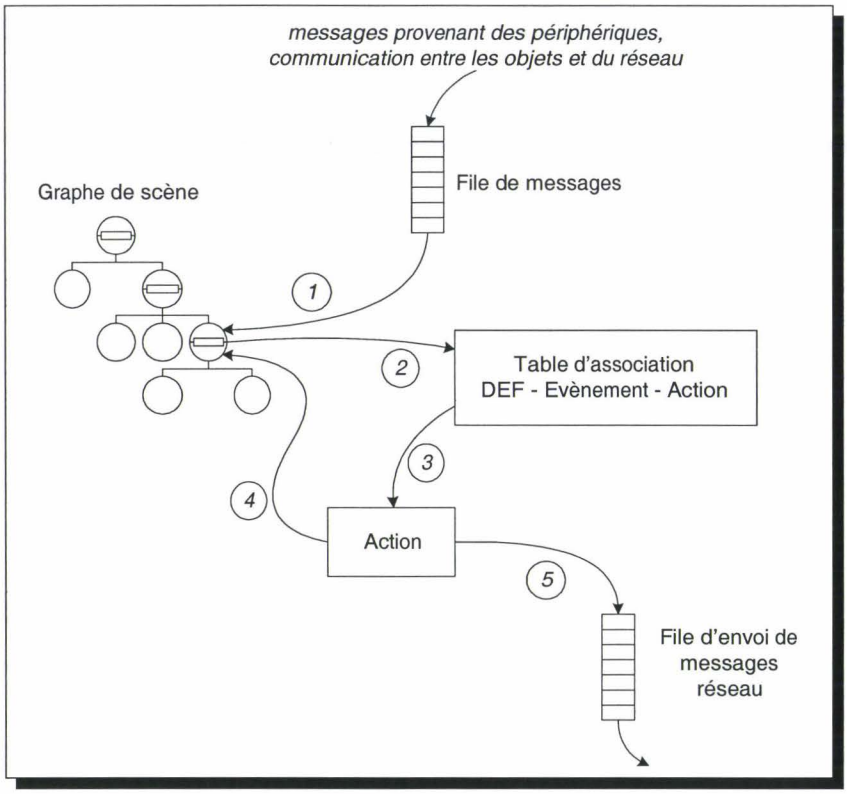


FIG. 4.8 – *Fonctionnement du mécanisme d'interaction du premier prototype de Spin*

compréhension de la part des utilisateurs apparaissent, ce qui est néfaste au bon déroulement de l'activité coopérative.

Pour gérer l'activité coopérative, la plate-forme offre différents services : gestion du groupe des participants, gestion des accès concurrents aux objets partagés de l'interface. La gestion des accès concurrents s'effectue via un mécanisme de verouillage : il empêche une action de se lancer lorsqu'une autre action est déjà en cours d'exécution sur le même `DEF`, empêchant ainsi à deux utilisateurs différents de manipuler le même objet partagé. Avec un tel mécanisme, on protège la sélection des objets partagés : un utilisateur ne peut sélectionner un objet partagé uniquement que si il est libre. L'objet est verrouillé, dès la sélection, tout le temps des manipulations de l'utilisateur, jusqu'à la désélection.

Lorsqu'un utilisateur veut réaliser une action sur un objet partagé, Spin-v1 diffuse une demande de verrou à tous les participants :

- si une autre instance de Spin-v1 possède déjà le même verrou, soit elle décide de le céder et lui transmet le verrou afin qu'il puisse manipuler l'objet, soit elle refuse la demande de verrou et l'utilisateur ne peut manipuler l'objet,

- si personne ne le possède, le verrou est créé par le demandeur et il peut manipuler l'objet.

Ce système impose de connaître tous les participants, mais ce n'est pas un problème dans Spin-v1 puisque le système doit gérer les différents participants connectés (entrée/sortie des participants pour la gestion de leur avatar).

De tels mécanismes utilisent également le mécanisme d'envoi de messages proposé par la plate-forme de communication : des messages particuliers circulent entre les différents postes réalisant ainsi un mécanisme similaire à celui de l'appel de procédure à distance (RPC, *Remote Procedure Call*). Le manque de fiabilité au niveau de la couche de transport des messages empêche ces différents services de fonctionner correctement. Il existe dans l'interface d'autres mécanismes qui nécessitent l'utilisation de ce "pseudo" mécanisme de RPC : un certain nombre d'informations sont interprétées localement avant d'être envoyées sur le réseau. En effet, du fait que les différents documents ne sont pas placés de la même façon chez tous les participants, seuls les terminaux distants sont capables de rendre correctement ces informations. Les terminaux distants traitent alors l'information pour délivrer l'information à l'utilisateur. C'est notamment le cas pour la gestion du télépointeur et la gestion du regard des avatars. Ce mécanisme de RPC permet ainsi d'envoyer des ordres de commande aux autres participants.

Bilan

La première version de Spin a permis de valider les différents concepts de l'interface, de l'interaction et de montrer que l'on peut mener une activité coopérative au travers d'un environnement virtuel 3D. Les différentes techniques mises en oeuvre pour la conception du premier prototype Spin-v1 manquent de généricité. L'utilisation de nouvelles techniques peuvent résoudre ce problème :

- en ce qui concerne la description des documents 3D, le format VRML 1.0 ne permet pas de décrire l'interaction avec l'objet. De nouveaux formats ont été proposés depuis 1994, comme par exemple le format VRML 97, permettant de décrire l'interaction et le partage de données (par la définition d'extensions),
- en ce qui concerne les communications entre les différents terminaux, la couche logicielle de communication a été développée spécifiquement pour Spin-v1. Les possibilités d'évolution peuvent être remises en cause. En effet, une action peut avoir besoin de messages particuliers et par conséquent il faut modifier la couche réseau pour tenir compte de l'ajout de ce nouveau type de message. De même, pour le pseudo mécanisme de RPC mis en oeuvre, il faut ajouter de nouveaux messages pour pouvoir faire transiter de nouvelles informations.

La programmation objet distribuée peut nous offrir de nouvelles solutions en terme de généricité.

L'étude des communications mises en oeuvre dans ce prototype nous a permis d'identifier deux cas d'utilisation de la plate-forme de communication. Premièrement, elle permet de maintenir cohérent l'état de variables dites "partagées" dans les différents terminaux participant à la session. Deuxièmement, elle permet aux différents terminaux de communiquer entre eux par un mécanisme similaire à celui des RPC permettant d'échanger des évènements ne pouvant être décrits à l'aide de variables partagées, comme par exemple l'entrée ou la sortie d'un participant, la demande d'un verrou, etc.

4.3 Recommandations pour une nouvelle version du terminal de travail coopératif Spin-3D

Le premier prototype de l'interface Spin a permis de valider un ensemble de concepts d'IHM pour l'utilisation d'un EVC 3D comme support au TCAO. Cependant, son utilisation à grande échelle est freinée par le fait des différentes techniques logicielles utilisées pour sa conception. L'équipe Graphix et France Télécom R&D ont décidé de poursuivre les recherches, au travers du projet Spin-3D, afin de proposer plus qu'une interface tridimensionnelle pour le TCAO synchrone mais une véritable plate-forme de développement d'applications coopératives. Nous souhaitons proposer un système avec lequel les concepteurs d'applications n'ont pas à se soucier des différents problèmes techniques sous-jacents à la distribution des utilisateurs et au maintien de la cohérence dans l'environnement virtuel 3D partagé. Ainsi, ils peuvent se concentrer sur le cœur même de l'application.

Dans cette section, en nous basant sur l'étude bibliographique précédente, nous présentons notre mécanisme de communication et de partage permettant de supporter la coopération entre des utilisateurs distants. Nous avons une couche de haut niveau (ie. au niveau du concepteur d'applications coopératives) permettant de masquer les différents problèmes liés à la distribution des utilisateurs et au maintien de la cohérence dans l'environnement virtuel 3D partagé. Ces différents problèmes sont gérés par une couche bas niveau (ie. au niveau du concepteur de la plate-forme de développement).

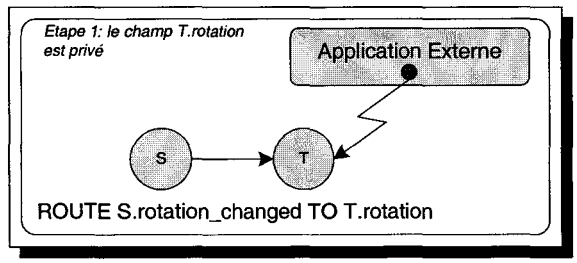
4.3.1 Description des objets partagés

Nous proposons de décrire les différents documents 3D en utilisant le langage VRML 97. Le langage VRML 97 permet d'embarquer l'interaction dans le même fichier que la description géométrique. Cependant, nous ne pouvons pas utiliser les navigateurs VRML 97 déjà existants du fait des spécificités de l'interface de travail que nous proposons. En effet, nous laissons à chaque utilisateur la possibilité d'organiser son espace de travail comme il le souhaite. Contrairement à la philosophie des mondes multi-utilisateurs réalisés en VRML 97, chaque utilisateur dispose de son propre graphe de scène construit par l'assemblage de plusieurs documents contenus dans la base de données. De plus, les mécanismes d'interactions comme les boîtes englobantes ou les ombres sont difficilement réalisables à l'aide de VRML 97. Pour ces raisons, nous préférons réaliser notre propre terminal VRML 97, qui peut être assimilé à un navigateur VRML 97 étendu permettant de gérer les spécificités de notre interface de travail (en terme d'organisation de l'espace de travail et de mécanismes d'interaction).

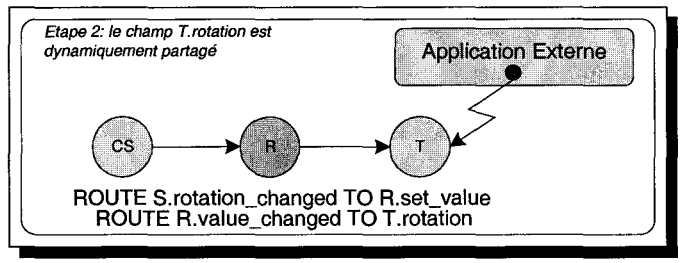
Le terminal d'exécution de notre interface de travail doit également être capable de gérer le partage de documents constituant le graphe de scène. Pour cela, nous souhaitons proposer un système descriptif: la réalisation d'applications coopératives s'effectue par la conception de contenus multi-utilisateurs, le concepteur n'ayant à spécifier que les données partagées pour chaque objet. Ainsi aucune compétence en programmation n'est requise, facilitant l'utilisabilité de notre plate-forme. Nous choisissons un mécanisme permettant de décrire le partage à l'intérieur même de chaque contenu VRML 97 comme par exemple (Core) Living Worlds, mais le mécanisme proposé doit rester simple d'accès (comme l'approche VSPLUS).

Cependant, nous devons proposer un nouveau paradigme basé sur un autre mécanisme que celui d'insertion. En effet, comme nous l'avons souligné précédemment, le paradigme d'insertion ne permet pas, facilement, de maintenir cohérent l'état des *sensors* : des mécanismes complexes à base de scripts sont à mettre en œuvre pour résoudre ce problème (cf. annexe C, page 240). Nous devons souligner un autre problème inhérent aux systèmes de TCAO qui nous pousse à ne pas utiliser le paradigme d'insertion. Dans une plate-forme de TCAO comme celle que nous proposons, nous n'avons pas pour but de vouloir intégrer tous les outils de travail (la tâche serait beaucoup trop importante et l'utilisateur est habitué à certains outils spécialisés). Au contraire, nous préférons laisser à l'utilisateur la possibilité d'utiliser un outil spécialisé (auquel nous aurons ajouté un module de communication lui permettant d'interagir avec le noyau de la plate-forme Spin) pour manipuler un objet (manipulation indirecte). Dans de telles situations, le paradigme d'insertion est inadapté (cf. figure 4.9). Sur la figure 4.9(a), l'utilisateur manipule dans son espace privé un objet par l'intermédiaire d'une application externe. Pour cela, l'application a obtenu une référence vers un champ d'un nœud contenu dans le graphe de scène. Sur la figure 4.9(b), pour

une raison quelconque, l'utilisateur souhaite publier auprès des autres participants le résultat de son travail. En utilisant le mécanisme d'insertion, des nœuds de partage sont ajoutés au graphe de scène. Cependant, l'application possède toujours une référence directe sur le champ : toutes les modifications effectuées par l'utilisateur depuis l'application externe ne sont pas transmises à distance entraînant ainsi des incohérences et des perturbations dans l'activité coopérative.



(a) avant le partage



(b) après le partage

FIG. 4.9 – *Application externe et partage dynamique en utilisant le paradigme d'insertion*

Nous devons définir un nouveau paradigme de partage respectant les contraintes suivantes :

- utilisant le langage VRML 97 pour décrire le partage,
- proposant un mécanisme, simple d'accès, permettant la création de contenus VRML 97 multi-utilisateurs à partir de contenus VRML 97 existants (mono-utilisateur) sans aucune modification de l'existant (hiérarchie géométrique ou chemins d'animation) ,
- assurant très facilement la synchronisation de l'état des `Sensors`,
- résistant au partage dynamique et à la connexion d'applications externes.

4.3.2 Mécanismes de communication

Dans l'architecture que nous proposons, nous distinguons deux mécanismes de communication :

- un mécanisme de communication que nous qualifions d'"interne" et qui est utilisé pour

assurer la coopération entre des utilisateurs physiquement distribués et permettant la communication entre les différents terminaux de l'EVC 3D,

- un mécanisme de communication que nous qualifions d'"externe" et qui utilisé pour la connexion d'applications sur le terminal de travail coopératif.

4.3.2.1 Mécanisme de communication interne

La scalabilité de notre système n'est pas primordiale : dans le cadre du projet Spin, du fait de l'organisation spatiale de l'interface de travail, nous nous plaçons dans un contexte de réunion regroupant un petit nombre de personnes (environ cinq personnes). De plus, les différents utilisateurs sont placés dans une unique pièce virtuelle et constamment en interaction directe.

La médiatisation d'une activité coopérative nous impose de respecter différentes contraintes afin de permettre aux utilisateurs de collaborer dans de bonnes conditions. Tout particulièrement, le système de TCAO que nous proposons doit être interactif afin que chaque utilisateur soit capable de comprendre l'activité, ie. ses propres actions et les actions des autres dans l'interface (*qui fait quoi ?*). Pour cela, nous devons nous tourner vers une base de données dupliquée. L'environnement virtuel proposé dans le cadre du projet Spin-3D étant de petite taille (réduit à une unique pièce), chaque participant dispose sur sa machine locale d'une copie complète des différents objets partagés présents dans l'interface de travail. Toujours pour respecter les contraintes en interactivité, nous choisissons pour notre plate-forme de communication une topologie décentralisée, ie. sans aucun serveur. Ainsi, chaque utilisateur manipule les données partagées stockées en local sur sa machine et prévient les postes distants des modifications qu'il a effectuées. Cela permet de minimiser le temps de réponse lors de l'interaction.

Pour le bon déroulement de l'activité coopérative, nous avons besoin d'assurer une cohérence forte entre les données partagées. Cependant, ce degré de cohérence peut très souvent entrer en conflit avec le besoin d'interactivité. Il faut donc être capable de relâcher la cohérence pour certaines actions de l'utilisateur. Ainsi nous distinguons plusieurs types de données transportées par notre plate-forme de communication :

- des données correspondant à des actions ponctuelles de la part des utilisateurs (comme un "clic" sur un bouton) ou de la part du système pour gérer l'environnement virtuel (entrée/sortie de participants, gestions des accès concurrents),
- des données correspondant à des flots multimédia (animation d'un objet, audio ou vidéo). On distingue deux types de flots de données, ceux qui sont limités dans le temps (manipulation d'un objet 3D présent dans l'interface de travail) et ceux qui sont infini (audio/video).

Pour gérer l'activité coopérative, la plate-forme de communication doit fournir un ensemble de services comme la gestion du groupe, la gestion des accès concurrents. Le gestionnaire de groupe est en charge des entrées et sorties des différents utilisateurs : il permet de prévenir l'interface afin d'afficher les différents avatars des personnes présentes à la réunion virtuelle. Pour la gestion des accès concurrents, il est préférable d'utiliser une approche pessimiste afin de respecter les contraintes liées à l'utilisateur : l'utilisateur ne peut manipuler une donnée partagée que si il en a reçu l'autorisation des autres postes en session. Comme il n'est pas concevable d'appliquer l'approche – *acquisition du verrou, changement de la valeur, libération du verrou* – à chaque modification d'une donnée partagée, nous intégrons le mécanisme de contrôle d'accès au mécanisme d'interaction proposé par notre interface de travail. L'interaction se fait en plusieurs étapes : dans une première étape, l'utilisateur doit pointer l'objet qu'il souhaite manipuler avec le périphérique isotonique, ensuite il le sélectionne en cliquant dessus, le manipule avec le périphérique isométrique, et finalement, le relâche en cliquant de nouveau sur le périphérique isotonique. Le verrou est indépendant de la modification de l'objet partagé : il permet uniquement d'éviter que plusieurs utilisateurs sélectionnent le même objet partagé. Le contrôle d'accès s'effectue au moment de la sélection. Le système autorise ou non l'utilisateur à acquérir l'objet (ie. à obtenir le verrou qui lui est associé) : si un autre utilisateur est déjà en train de manipuler l'objet alors le système refusera la demande sinon il l'acceptera. Une fois l'objet acquis, l'utilisateur peut le manipuler à son gré, en effet, il lui appartient jusqu'à ce qu'il le désélectionne en cliquant une nouvelle fois. Au niveau du mécanisme d'interaction, la désélection provoque la libération du verrou associé. L'acquisition du verrou ne s'effectue qu'une seule fois par manipulation permettant de réduire au minimum les communications avec les postes distants. Il est à noter que pour les prochaines manipulations de cette objet partagé, dans le cas où c'est le dernier propriétaire qui souhaite le manipuler (comme c'est le cas par exemple pour des opérations d'ajustement), nous pouvons bénéficier d'accélération au niveau du traitement de l'acquisition du verrou.

Pour l'implémentation de la plate-forme de communication, nous préférons nous tourner vers les mécanismes à appel de méthodes, plus particulièrement vers la technologie CORBA, beaucoup plus générique que les librairies à passage de messages. La figure 4.10 présente notre plate-forme de communication avec les deux canaux de communication qu'elle propose.

Pour l'implémentation du premier canal de communication, la nouvelle norme MIOP proposée par l'OMG nous permet de réaliser très naturellement et très simplement des communications de groupes : à chaque donnée partagée correspond un groupe d'objets au sein duquel il faut maintenir la cohérence. Pour assurer une cohérence forte, la couche de transport utilisée par MIOP doit être fiabilisé. L'abstraction de la couche de transport utilisé par MIOP permet de s'adapter au contexte technique : des solutions utilisant le point-à-point ou un pont de diffusion sont envisageables et implémentables dès à présent. L'utilisation du multipoint pose cependant des

problèmes dans le sens où un mécanisme de fiabilisation de l'IP multipoint, adapté au transport de requêtes GIOP/MIOP, doit être proposé.

Pour l'implémentation du deuxième canal de communication, nous proposons d'utiliser le service de flux multimédia proposé par l'OMG. Tel qu'il est défini dans la norme, ce service est défini dans le cadre d'une activité client/serveur. Nous devons proposer un fonctionnement adapté à notre contexte de travail qui est de travailler sans serveur. Les flots utilisent également l'IP multipoint comme couche de transport.

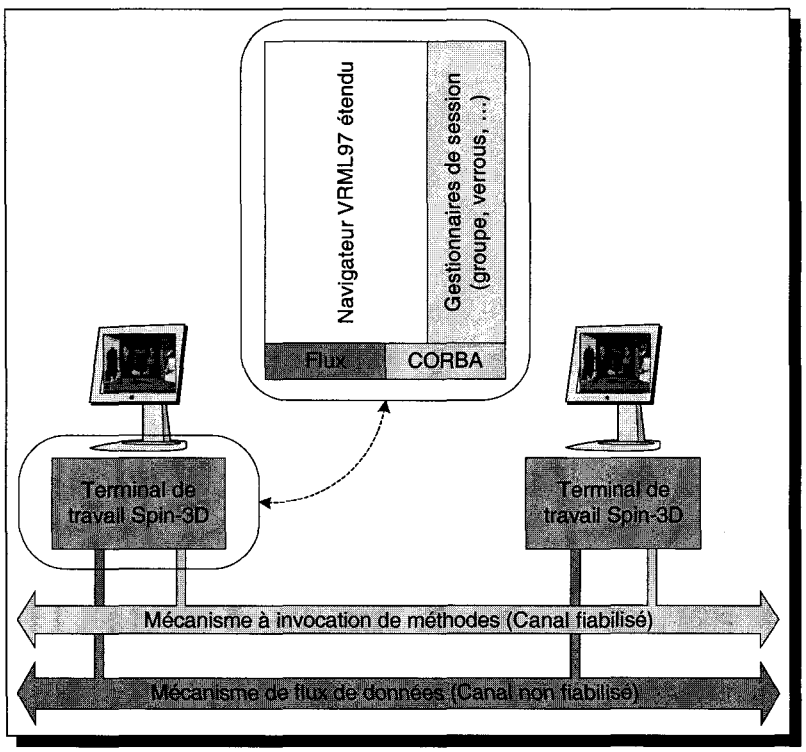


FIG. 4.10 – *Canaux de communications internes du terminal de travail Spin-3D*

Les différents services proposés par la plate-forme doivent utiliser des algorithmes complètement distribués (ie. sans serveur central), implémentés au dessus de notre plate-forme de communication. Nous avons identifié trois services :

- un service chargé de la gestion du groupe des participants à la session de travail virtuelle, et permettant d'afficher, dans l'interface, les différents avatars 3D représentant les utilisateurs distants,
- un service chargé de solutionner les problèmes d'accès concurrents aux documents partagés afin de permettre aux utilisateurs de coopérer dans les meilleures conditions et de comprendre l'activité,
- un service chargé de la persistance permettant à la fois de fournir l'état courant des ob-

jets partagés lorsqu'un utilisateur arrive en cours de session, et également permettant de sauvegarder, à la fin de la session, l'état courant des documents partagés.

4.3.2.2 Mécanisme de communication externe

Notre interface de travail 3D n'a pas pour but d'intégrer tous les outils classiques existants ; de même, nous n'avons pas pour but d'intégrer la plate-forme de travail coopératif dans toutes les outils classiques d'édition. Pour des raisons liées à l'usage, dans certains cas, nous préférons laisser les utilisateurs utiliser un outil classique pour éditer/modifier les documents présents dans l'interface 3D : d'une part, chaque utilisateur peut être habitué à un outil en particulier et à son fonctionnement (comme c'est le cas notamment avec les différents outils de CAO) ; d'autre part, les mécanismes d'interaction de ces outils ne sont pas forcément adaptés à un environnement 3D. Nous souhaitons donc mettre en place une interface de communication permettant à des composants externes d'interagir avec les documents contenus dans l'interface de travail Spin-3D (cf. figure 4.11).

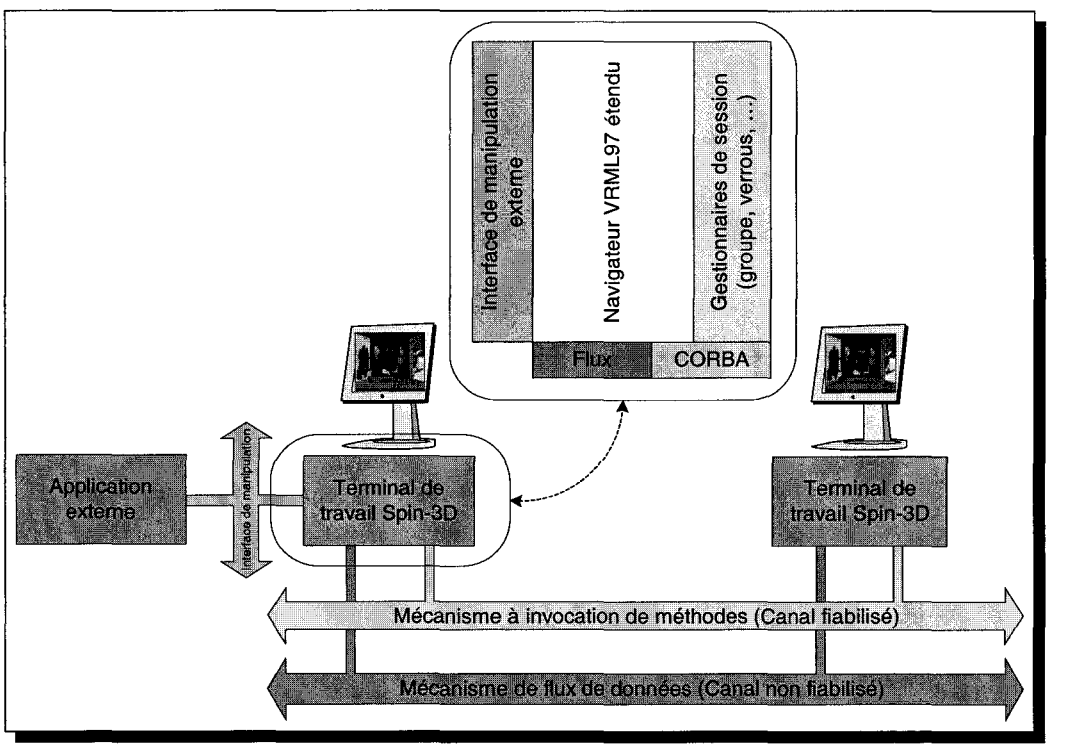


FIG. 4.11 – *Canal de communication externe du terminal de travail Spin-3D*

Pour l'implémentation d'un tel service, nous nous tournons vers l'EAI que propose la norme VRML 97. En effet, dans un navigateur VRML 97 classique, l'EAI permet à des programmes externes (des applets) de modifier la scène VRML 97 contenue dans le navigateur. Cependant,

bien que la définition de l'EAI soit générique, les navigateurs VRML 97 classiques permettent uniquement la communication entre un contenu VRML 97 et une *applet* Java contenue dans la même page HTML. Nous souhaitons proposer un mécanisme permettant d'étendre les implémentations classiques de l'EAI pour que n'importe quel programme écrit dans n'importe quel langage et s'exécutant n'importe où et sur n'importe quelle plate-forme (sur une machine différente, comme par exemple un PDA) soit capable d'interagir avec le noyau d'exécution de notre terminal de travail.

Deuxième partie

Propositions

Chapitre 5

Mécanisme de communication interne

Sommaire

Nous avons plusieurs terminaux de travail distribués géographiquement et connectés entre eux par un réseau informatique. Chaque terminal possède une base de données 3D, les données 3D partagées sont dupliquées sur chaque terminal de travail, ie. chacun possède sa propre instance de chaque donnée partagé. À chaque donnée partagée, nous associons un groupe "logique" constitué des différentes copies possédées par chaque terminal de travail. Nous devons maintenir la cohérence au sein de chaque groupe "logique", ie. dès que dans un groupe, un membre est modifié, les autres membres appartenant au même groupe "logique" doivent également être modifiés de la même façon.

Dans ce chapitre, nous allons présenter le mécanisme de communication utilisé par l'EVC Spin-3D qui permet de maintenir la cohérence dans les différents groupes représentant les données partagées présentes dans l'interface de travail [LDPDG+01].

5.1 Communication de groupe

Nous distinguons trois types de données que les terminaux de travail sont susceptibles de s'échanger pendant la session coopérative :

- des données ponctuelles correspondant à des actions ponctuelles de l'utilisateur dans l'interface, comme par exemple l'action d'appuyer sur un bouton,
- des données correspondant à des évolutions continues et illimitées dans le temps comme par exemple le flux sonore utilisé par les participants à la session pour communiquer oralement,
- des données correspondant à des évolutions continues et limitées dans le temps comme par exemple l'animation d'un objet présent dans l'interface.

Pour le premier type de données, nous avons besoin d'une *cohérence forte*. Ces données doivent être transmises de façon fiable assurant que tous les participants aient la même vue du monde virtuel 3D partagé. Pour le deuxième type de données, on peut tolérer une *cohérence relâchée* : dans un flot, des informations peuvent ne pas être toutes reçues, les informations suivantes rétabliront la consistance.

Enfin, pour le troisième type de données, nous proposons un compromis utilisant à la fois une cohérence faible et une cohérence forte. En effet, par exemple dans la plupart des cas de manipulation d'objet (translation, rotation), la dernière information est importante pour maintenir cohérent l'espace de travail virtuel : la dernière information contient l'état final de la donnée partagée après manipulation, par conséquent, elle doit être transmise de façon fiable. Toutes

les informations qui la précèdent peuvent être transportées sur un canal non fiabilisé : elles permettent uniquement de renseigner les participants sur l'activité mais elles n'ont aucune incidence sur le résultat final de l'action. Pour ce type de données partagées, les manipulations opérées sont transmises aux terminaux distants en valeur absolue.

Pour cela, nous offrons deux canaux de communications : un premier canal offrant une cohérence forte et un deuxième canal offrant une cohérence relâchée. Nous allons maintenant détailler le fonctionnement de chacun des canaux de communication.

5.1.1 Canal à cohérence forte

Comme nous l'avons mentionné précédemment, le protocole MIOP, tel qu'il est normalisé par l'OMG, ne permet pas d'assurer la cohérence au sein du groupe d'objets. En effet, dans la norme MIOP, le message GIOP correspondant au codage de la méthode à invoquer est fractionné en plusieurs paquets et ces paquets sont envoyés en utilisant le protocole réseau multipoint IP non fiable (UDP/IP) aux sites distants hébergeant un objet du groupe. Sur un site possédant un membre du groupe, il peut arriver qu'une requête GIOP ne puisse être reconstruite, et par conséquent exécutée, perturbant alors la cohérence du groupe. Or, dans la mesure où notre canal à cohérence forte repose sur le protocole de communication de groupe MIOP, nous devons proposer un mécanisme de fiabilisation de MIOP.

Pour certains traitements, il se peut que l'on ait besoin d'établir une connexion point à point fiable entre deux terminaux. Pour cela, nous proposons d'utiliser le protocole IIOP tel qu'il a été défini par l'OMG. Comme nous le verrons dans la suite de ce chapitre, le service de gestion de groupe utilise des techniques de communication point à point pour permettre à deux terminaux de s'échanger des informations.

5.1.1.1 Fiabilisation du protocole MIOP

Pour fiabiliser le protocole MIOP, nous devons assurer la réception de tous les messages GIOP, ie. la réception de tous les fragments composant chaque message GIOP. Ainsi, chaque message GIOP peut être reconstruit et exécuté sur chaque site hébergeant un membre du groupe.

Les protocoles à base uniquement d'acquittements négatifs (`NACK`) ne sont pas utilisables : de tels protocoles sont destinés à être utilisés dans le cadre de flux continus, ie. après un fragment suit toujours un autre fragment permettant ainsi de détecter la perte éventuelle de fragments

intermédiaires. Or dans notre cas de figure, chaque message GIOP étant de taille limitée, nous avons un nombre limité de fragments à envoyer. De plus, dans certains cas, un message GIOP peut être contenu entièrement dans un seul fragment, et par conséquent avec un protocole à acquittement négatif, il devient impossible de détecter la perte de ce fragment, et donc du message. Les protocoles à base d'acquittements positifs entraînent une surcharge au niveau de l'émetteur par la réception de multiples acquittements positifs.

Nous proposons d'utiliser un protocole hybride, c'est à dire de soumettre un fragment composant chaque message GIOP à une politique d'acquittement positif (cf. figure 5.1), les autres obéissant à une politique d'acquittement négatif. Un message GIOP est découpé en n fragments et chacun des fragments est numéroté. Les $n-1$ premiers fragments sont soumis à une politique d'acquittements négatifs : à la réception d'un fragment, le récepteur découvre les fragments précédents qui sont susceptibles de manquer en comparant le numéro du dernier fragment reçu avec celui qu'il vient juste de recevoir. Quant à lui, le dernier fragment est soumis à une politique d'acquittement positif : l'émetteur émet le fragment jusqu'à la réception de l'acquittement positif de tous les destinataires. À la réception du dernier fragment, chaque récepteur envoie éventuellement une demande de retransmission des différents fragments manquants pour la reconstruction complète du message GIOP originel. Une fois le message complètement reconstitué, chaque récepteur envoie son message d'acquittement positif à l'émetteur.

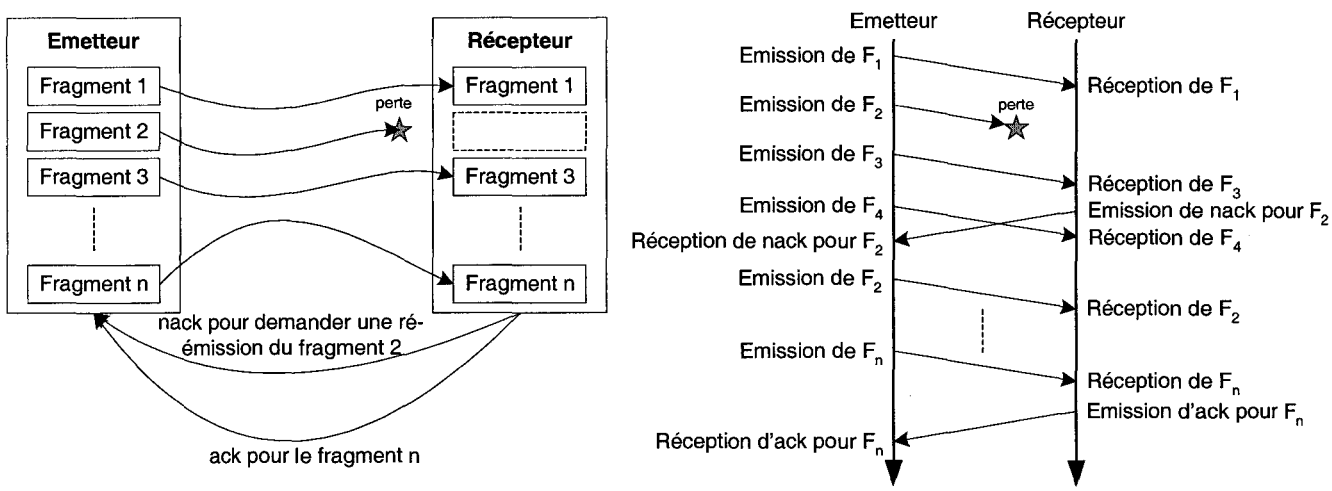


FIG. 5.1 – *Protocole multipoint fiable (hybridation NACK/ACK) utilisé pour la fiabilisation de MIOP (les n fragments représentent un message GIOP)*

Le schéma du fonctionnement de notre protocole MIOP fiabilisé (que nous appellerons RMIOP - *Reliable MIOP* - par la suite) est présenté sur la figure 5.2.

Avec un tel schéma de fiabilisation, l'émetteur de la requête a bien évidemment besoin de

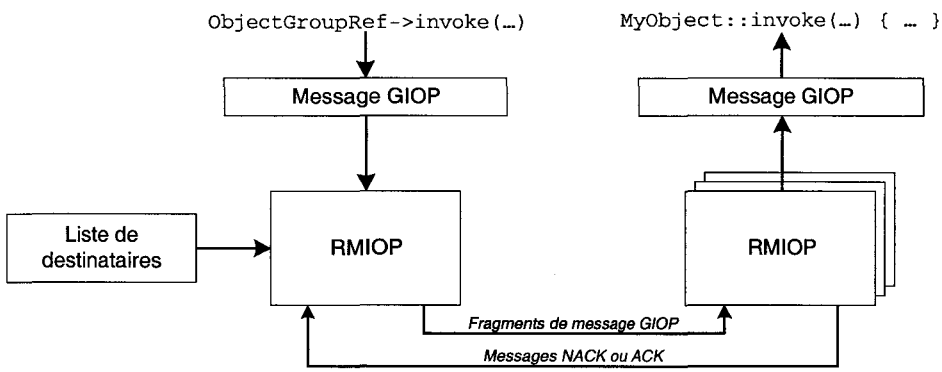


FIG. 5.2 – *Schéma de fonctionnement de notre couche RMIOP*

connaître la liste des destinataires pour gérer la réception des acquittements positifs du dernier fragment. Comme nous le verrons dans la suite de chapitre, le service de groupe a pour but d'assurer cette fonction. Nous profitons du fait que, dans le cadre de notre activité, la liste des participants doit être connue afin d'assurer la gestion des avatars, et donc nous pouvons également utiliser cette liste, à plus bas niveau, pour fiabiliser le transport.

5.1.1.2 Implémentation

Pour l'implémentation de RMIOP, nous avons utilisé le bus CORBA Orbacus développé par Iona. Nous utilisons la version 4.0.5 d'Orbacus. Il est à noter que dans les versions plus récentes d'Orbacus (version 4.1.0 ou supérieure) fournies à partir de décembre 2002, un plug-in de communication multipoint est fourni. Ce plug-in utilise le multipoint IP non fiable. Au moment où cette thèse a démarré, en octobre 2000, aucun plug-in multipoint n'était fourni. Nous avons donc dû implémenter notre propre couche de communication MIOP : dans un premier temps, elle utilisait le multipoint IP non fiabilisé ; dans un second temps, nous avons implémenté la fiabilisation du multipoint IP en utilisant la mécanisme décrit ci-dessus.

Avec le bus CORBA Orbacus, une interface de programmation [Ionb], appelée Open Communications Interface (OCI), est fournie permettant de développer de nouveaux protocoles de communication, comme par exemple MIOP, et se substituant au classique protocole IIOP fourni avec l'ORB. Le fonctionnement de cette interface de programmation est fourni dans [Ionc]. Dans l'annexe E (page 250), le lecteur trouvera des informations techniques sur le fonctionnement de l'OCI et sur le protocole RMIOP.

5.1.2 Canal à cohérence faible

Pour d'autres données mises à jour régulièrement comme le son, la vidéo ou une animation 3D, il n'est pas nécessaire d'avoir une cohérence forte, une cohérence faible suffit. Le protocole GIOP utilisé pour encoder les appels de méthodes introduit un surcoût important du fait qu'il contient la référence de l'objet, la signature de la méthode et les paramètres. Par exemple, le codage d'un appel de méthode prenant en paramètre quatre flottants (`setValue(float v[4])`) crée un message GIOP d'une taille de 112 octets. Sachant qu'un flottant est codé sur 4 octets, nous avons donc un surcoût dû au codage de $112 - 4 * 4 = 96$ octets (soit plus de 85% de la taille du message total). C'est pourquoi, nous réservons le canal à cohérence forte pour seulement certaines données qui ont besoin d'être mise à jour ponctuellement. De plus, GIOP n'offre pas de mécanisme de synchronisation, souvent utilisé pour le transport de données multimédia temps réel comme le son ou la vidéo.

Notre plate-forme offre un deuxième canal de communication permettant de relâcher la cohérence : nous proposons d'établir entre les membres d'un même groupe un flot multimédia par lequel transiteront les données. Le transport d'un message par ce flot de données est non fiabilisé, ie. le terminal émetteur ne possède aucune garantie de la bonne réception d'un message émis à destination des terminaux distants, permettant de limiter le surcoût en temps et en bande passante dû à la fiabilisation du transfert.

5.1.2.1 Contrôle des flots multimédia

Une copie de chaque donnée partagée est présente dans chacun des terminaux. L'ensemble des copies formant un groupe "logique" au sein duquel il faut maintenir la cohérence. Dans le cadre d'une cohérence relâchée, un flot de données est établi entre les différents membres du groupe "logique". Nous allons maintenant nous intéresser au mécanisme permettant de contrôler les flots multimédia.

Pour gérer les différents flots, nous utilisons le service de flux multimédia proposé par l'OMG (norme *A/V Streams* que nous avons abordé succinctement au cours de l'état de l'art (cf. 2.2.3.3.3, page 62) et dont le lecteur trouvera en annexe une présentation plus détaillée (cf. B.3, page 236)). Cependant, le service, tel qu'il a été défini, est utilisé dans le cadre d'applications basées sur le schéma classique client/serveur : un service de contrôle de flux multimédia, s'exécutant sur une unique machine, permet de mettre en relation des producteurs et des consommateurs. Pour pouvoir utiliser le service de contrôle de flux, nous devons adapter son fonctionnement à notre

architecture de gestion décentralisée.

À haut niveau, nous définissons un objet partagé de l'interface de travail comme étant un objet exposant un certain nombre de propriétés partagées (ie. modifiables par tous les utilisateurs). En reprenant la terminologie introduite par la norme, l'objet partagé peut être considéré comme un périphérique multimédia (`MMDevice`) et les différentes propriétés exposées par un objet partagé peuvent être considérées comme des producteurs/consommateurs de flots (`FlowDevices`). L'objet partagé (`MMDevice`) est donc associé à un flux (`StreamEndpoint`), et chacune de ses propriétés exposées (`FlowDevices`) est associée à un flot élémentaire (`FlowEndpoint`) contenu dans le flux. Dans le chapitre suivant, nous présentons également comment décrire le découpage logique en `MMDevice/FlowDevices` (`StreamEndpoint/FlowEndpoint`) d'un objet partagé présent dans l'interface de travail (cf. 6.2.1, page 172 et cf. 6.2.2, page 174). La figure 5.3 présente un exemple d'objet 3D partagé sur lequel nous effectuons un découpage logique en `MMDevice/FlowDevice` (`StreamEndpoint/FlowEndpoint`).

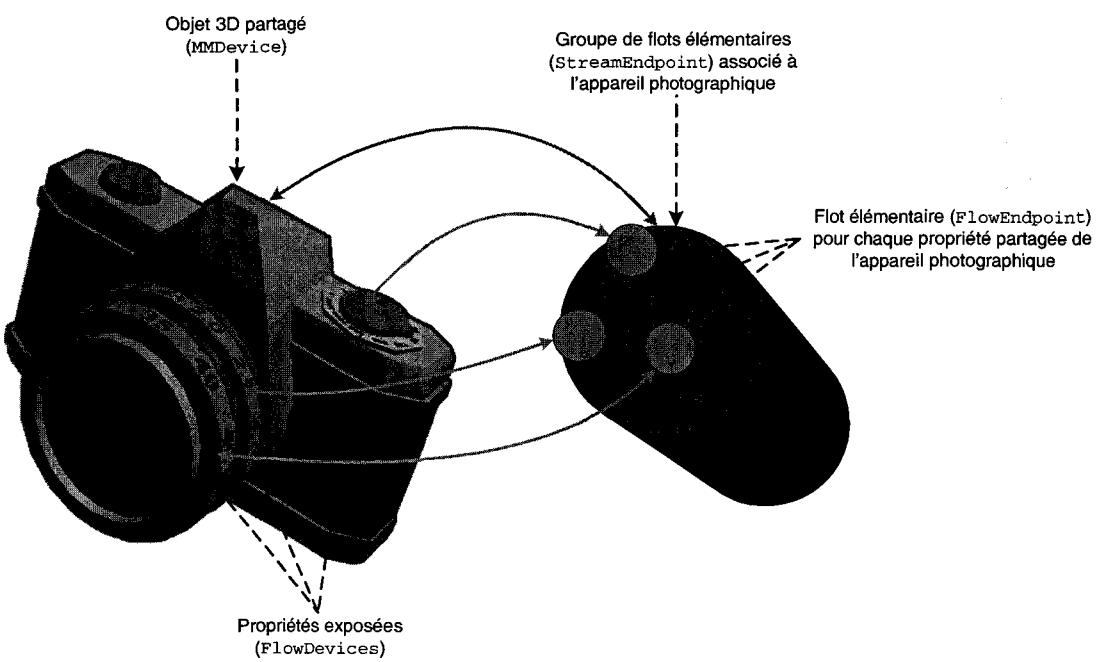


FIG. 5.3 – *Exemple de découpage logique d'un objet 3D partagé en `MMDevice/FlowDevices` et `StreamEndpoint/FlowEndpoints`*

La norme de l'OMG définit, lors de l'établissement d'un flux entre un producteur et un consommateur, une phase de négociation/configuration afin que les deux parties se mettent d'accord sur le format des données échangées via chacun des flots élémentaires. Dans notre architecture, nous ne pouvons pas mettre en place cette phase de négociation/configuration. D'une part, dans notre architecture, les producteurs de données transportées par un flot sont également les consommateurs. Lors d'une communication par flot entre les différents membres

d'un groupe représentant une donnée partagée, à un instant donné, une copie est producteur, les autres copies étant des consommateurs : la diffusion s'effectue de 1 vers n. D'autre part, dans un contexte de diffusion, il n'est pas envisageable d'exécuter cette phase de dialogue, la norme de l'OMG ayant défini cette phase de configuration pour deux parties misent en relation directement (ie. flot point à point). C'est pourquoi, nous ne gérons pas cette phase de négociation entre les producteurs et les consommateurs.

Avec ce canal de communication à cohérence faible, nous sommes dans un contexte de diffusion, par conséquent, chaque terminal de travail n'a pas connaissance des terminaux qui écoutent. Contrairement à la norme les flots établis sont bidirectionnels : à un instant donné, sur un terminal, un flot peut être utilisé en sortie, pour transmettre aux terminaux distants des modifications ; à un autre instant, sur ce même terminal, ce même flot peut être utilisé en entrée, pour recevoir des modifications provenant d'un terminal distant. Les opérations de création et destruction des structures associant un objet partagé à un flux (`StreamEndpoint`) ainsi que chacune des propriétés exposées à un flot élémentaire (`FlowEndpoint`) sont laissées à la charge de chaque terminal : au moment du chargement d'un objet partagé, chaque terminal crée les différentes structures nécessaires à une communication par flots et qui sont associées à l'objet chargé. Tout particulièrement, un objet `StreamCtrl` en charge de gérer le flux (`StreamEndpoint`) est créé.

Deux profils sont disponibles pour le contrôle des flux. Un profil (*Full Profile*) permet directement le contrôle des flux et des flots élémentaires. Un autre profil (*Light Profile*) autorise uniquement le contrôle des flux. Nous proposons de contrôler les flots uniquement au travers de l'interface du `StreamEndpoint`, ie. en utilisant le *Light Profile*, ce qui, d'un point de vue sémantique, revient à effectuer le contrôle sur l'objet lui-même. Chaque flot élémentaire est donc identifié à l'aide d'une URL (`FlowSpec`), identique sur tous les terminaux, et permettant de le contrôler via l'interface du `StreamEndpoint` auquel il appartient. Chaque terminal associe à chaque objet partagé un flux (`StreamEndpoint`) et un contrôleur (`StreamCtrl`) en charge de gérer le flux. Nous proposons de connecter chaque objet `StreamCtrl` au canal RMIOP présenté précédemment. Cet ensemble d'objets `StreamCtrl` identiques forme un groupe : lorsqu'une opération de contrôle est réalisé sur un objet `StreamCtrl` membre du groupe, il prévient les autres membres du même groupe afin qu'ils exécutent eux aussi la même opération. Pour cela, il utilise des appels de méthodes à distance sur la référence de groupe.

Pour un flot limité dans le temps, lorsque des données doivent être émises sur un flot, ce dernier doit tout d'abord être activé. Pour cela, une requête est réalisée auprès de l'objet `StreamCtrl` en charge de la gestion du *StreamEndpoint* contenant le flot. Le `StreamCtrl` prévient alors ses homologues distants afin qu'ils activent également le flot. Une fois les données émises, le flot doit être désactivé en effectuant une requête auprès du `StreamCtrl`, ce dernier étant en charge de

prévenir les autres membres de son groupe de la désactivation du flot.

Pour un flot illimité dans le temps, ce dernier doit être immédiatement démarré. C'est le cas par exemple du canal sonore utilisé par les participants pour communiquer entre eux pendant la session de travail. Un flux, représentant l'utilisateur local et contenant différents flots élémentaires, est créé au démarrage du terminal de travail. On trouve, par exemple un flot élémentaire utilisé à la fois pour émettre les données sonores de l'utilisateur local à destination des autres participants, et à la fois pour recevoir les données sonores des autres participant, ou bien encore un flot associé au pointeur local permettant de transmettre, à distance, les mouvements du pointeur local. Une fois créé, chaque flot illimité est immédiatement démarré et plus aucune opération de contrôle n'est réalisée au cours de la session. Les terminaux distants ne sont pas prévenus de l'activation de ces flots du fait que chaque terminal exécute le même schéma à l'initialisation.

5.1.2.2 Implémentation

La figure 5.4 présente la description IDL-OMG des interfaces `StreamCtrl`, `StreamEndpoint` et `MMDevice`. Ces interfaces ont été adaptées à l'architecture décentralisée.

La figure 5.5 présente le diagramme de séquence exécuté lors de la connexion d'un objet `MMDevice` à un objet `StreamCtrl`. Cela permet de créer dans la mémoire de chaque terminal les différentes structures nécessaires pour une communication par flux, ie. le `StreamEndpoint` et les `FlowEndpoints` associés.

Une fois que le `MMDevice` est relié à un `StreamCtrl`, il est possible d'activer ou désactiver les `FlowEndpoints` qui sont limités dans le temps en utilisant les méthodes `start` ou `stop` de l'objet `StreamCtrl` en charge de gérer le `StreamEndpoint` les contenant. Lorsqu'un contrôleur effectue une opération (`start`, `stop`) sur un flot, il prévient ses contrôleurs homologues distants qu'il a effectué une opération en utilisant un appel de méthode (`request_start` et `request_stop`). La figure 5.6 montre le diagramme de séquence exécuté lors de l'activation d'un flot, au niveau du terminal local (figure 5.6(a)) et au niveau des terminaux distants (figure 5.6(b)).

Lors de l'exécution de la méthode `stop` sur un flot, les données déjà reçues et en attente d'être consommées par le périphérique multimédia sont ignorées. De même, les données tardives arrivant après l'arrêt du flot sont également ignorées. Ainsi, pour les données au troisième type de partage (cf. 5.1, page 138), après l'arrêt du flot, l'émetteur envoie la dernière valeur sur canal à cohérence forte.

```
interface StreamCtrl {
        // Association du StreamCtrl avec un MMDevice
        void bind(in MMDevice device);
        void unbind();

        // Opérations de contrôle sur le StreamEndpoint associé
        void start(in FlowSpec spec);
        void stop(in FlowSpec spec);

        // Opérations utilisées pour communiquer avec les StreamCtrl distants
  oneway void request_start(in FlowSpec spec);
  oneway void request_stop(in FlowSpec spec);
};

interface StreamEndpoint {
  // Opérations de contrôle sur les FlowEndpoints (en utilisant le FlowSpec)
  void   start(in FlowSpec spec);
  void   stop(in FlowSpec spec);
  void   destroy(in FlowSpec spec);

  // Gestion des FlowEndpoints associés
  FlowSpec     add_fep(in name fep);
  void         remove_fep(in FlowSpec spec);
};

interface MMDevice {
  // Opérations appelées par le controleur lors de la creation
  // ou la destruction du flux
  StreamEndpoint create_stream_endpoint();
  void           destroy(in StreamEndpoint sep);

  // Gestion des FlowDevices associés
  string     add_fdev(in FlowDevice fdev);
  FlowDevice get_fdev(in string name);
  void       remove_fdev(in string name);
};
```

FIG. 5.4 – *Descriptions IDL-OMG des interfaces* `StreamCtrl`, `StreamEndpoint` *et* `MMDevice`

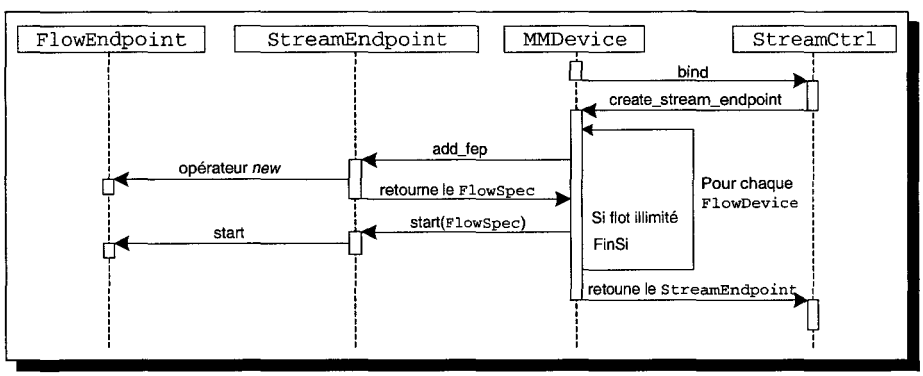
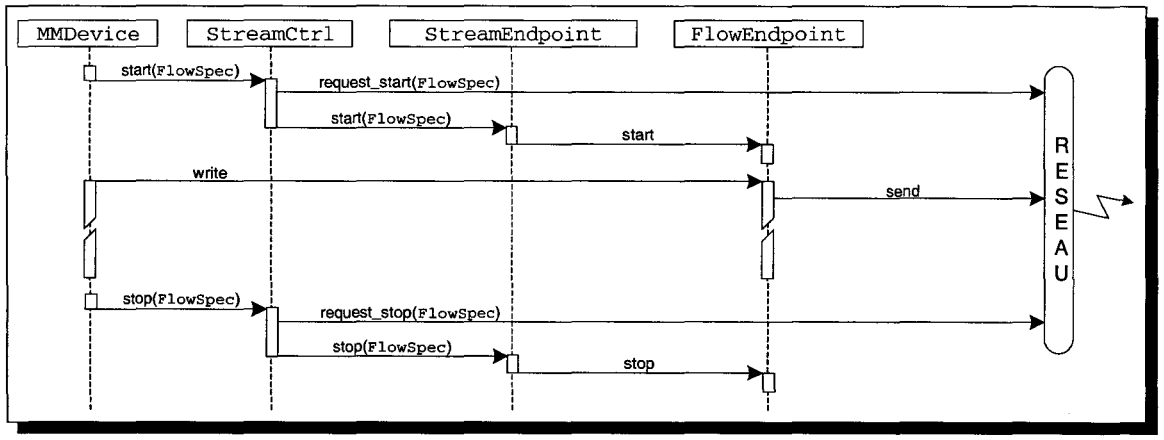
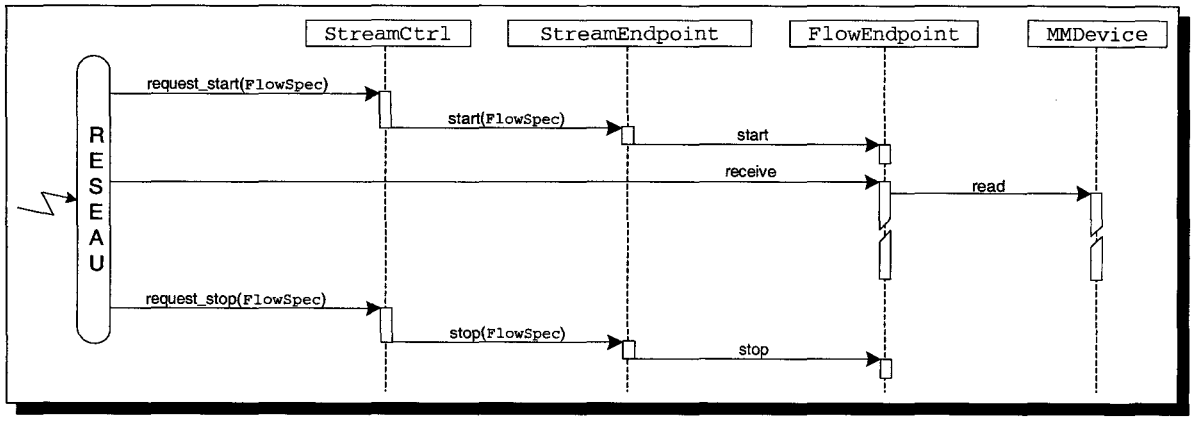


FIG. 5.5 – *Diagramme de séquence pour la connexion d'un objet* `MMDevice` *à un objet* `StreamCtrl`



(a) en local



(b) à distance

FIG. 5.6 – *Diagramme de séquence pour l'activation d'un flot*

5.1.3 Expérimentations

Afin de démontrer la faisabilité de notre architecture de communication, nous avons réalisé plusieurs tests visant à comparer notre protocole RMIOP avec le protocole IIOP. Pour cela, nous avons réalisé une application permettant le transfert d'un fichier entre un client et des serveurs. Le transfert de fichier est d'abord réalisé avec notre protocole RMIOP (qui utilise notre protocole multipoint fiable) et avec le protocole IIOP (qui utilise TCP/IP) fourni avec le bus CORBA Orbacus.

Le fichier est lu par l'émetteur par paquets de taille fixe (T_{Buffer}). Les données lues sont placées dans une mémoire tampon et sont envoyées au serveur en effectuant un appel de méthode sur l'objet d'implémentation localisé au niveau du serveur. À chaque appel de méthode, un message GIOP embarquant le tampon de données est généré et dont la taille est de $T_{GIOP} = T_{PayloadGIOP} + T_{Buffer}$ octets. Pour notre expérience, la taille de l'entête GIOP ($T_{PayloadGIOP}$) est de 96 octets : cet entête contient les informations permettant de localiser l'objet d'implémentation, la méthode à invoquer, etc.

Au niveau de IIOP, le message GIOP est envoyé au serveur en utilisant TCP/IP. Pour transmettre le message, TCP/IP le découpe en plusieurs fragments de taille de 1460 octets. Cela nous amène à une taille de 1500 octets pour le paquet IP, les entêtes IP et TCP étant de 20 octets chacun, sans les options, soit la taille maximale des données encapsulées dans un paquet Ethernet. Pour chaque fragment émis, l'émetteur attend un message `ACK` de la part du récepteur. L'attente d'un message `ACK` est pénalisante, c'est pourquoi TCP/IP utilise le mécanisme de "fenêtre glissante" définissant le nombre d'octets que l'émetteur peut émettre sans recevoir de message `ACK`. Au fur et à mesure de la réception des `ACK`, cette fenêtre est déplacée (glissée) le long du message à émettre.

Au niveau de RMIOP, le message GIOP est envoyé au serveur en utilisant notre protocole multipoint fiable fonctionnant au dessus de UDP/IP. Notre protocole découpe le message GIOP en n fragments dont nous avons fixé la taille à 8100 octets ($T_{Fragmentation}$), soit une taille proche de la "fenêtre glissante" de TCP/IP. La figure 5.7 présente en fonction de différentes taille de buffer (T_{Buffer}), la taille du message GIOP généré et le nombre de fragments par message GIOP.

Ceci nous amène à distinguer deux situations :

- $T_{GIOP} = (T_{PayloadGIOP} + T_{Buffer}) < T_{Fragmentation}$: dans ce cas, le message GIOP fragmenté occupe un seul fragment. Notre protocole multipoint fiable se comporte comme un protocole multipoint fiabilisé à l'aide de messages `ACK`,

T_{Buffer} (Ko)	T_{GIOP} (octets)	Nb de fragments par message GIOP
7	7168	1
16	16480	3
32	32864	5
64	65632	9
128	131168	17

FIG. 5.7 – *Taille et fragmention d'un message GIOP en fonction de plusieurs tailles de buffer*

- $T_{GIOP} = (T_{PayloadGIOP} + T_{Buffer}) >= T_{Fragmentation}$: dans ce cas, le message GIOP fragmenté occupe donc plusieurs fragments, l'émetteur doit alors répondre à des messages `NACK` et `ACK`.

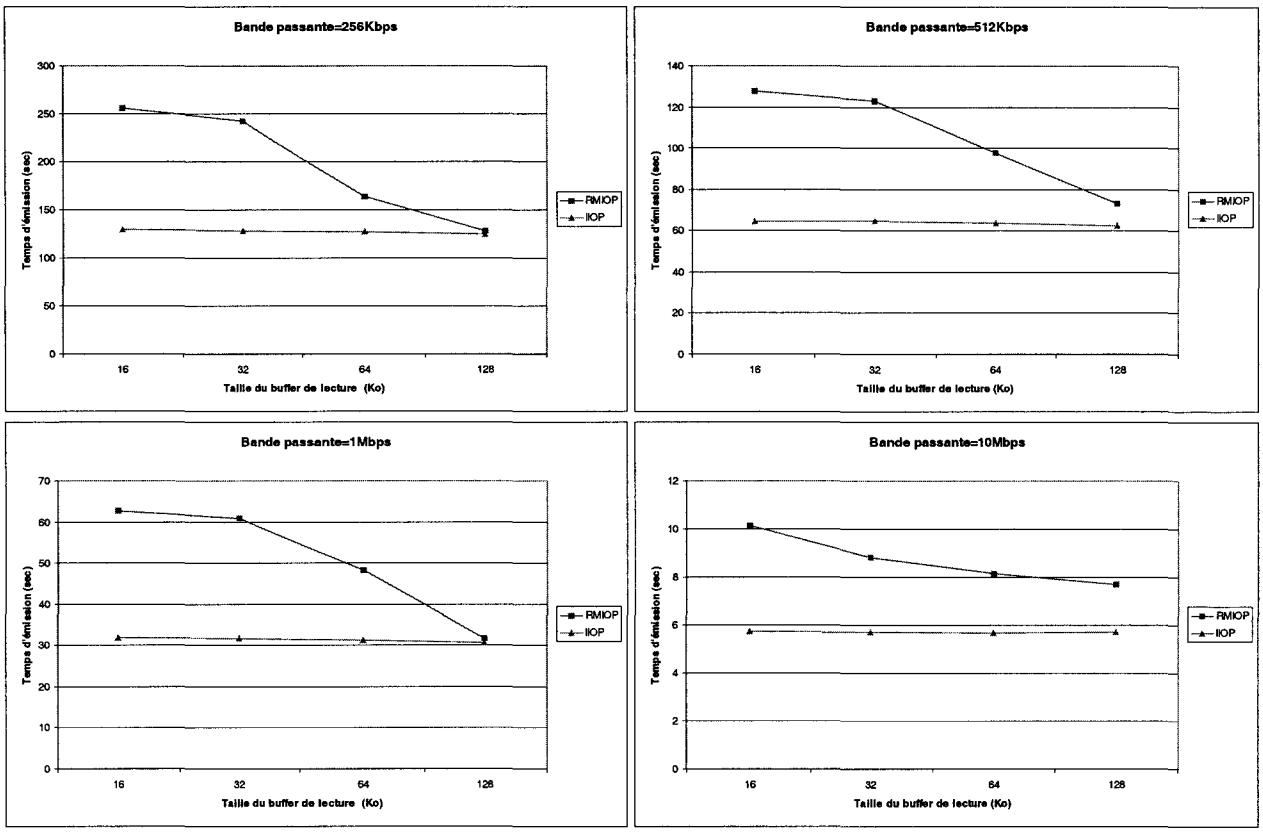
Le fichier transmis est d'une taille de 4Mo, ce qui correspond à plusieurs appels de méthodes fonction de la taille du buffer de lecture (T_{Buffer}). La figure (cf. figure 5.8) présente en fonction différentes tailles de buffer de lecture, le nombre de requête GIOP générées ainsi que le nombre de fragments total et le pourcentage de fragments soumis à une politique d'acquittement positif.

T_{Buffer} (Ko)	Nb de requêtes GIOP générées	Nb total fragments à envoyer	% de fragments soumis à `ACK`
7	586	586	100%
16	256	768	33.3%
32	128	640	20%
64	64	576	11.1%
128	32	544	5.9%

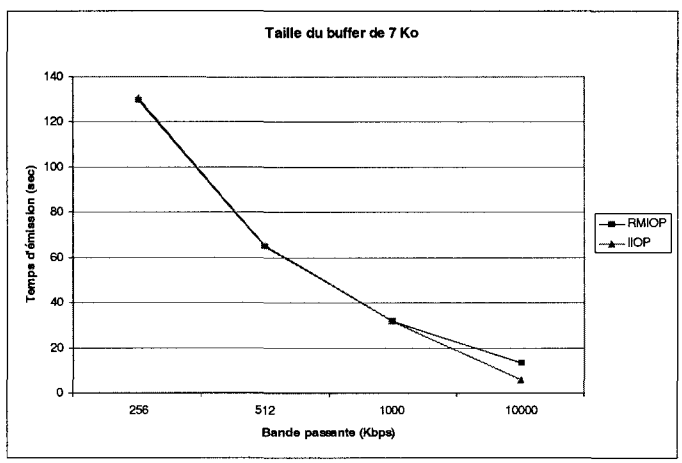
FIG. 5.8 – *Nombre de requêtes GIOP générées en fonction de la taille du tampon de lecture du fichier (d'une taille de 4Mo)*

5.1.3.1 Communication avec un seul serveur

Dans cette première expérience, nous effectuons le transfert de fichier entre le client et un seul serveur connectés entre eux par un routeur sur lequel nous sommes capables de modifier la bande passante. Le fait de limiter la bande passante permet de générer artificiellement de la perte de paquets.



(a) $T_{GIOP} = (T_{PayloadGIOP} + T_{Buffer}) >= T_{Fragmentation}$



(b) $T_{GIOP} = (T_{PayloadGIOP} + T_{Buffer}) < T_{Fragmentation}$

FIG. 5.9 – *Transfert d'un fichier d'un client à un serveur en utilisant IIOP et RMIOP sur un réseau sur lequel il possible de régler la bande passante*

Lorsqu'un message GIOP occupe n fragments ($n > 1$) fragments (cf. figure 5.9(a)), nous remarquons que plus la taille du buffer de lecture est grande, plus les performances de RMIOP se rapproche de celles obtenues avec IIOP. Nous expliquons cela par le fait que plus la taille du buffer est grande, moins nous avons de messages `ACK` à attendre (cf. figure 5.8).

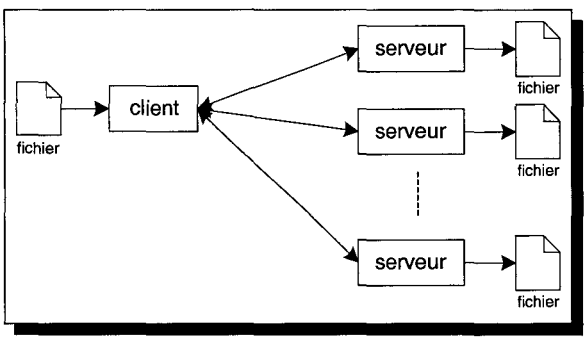
Lorsqu'un message GIOP occupe un seul fragment (cf. figure 5.9(b)), avec une faible bande passante, notre protocole multipoint fiable offre les mêmes performances qu'avec TCP/IP. Quand la bande passante augmente, le temps d'émission est 2 fois plus important avec RMIOP qu'avec IIOP. Cela s'explique par les optimisations de TCP/IP : comme par exemple la technique du "départ lent" qui permet d'asservir le glissement de la fenêtre d'émission au rythme de réception des `ACK`, évitant ainsi d'émettre les paquets de la fenêtre glissante d'un coup, aussi rapidement que l'autorise le système ou le débit théorique du réseau. Cependant, le multipoint montre un intérêt lorsque l'émetteur souhaite transmettre un message à n serveurs en permettant de ne l'envoyer qu'une seule fois.

5.1.3.2 Communication à n serveurs

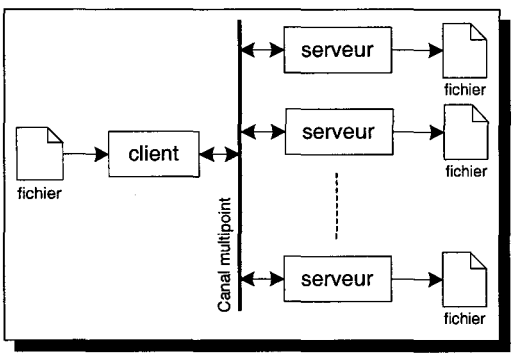
Dans cette deuxième expérience, nous avons multiplié le nombre de serveurs auxquels le fichier doit être envoyé. Pour cela, nous avons utilisé 7 machines identiques : une machine en charge d'émettre un fichier, 6 autres machines en charge de recevoir le fichier émis. Les machines sont reliées entre elles par un réseau Ethernet 100Mbps. La taille du fichier à transférer est de 4Mo.

5.1.3.2.1 Sur un réseau LAN

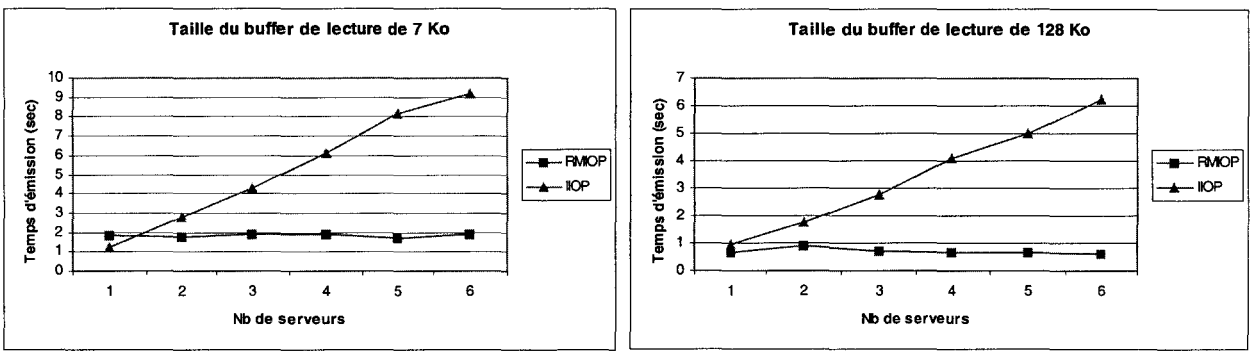
La figure 5.10 présente les temps d'émission en fonction du nombre de serveurs. Nous remarquons que pour TCP/IP, la courbe croit linéairement : le client transmet individuellement à chaque serveur le fichier, le fichier est donc envoyé n fois (n étant le nombre de serveurs). Avec RMIOP, nous bénéficions de l'utilisation du multipoint : cela nous évite d'envoyer individuellement le fichier à chacun des serveurs. Dans le cas d'un buffer de 7Ko (ie. un fragment suffit pour transporter le message GIOP complet), même si le temps d'émission est deux fois plus important avec RMIOP qu'avec IIOP, dès que nous avons plus de deux serveurs, les résultats de RMIOP



(a) Architecture de l'expérience en utilisant IIOP



(b) Architecture de l'expérience en utilisant RMIOP



(c) Résultats obtenus

FIG. 5.10 – *Transfert d'un fichier d'un client à n serveurs en utilisant IIOP et RMIOP sur un réseau* $100Mbps$

sont meilleurs que ceux obtenus avec IIOP.

5.1.3.2.2 Sur un réseau WAN simulé

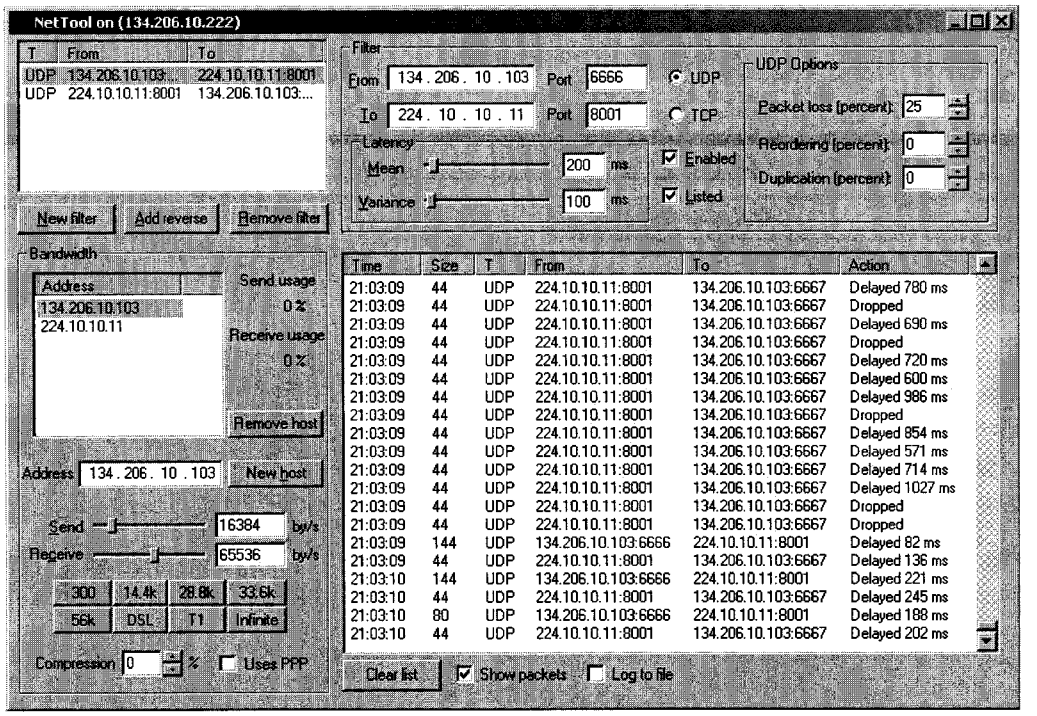
Nous avons réalisé cette même expérience en utilisant un simulateur de réseau proposé par Kirmse [TD01] et dont nous avons adapter le fonctionnement au multipoint. Le simulateur s'exécute sur une machine du réseau qui est utilisée comme passerelle entre le client et les serveurs. Le simulateur de réseau permet d'émuler un réseau WAN en réglant la bande passante, la latence, la perte de paquets, etc (cf. figure 5.11). Il est impossible de réaliser cette expérience avec IIOP du fait qu'il ne supporte pas les passerelles. À l'aide de ce simulateur, nous avons introduit :

- de la latence ($200ms$ $+ou-$ $100ms$) simulant un parcours avec plusieurs sauts
- de la perte de paquets à hauteur de 25%.

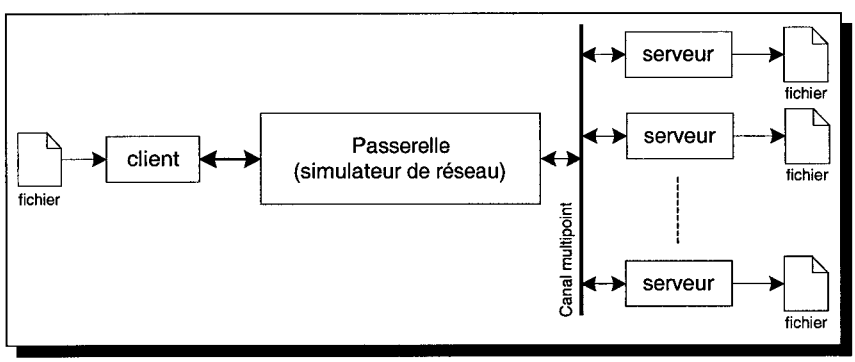
Comme le montre la figure 5.12, l'introduction d'une machine centralisant les communications et les traitements logiciels effectués par cette dernière sur les paquets introduisent de la latence supplémentaire. Sur un réseau LAN 100 Mbps, le temps d'émission est d'environ une seconde (taille du buffer de lecture de 128Ko, cf. figure 5.10(c)) ; uniquement avec l'introduction de la passerelle (sans positionner de limitation au niveau de la latence, de la perte de paquets, etc.), le temps d'émission du fichier est alors d'environ 15 secondes.

Bilan

Dans le contexte le travail qui est le notre, ie. la diffusion de petit groupe, ces différentes expériences montrent aucune dégradation du protocole : le temps d'émission peut être considéré comme constant quelque soit le nombre de récepteurs. cependant, nous pouvons nous demander à partir de quelles limites (bande passante, latence, perte, taille du groupe) les performances se dégradent-elles ? Pour cela des tests complémentaires doivent être réalisés pour répondre à cette question.



(a) Interface du simulateur de réseau



(b) Architecture de l'expérience

FIG. 5.11 – *Utilisation du simulateur de réseau*

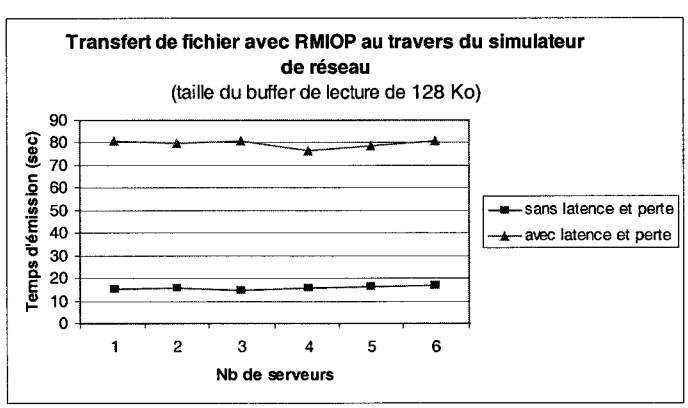


FIG. 5.12 – *Résultats obtenus avec le simulateur de réseau*

5.2 Les services

Au dessus de notre couche de communication, nous avons défini un ensemble de services de haut niveau. Ils ont pour but de gérer l'environnement virtuel 3D et la session de travail. Nous proposons tout d'abord deux services :

- un service de gestion de groupe permettant de gérer les participants présents à la réunion de travail virtuelle. Les participants distants sont affichés sous la forme d'avatars 3D dans l'interface de travail,
- un service de gestion des accès concurrents permettant de prévenir que deux participants ne manipulent, en même temps, une même donnée partagée.

Nous nous plaçons dans un contexte décentralisé : il n'existe aucun serveur en charge de gérer l'environnement virtuel 3D partagé. Nous avons donc mis en œuvre des algorithmes distribués pour ces deux services.

5.2.1 Gestion du groupe de terminaux

Le service de gestion de groupe permet de gérer les différentes autres machines présentes à la session de travail. C'est un service à la fois de bas niveau, utilisé pour fiabiliser les communications, mais aussi de haut niveau, pour gérer les autres utilisateurs dans l'interface de travail (ie. les avatars).

5.2.1.1 Algorithme

Nous nous plaçons dans un contexte où il n'existe aucun serveur pour la gestion de l'environnement virtuel 3D distribué. La gestion du groupe des utilisateurs présents à la réunion de travail virtuel utilise donc un algorithme complètement décentralisé. Dans chaque terminal de travail s'exécute un gestionnaire de groupe et c'est l'ensemble des gestionnaires de groupe qui forment le service de gestion de groupe.

Chaque terminal dispose d'un identifiant (ID) unique correspondant à l'utilisateur qui l'utilise. Le gestionnaire de groupe gère une liste d'identifiants, et à partir de chaque ID, il est capable de donner les informations de l'utilisateur correspondant : un utilisateur dispose de plusieurs informations qui lui sont propres, comme son nom, son adresse IP, sa couleur, son avatar, etc.

Nous avons identifié plusieurs sources de problèmes qui peuvent survenir au cours de la session lors de la gestion du groupe: des problèmes liés à la machine exécutant le terminal, et des problèmes liés à l'infrastructure réseau, créant un partitionnement où les utilisateurs se retrouvent en petits groupes isolés, comme par exemple le phénomène de partition du réseau IRC (en anglais, le terme *split* est employé). Avec le phénomène de partitionnement du réseau, des problèmes complexes liés à l'interface apparaissent: on se retrouve avec deux (ou plus) instances de la même session de travail qui vont évoluer différemment et indépendamment. Certes, nous pouvons les laisser travailler indépendamment, mais que se passe-t-il alors lorsque la connexion réseau est de nouveau rétablie et que les groupes fusionnent: des modifications sur des données partagées peuvent avoir été effectuées dans chacun des groupes, ou encore, un même objet peut être en cours de manipulation par plusieurs personnes au moment de la fusion. Un autre problème lié à l'infrastructure réseau, c'est lorsqu'un terminal A et un terminal B ne peuvent dialoguer ensemble, mais que tous deux sont capables de communiquer avec un terminal C. Les solutions à ce genre de problèmes sont très dépendants des applications et ne peuvent être résolus de manière générique au niveau de la couche de communication. L'algorithme tel que nous le proposons permet uniquement de gérer les problèmes liés au terminal lui même.

À intervalle régulier, le gestionnaire de groupe de chaque terminal émet un message pour se signaler aux autres (messages de *heartbeat*). Ce type de message permet de constituer et de maintenir le groupe de terminaux. Ce message est envoyé en utilisant un protocole non fiable, du fait que l'émetteur ne connaît pas, à priori, tous les récepteurs.

Un gestionnaire de groupe utilise le dernier message de *heartbeat* reçu de chaque terminal afin de résoudre le problème de terminal défectueux. En effet, en notant $T_{signalisation}$ la période d'émission des messages de signalisation, si le dernier message de signalisation d'un terminal a été reçu à un instant plus ancien que $t - T_{signalisation}$ (où t est l'instant présent), on peut alors considérer que le terminal est en défaut. Cependant, cette dernière formule n'est valable que dans le cas d'un réseau parfait. Nous devons prendre en compte le fait que le réseau n'est pas parfait (latence, perte de paquets lors d'un envoi non fiabilisé). Afin de prendre en compte la latence, nous utilisons la formule suivante, $t - (T_{signalisation} + T_{latence})$. En choisissant, $T_{signalisation}$ supérieure à la latence du réseau $T_{latence}$, nous obtenons la formule $t - 2 * T_{signalisation}$ (sans tenir compte de la perte des paquets dont l'envoi est non fiabilisé). Afin de prendre en compte l'éventuelle perte de paquets dont l'envoi est non fiabilisé, nous devons utiliser la formule $t - k * T_{signalisation}$ avec $k > 2$ pour détecter qu'un utilisateur a subitement quitté la session. La valeur de $T_{signalisation}$ ne doit pas être trop élevée afin de ne pas nuire à la réactivité du système.

Un processus de nettoyage par terminal distant détecté est en charge de vérifier qu'un message de signalisation a bien été reçu et de retirer le membre au moindre défaut suspecté. Ce

processus s'exécute avec une période $T_{nettoyage}$ qui en utilisant la formule précédente est égale à $k * T_{signalisation}$ ($k > 2$). Lorsqu'un terminal détecte qu'un terminal doit être retiré de la liste, il prévient alors les terminaux distants pour qu'ils retirent également ce terminal défectueux de leur liste respective.

Pour découvrir les terminaux distants et constituer le groupe, un gestionnaire utilise les différents messages de signalisation qu'il reçoit. La découverte des autres terminaux s'effectue en deux temps. Dans un premier temps, un gestionnaire de groupe découvre les identifiants des autres terminaux à l'aide des messages de signalisation que chacun émet. Pour coopérer, le gestionnaire de groupe doit être capable de fournir à l'interface de travail les informations concernant les utilisateurs distants (comme par exemple, la couleur et le clone de chaque utilisateur). Pour cela, après avoir identifié un terminal distant, un gestionnaire de groupe doit posséder les informations correspondant à l'utilisateur de ce terminal afin de l'ajouter effectivement au groupe. C'est pourquoi chaque gestionnaire dispose de deux listes :

- une première liste contenant les identifiants des terminaux complètement renseignés, ie. le gestionnaire de groupe possède pour chaque terminal de cette liste les informations relatives à leur utilisateur. C'est ce que l'on appelle la liste des terminaux en session. C'est cette liste qui est utilisée pour la fiabilisation de MIOP,
- une deuxième liste contenant les identifiants de terminaux non complètement renseignés, ie. les terminaux pour lesquels le gestionnaire ne dispose pas encore des informations nécessaires pour pouvoir les intégrer dans l'interface de travail. Il ne peut pas encore les ajouter au groupe, ie. à la liste des terminaux en session.

Lorsqu'un terminal reçoit un message de signalisation d'un terminal distant, nous avons deux possibilités :

- si c'est un message de signalisation d'un terminal qu'il connaît, ie. dont l'identifiant est dans la liste des terminaux en session, il met alors à jour le *heartbeat* correspondant utilisé dans la détection de défauts. Cela permet de réinitialiser le processus de nettoyage en charge de gérer ce terminal distant,
- si c'est un message de signalisation d'un terminal qu'il ne connaît pas encore, ie. dont c'est la première fois qu'il reçoit un message de signalisation, un dialogue s'établit, en point-à-point, entre les deux gestionnaires de groupe pour s'échanger plusieurs informations. D'une part, ils s'échangent les informations concernant leurs utilisateurs respectifs permettant ainsi de construire la liste des terminaux en session. D'autre part, ils s'échangent également leur liste respective des identifiants de terminaux en session permettant ainsi d'enrichir la liste des terminaux "potentiels" : chaque gestionnaire de groupe fusionne sa liste de terminaux

"potentiels" avec la liste qu'il reçoit de l'autre. Il y a équilibre quand la liste des terminaux potentiels que possède un gestionnaire de groupe est vide, ie. il n'existe pas de terminal pour lequel le gestionnaire de groupe ne possède pas les informations le concernant. Un processus est en charge de vérifier que cette liste est effectivement vide : au moment de la vérification régulière, si la liste des terminaux potentiels n'est pas vide, le gestionnaire de groupe va forcer la rencontre en cherchant à contacter directement le(s) terminal(aux) en suivant la même procédure que lorsqu'il reçoit un message de signalisation d'un terminal qu'il ne connaît pas encore.

5.2.1.2 Implémentation

Le gestionnaire de groupe a été implémenté sous la forme d'un objet CORBA. La figure 5.13 présente l'interface de cet objet décrite en utilisant le langage IDL-OMG.

```
typedef string GroupManagerID;

struct GroupMember {
  GroupManagerID id;
  any extra;
};

typedef sequence<GroupManagerID> ListOfID;

interface GroupManager {
  void register_member(in GroupMember local, in ListOfID local_list_of_ID,
                       out GroupMember remote, out ListOfID remote_list_of_ID);

  oneway void unregister_member(in GroupManagerID id);

  oneway void heartbeat(in GroupManagerID id);
};
```

FIG. 5.13 – *Définition IDL-OMG de l'interface du gestionnaire de groupe*

Chaque terminal exécute un objet dont nous venons de décrire l'interface. Le gestionnaire est connecté au bus CORBA et possède deux références : une référence de groupe (référence (R)MIOP), identique pour tous les gestionnaires de groupe, et une référence propre à chaque gestionnaire utilisé pour les communications point à point, ie. entre deux gestionnaires de groupe (référence IIOP). Il communique avec ses homologues distants en utilisant un mécanisme d'appel de méthodes à distance :

- pour se signaler aux autres terminaux, chaque gestionnaire de groupe appelle la méthode `heartbeat(...)` sur la référence de groupe. Lorsqu'un terminal se signale aux autres, il ne connaît pas à priori les terminaux qui vont lui répondre, c'est pourquoi le canal de communication utilisé pour la signalisation n'est pas fiabilisé. Pour cela, nous utilisons le protocole MIOP non fiabilisé pour transporter l'appel de méthode aux différents membres

du groupe,
- comme nous l'avons mentionné précédemment, lorsque deux terminaux sont mis en relation pour la première fois, une liaison fiable est établie entre eux afin qu'il puisse s'échanger plusieurs informations, comme leur identité respective et la liste des autres terminaux qu'ils connaissent déjà, etc. Pour cette phase de l'algorithme, les deux gestionnaires de groupe utilisent leurs références IIOP pour dialoguer en utilisant la méthode `register_member(...)`,
- la méthode `unregister_member(...)` est utilisé par un terminal pour signaler aux autres membres du groupe son départ. Pour cela, il appelle cette méthode sur la référence de groupe et utilise le canal RMIOP pour transporter la requête.

5.2.2 Gestion des accès concurrents

Comme nous l'avons présenté au cours de l'état de l'art, il existe deux méthodes pour gérer l'accès aux données partagées :

- l'approche optimiste où l'on part du principe qu'il n'y aura pas de conflit. L'utilisateur commence à modifier la donnée partagée sans réaliser de contrôle, et si l'on rencontre un conflit (ie. plusieurs personnes veulent modifier en même temps la même donnée partagée), on le résout en annulant les modifications effectuées,
- l'approche pessimiste où l'on préfère prévenir les conflits en assurant, qu'à un instant donné, une seule personne peut avoir accès à la donnée partagée pour la modifier.

Dans notre contexte de travail, nous souhaitons offrir aux utilisateurs du système la meilleure compréhension possible de l'activité : un utilisateur doit être capable de comprendre ses propres actions ainsi que les actions des utilisateurs distants. L'approche optimiste est susceptible de perturber les utilisateurs dans leur compréhension de l'activité : en cas de conflit, le retour visuel ne va pas correspondre aux actions des utilisateurs puisque le système va revenir en arrière pour annuler les manipulations. C'est pourquoi, nous préférons utiliser une approche pessimiste pour gérer les accès concurrents aux données partagées : avant toute action d'un utilisateur sur une donnée partagée, son terminal de travail vérifie qu'aucun autre utilisateur n'est pas déjà en train de la modifier.

D'autre part, le mécanisme de gestion des accès concurrents que nous proposons ne se veut pas être un mécanisme bas niveau ayant pour fonction de protéger la donnée partagée dès qu'une écriture se produit (comme par exemple, le mécanisme proposé par la plate-forme technologique Blaxxun). Une telle approche serait bien trop coûteuse et nuirait à l'interactivité du terminal de travail. Nous préférons nous orienter vers une solution s'intégrant au mécanisme d'interaction à

trois phases que propose notre interface de travail (cf. 4.1.3, page 119). Ainsi, nous protégeons les actions de l'utilisateur sur des données partagées plutôt que la donnée partagée en elle-même.

5.2.2.1 Algorithme

Chaque action partagée (ie. une interaction sur une donnée partagée) est associée à un jeton : seul le terminal possédant le jeton associé à l'action partagée a le droit de l'effectuer. Nous avons donc mis en place un gestionnaire de jetons dans notre terminal. Le service tel que nous l'avons conçu fonctionne sans serveur central : chaque terminal possède un gestionnaire de jetons représentant le service de gestion des jetons et c'est l'ensemble des gestionnaires de jetons qui forment le service. Les différents terminaux (ie. les différents gestionnaires de jetons) dialoguent entre eux pour s'échanger ou créer les jetons.

Chaque gestionnaire de jetons possède sa propre liste de jetons correspondant aux différents jetons que le terminal local possède. Chaque jeton présent dans la table a un état qui est soit "verrouillé", ie. l'utilisateur a déclenché l'action associée au jeton, soit "déverrouillé", ie. l'utilisateur est le dernier utilisateur à avoir manipulé la donnée associée au jeton, mais à l'heure actuelle, personne n'est en cours de manipulation.

Quand un utilisateur souhaite déclencher une action partagée, le terminal cherche à acquérir le jeton qui lui est associée. Pour cela, il s'adresse au gestionnaire de jeton local. Lors d'une demande locale, le gestionnaire de jeton regarde tout d'abord dans sa table si le jeton demandé s'y trouve ou pas. Si le jeton y est présent, à partir de l'état du jeton, il décide si il peut accepter ou non la demande. Si le jeton ne se trouve pas dans la table locale, il contacte alors les terminaux distants. C'est à partir des réponses qu'il reçoit qu'il décide d'accepter ou de refuser la demande (les automates de fonctionnement sont présentés dans l'annexe F, page 256) :

- un terminal distant lui répond qu'il refuse sa demande. On trouve deux origines à cette réponse : soit le jeton est détenu et verrouillé par ce terminal distant et l'action partagée est déjà en cours d'exécution ; soit le terminal distant est déjà en train d'effectuer une demande pour ce jeton. Après un refus, puisque notre gestionnaire de jetons est intégré au mécanisme d'interaction du terminal, l'utilisateur est libre d'effectuer ultérieurement une nouvelle demande de jeton, ie. de chercher à re-déclencher l'action,
- un terminal distant accepte de lui céder le jeton, ie. il possède le jeton dans l'état non verrouillé, par conséquent personne n'est en train d'exécuter l'action partagée associée au jeton. L'échange entre les deux terminaux peut être réalisé. Le terminal "donneur" cède le jeton au terminal "demandeur",

- tous les terminaux distants répondent qu'ils ne connaissent pas le jeton demandé (c'est la réponse "*jeton inconnu*"). En effet, une telle situation peut se produire notamment au démarrage de la session de travail : comme nous nous plaçons dans un contexte sans serveur, il n'existe aucun jeton au lancement des différents terminaux. Cela peut également se produire lorsque le terminal qui possédait auparavant le jeton a un problème. Les jetons sont créés à la volée, au fur et à mesure des demandes. Lorsque le demandeur reçoit la réponse "*jeton inconnu*" de tous les terminaux distants, il peut alors créer le jeton et le verrouiller afin d'autoriser localement l'action partagée qui lui est associée,
- des terminaux distants lui répondent qu'ils ne connaissent pas le jeton demandé mais il manque des réponses. Dans le doute, la demande de jeton est refusée. L'interface de travail via l'utilisateur qui la pilote réitérera ultérieurement sa demande.

Une fois que l'action de l'utilisateur est terminée, le terminal de travail le signale au gestionnaire de jetons local qui libère alors le jeton : l'état du jeton passe alors de "verrouillé" à "déverrouillé". Il est à noter que dans le schéma de fonctionnement présenté ci-dessus, lors d'une demande de jeton, une collaboration est établie entre le gestionnaire de jetons et le gestionnaire de groupe : en effet, cela permet au gestionnaire de jetons de connaître la liste des terminaux desquels il doit attendre une réponse.

5.2.2.2 Intégration au mécanisme d'interaction de l'interface de travail

Nous souhaitons proposer un système interactif afin de fournir aux utilisateurs les meilleures conditions pour collaborer et comprendre l'activité, nous intégrons le mécanisme de gestion de jeton au mécanisme d'interaction de notre interface de travail. Pour cela, c'est dans les phases de sélection et de dé-sélection qu'ont lieu respectivement l'acquisition et la libération du jeton associé à une action partagée.

Dans l'interface de travail, un objet est modifié via la manipulation des `Sensors` VRML 97. Alors que les différents `Sensors` proposés par la norme VRML 97 ne permettent qu'une manipulation au travers d'un périphérique 2D (la souris), nous proposons un nouveau `Sensor`, que nous appelons `Sensor3D`, qui est plus adapté à une manipulation en 3D. La définition de ce nouveau nœud `Sensor3D` sort du cadre de cette thèse, cependant le lecteur trouvera la spécification de son interface en annexe (cf. annexe D, page 244). Un `Sensor3D` est associé à un objet 3D de l'interface de travail par l'intermédiaire du mécanisme de `ROUTE` VRML 97 : on peut rediriger les sorties du `Sensor` vers des champs d'autres nœuds VRML 97.

Lors de la phase de sélection, le terminal de travail détermine l'objet 3D (et par conséquent

le Sensor3D) que l'utilisateur désire manipuler. Après avoir cliqué, l'objet est sélectionné et le `Sensor3D` est dit "actif" : les différentes manipulations qu'opére l'utilisateur modifient l'état du `Sensor3D` et les changements effectués au niveau du `Sensor3D` sont propagés aux autres objets de la scène en utilisant le mécanisme de `ROUTE` VRML 97.

Comme le montre la description VRML 97 du nœud `Sensor3D`, il comporte un champ nommé `lockName` : la valeur de ce champ correspond au nom du jeton à acquérir pour pouvoir manipuler le `Sensor3D`. Lorsque la valeur de ce champ est vide (`""`), aucun contrôle d'accès n'est effectué. Au contraire, lorsque le concepteur spécifie une valeur, notre terminal utilise le gestionnaire de jetons avant de permettre la manipulation du `Sensor3D`.

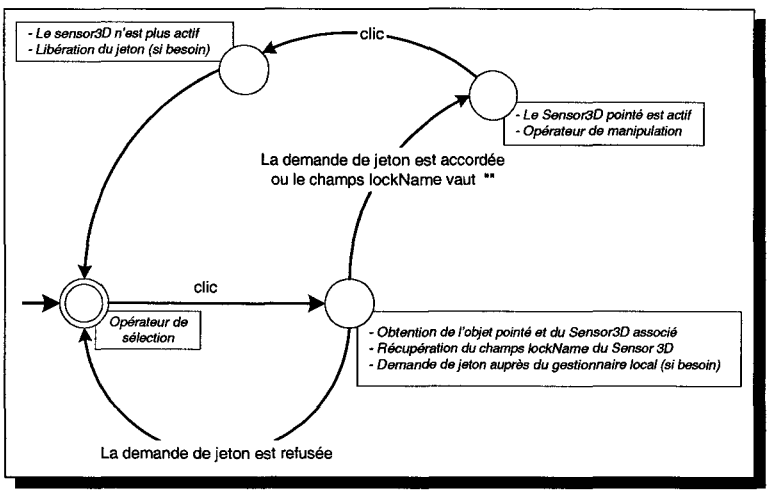
L'automate d'interaction de notre terminal de travail est présenté sur la figure 5.14(a). Il intègre la gestion des phases d'acquisition et de libération des jetons permettant de protéger les actions de l'utilisateur sur les objets 3D partagés. Au moment de la sélection, le terminal utilise le champ `lockName` du nœud `Sensor3D` pointé et questionne le gestionnaire de jetons pour déterminer si il peut rendre "actif" le `Sensor3D` et permettre à l'utilisateur de manipuler l'objet. L'automate d'interaction peut être modifié comme montré sur la figure 5.14(b) afin d'y intégrer des états permettant de retranscrire, sous la forme de retours visuels, les différentes étapes de l'automate pour que les utilisateurs comprennent ce qui se passe dans l'interface de travail.

5.2.2.3 Implémentation

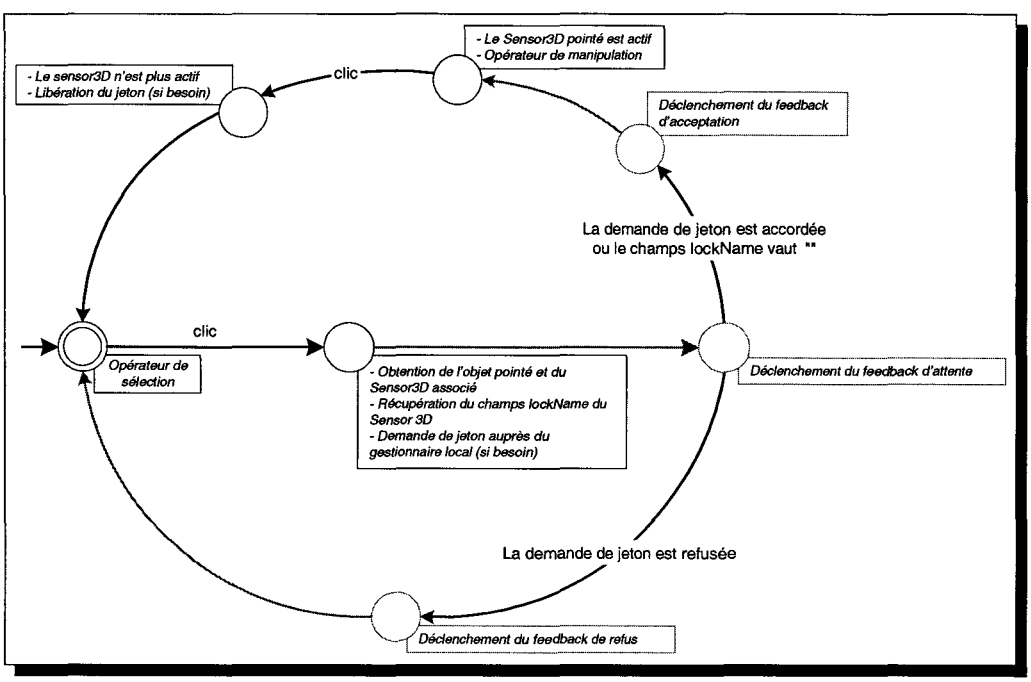
Comme pour le gestionnaire de groupe, le gestionnaire de jetons a été implémenté sous la forme d'un objet CORBA dont l'interface IDL-OMG est décrite sur la figure 5.15. Un gestionnaire de jetons s'exécute dans chaque terminal de travail. Les différents gestionnaires dialoguent entre eux en utilisant le mécanisme d'appel de méthode à distance. Les gestionnaires de jetons s'exécutant dans les différents terminaux sont connectés au bus CORBA, chacun d'entre eux possédant deux références :

- une référence RMIOP pour les communications de groupe, les gestionnaires de jetons des terminaux en session formant un groupe,
- une référence IIOP pour les communications point à point, ie. entre deux gestionnaires.

Les méthodes `lock()` et `unlock()` sont utilisées uniquement par un terminal pour dialoguer avec son gestionnaire de jetons local. Les autres méthodes sont utilisées par le gestionnaire pour dialoguer avec ses homologues distants. Lorsqu'un gestionnaire de jetons demande aux gestionnaires distants un jeton, il utilise la méthode `requestGetToken(...)` en l'appelant avec la



(a) sans retour visuel d'interaction



(b) avec retour visuel d'interaction

FIG. 5.14 – *Automates d'interaction de l'interface de travail intégrant le mécanisme d'acquisition et de libération de jeton*

```
typedef string TokenManagerID;
typedef string TokenID;
enum GetTokenReply {unknown, granted, refused};

interface TokenManager {
  void   lock(in TokenID token);
  void   unlock(in TokenID token)
  oneway void requestGetToken(in TokenManager requester, in TokenID token);
  void   replyGetToken(in TokenID token,
                       in TokenManager replier,
                       in GetTokenReply reply);
}; // End of interface TokenManager
```

FIG. 5.15 – *Définition IDL-OMG de l'interface du gestionnaire de jetons*

référence du groupe de gestionnaires de jetons. Les gestionnaires de jetons distants lui répondent, en communication point à point, en utilisant la méthode `replyGetToken(...)`.

5.3 Bilan

Dans ce chapitre, nous avons présenté les deux mécanismes de communications permettant de maintenir la cohérence au sein des différents groupes "logiques" représentant les données partagées de l'environnement virtuel 3D.

Pour cela, nous proposons un premier canal de communication offrant une cohérence forte. Ce canal utilise le paradigme d'appel de méthode à distance: dès qu'un membre d'un groupe logique est modifié, les autres membres du même groupe sont prévenus à l'aide d'un appel de méthode afin de maintenir la cohérence au sein du groupe logique. Pour cela, nous proposons un mécanisme de fiabilisation du protocole MIOP défini par l'OMG. C'est ce que nous appelons le protocole RMIOP.

Nous proposons également un deuxième canal de communication offrant une cohérence relâchée. Entre les différents membres d'un même groupe logique, nous mettons en place un flot de communication au travers duquel les copies peuvent dialoguer entre elles. Les données émises sur un flot multimédia ne sont pas fiabilisées. Pour contrôler les flots multimédia, nous utilisons le service de contrôle de flux, défini par l'OMG, et dont le fonctionnement a été adapté à notre architecture de gestion décentralisée. Dans ce chapitre, nous avons uniquement défini le protocole de contrôle des flux sans aborder le protocole utilisé au niveau des flots pour transporter les données: il est possible d'utiliser un protocole adapté au transport de données "temps réel", comme par exemple le protocole RTP (*Real-time Transport Protocol*). De même, nous n'avons pas défini de protocole (payload) permettant de coder les données transportées sur les flots.

Pour gérer l'EVC 3D, nous proposons deux services utilisant les canaux de communication que nous avons définis. Du fait que notre architecture de gestion est décentralisée, des algorithmes distribués ont été mis en oeuvre pour implémenter ces deux services.

Le service de groupe est en charge de gérer les participants présents à la réunion de travail virtuelle. Il est utilisé à haut niveau par l'interface de travail, pour présenter les utilisateurs distants sous la forme d'avatars 3D. Il est également utilisé à bas niveau par le protocole RMIOP pour sa fiabilisation.

Le service de gestion des accès concurrents permet de nous assurer qu'un seul participant à la fois ne peut réaliser une action sur un objet 3D partagé. Pour des raisons d'efficacité, l'utilisation de service est intégrée au mécanisme d'interaction proposé par notre interface de travail.

Le système que nous proposons doit permettre aux concepteurs d'applications coopératives de se concentrer sur le contenu. Nous souhaitons donc masquer les différents problèmes techniques sous-jacents à la distribution des utilisateurs et au maintien de la cohérence dans l'environnement virtuel 3D partagé. C'est pourquoi, dans le chapitre suivant, nous proposons une abstraction de haut niveau de la plate-forme de communication que nous venons de présenter, permettant aux concepteurs d'application de concevoir et d'intégrer des objets 3D partagés dans l'interface de travail coopératif.

Chapitre 6

Mécanisme de description des objets 3D partagés en VRML 97

Sommaire

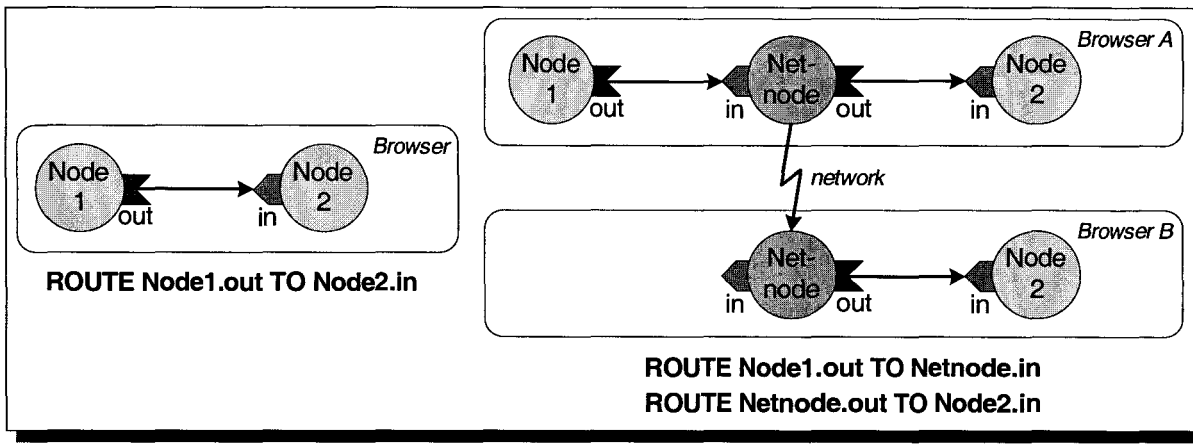
Afin de masquer l'utilisation de la plate-forme de communication présentée dans le chapitre précédent, nous proposons à l'utilisateur final de décrire les différents objets partagés présents dans l'interface de travail. Les différents objets 3D présents dans l'interface sont décrits en utilisant le langage VRML 97. Un objet 3D partagé expose plusieurs propriétés susceptibles d'être modifiées par n'importe quel participant. Dans ce chapitre, nous proposons un mécanisme de description de partage basé sur un nouveau paradigme de substitution. Dans un premier temps nous présentons le paradigme de substitution, puis dans un second temps, nous présentons la mise en œuvre en utilisant le langage VRML 97 : ainsi, nous incluons la description de partage dans le même fichier que la description géométrique. Nous offrons plusieurs nouveaux nœuds VRML 97 afin de décrire un objet 3D partagé et ses différentes propriétés exposées [LDPDG+02].

6.1 Le paradigme de substitution

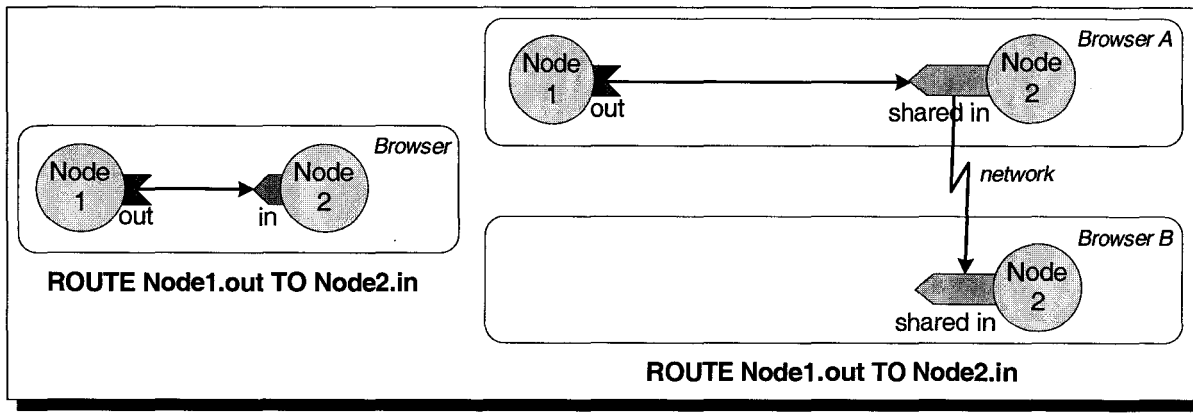
Dès lors que la description du partage est intégrée au graphe de scène VRML 97, les différentes propositions multi-utilisateurs en VRML 97 reposent toutes sur le paradigme d'insertion. Nous avons vu que l'utilisation d'un paradigme d'insertion pour le partage n'est pas sans poser de problèmes au niveau du maintien de la cohérence des `Sensors` de manipulation ainsi qu'au niveau de la gestion des applications externes et du partage dynamique (cf. 4.3.1, page 127).

Pour résoudre ces différents problèmes inhérents au paradigme d'insertion, nous introduisons un nouveau paradigme pour le partage de données : c'est ce que nous appelons "le paradigme de substitution". Le principe est le suivant : plutôt que d'insérer de nouveaux nœuds dans le graphe de scène en charge d'intercepter les différents évènements le long des chemins d'animation (`ROUTES`) et de les transmettre à distance (les nœuds insérés intègrent l'implémentation pour la communication), nous proposons d'intégrer à l'intérieur même des différents champs la gestion du partage et des notifications à distance (cf. figure 6.1). Pour chaque champ, nous disposons de deux implémentations différentes : une implémentation "normale" et une implémentation "partagée" capable de communiquer avec ses copies distantes. Lorsque notre terminal détecte qu'un champ doit être partagé, nous substituons l'implémentation partagée à l'implémentation normale. Aucun nœud de dérivation n'est nécessaire, le champ partagé effectuant lui même les notifications distantes dès qu'il est modifié. Il est à noter que l'approche proposée par le paradigme de substitution n'est pas réalisable avec un navigateur VRML 97 classique.

Avec notre approche, nous sommes capables de synchroniser l'état d'un `Sensor` : la valeur de sortie d'un `Sensor` peut être synchronisée en la substituant par son équivalent partagée (cf. figure 6.2(b)). Il est à noter que le `Sensor` doit disposer d'une implémentation spécifique permet-



(a) paradigme d'insertion

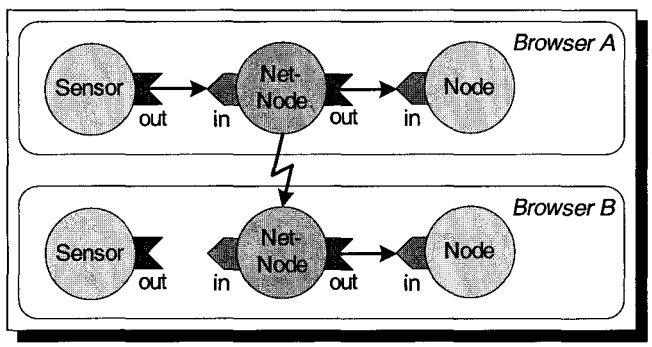


(b) paradigme de substitution

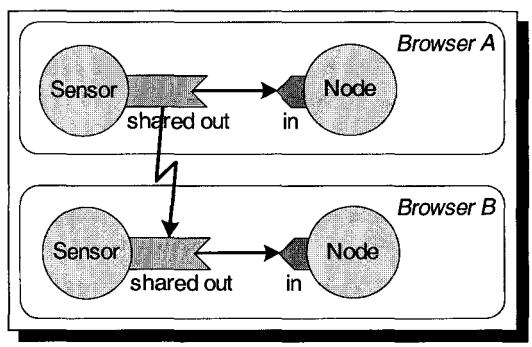
FIG. 6.1 – *Comparaison des paradigmes d'insertion et de substitution*

tant de prendre en compte le paradigme de substitution : il faut une implémentation capable de recalculer l'état interne du `Sensor` à partir de la valeur de ses sorties lorsque l'une d'entre elles est modifiée par le réseau. Comme montré sur la figure 6.2(a), avec le paradigme d'insertion, il est impossible de réaliser simplement la synchronisation des sorties du `Sensor` (il est à noter que nous présentons en annexe C une solution complexe utilisant des scripts pour la synchronisation des sorties du `Sensor`).

De plus, nous sommes également capables de gérer très facilement le partage dynamique de champs et leur manipulation depuis une application externe tout en maintenant la cohérence des données partagées. Avec notre approche, les changements effectués via l'application externe sont effectivement propagés à distance permettant ainsi de conserver la cohérence des données partagées (cf. figure 6.3(c)). Comme montré sur la figure 6.3(b), avec le paradigme d'insertion, il est impossible de gérer, de façon transparente, le partage dynamique de champs et leur manipulation depuis l'application externe tout en maintenant la cohérence des données partagées.

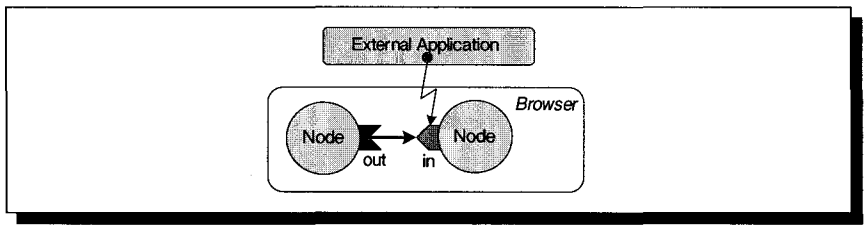


(a) paradigme d'insertion

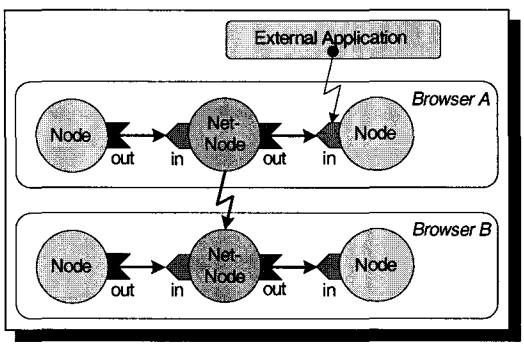


(b) paradigme de substitution

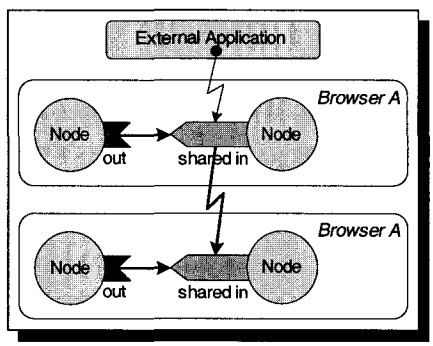
FIG. 6.2 – *Synchronisation de l'état d'un* `Sensor`



(a) avant le partage



(b) après avec le paradigme d'insertion



(c) après avec le paradigme de substitution

FIG. 6.3 – *Partage dynamique et application externe*

6.2 Mise en œuvre en VRML 97

Dans la section précédente, nous avons vu le principe de fonctionnement de notre paradigme de substitution. Dans cette section, nous présentons sa mise en œuvre en utilisant les fonctionnalités offertes par le langage VRML 97.Contrairement aux approches classiques comme (Core) Living Worlds ou VSPLUS, nous proposons de placer la description de partage en dehors de la description géométrique. De cette façon, l'utilisation de notre *framework* reste simple : nous permettons au concepteur de réutiliser des contenus VRML 97 existants auxquels il peut ajouter la description de partage sans avoir besoin de modifier l'existant (ie. la géométrie et l'interaction), rendant ainsi plus facile l'édition de contenus multi-utilisateurs. La description de partage est placée en début de fichier, en dehors de la description géométrique.

Notre extension se compose de plusieurs prototypes de nœuds définis en utilisant la fonctionnalité d'`EXTERNPROTO` fournie par le langage VRML 97 :

- un noeud de création de hiérarchie de partage (`SharedSet`) : d'un point de vue sémantique, un nœud `SharedSet` représente un objet 3D partagé,
- des noeuds de description du partage (`SharedNodes`), permettant de spécifier les champs à partager, ie. à substituer : d'un point de vue sémantique, un nœud `SharedNode` représente

une propriété exposée par un objet 3D partagé.

Le mécanisme d'`EXTERNPROTO` nous permet de conserver la compatibilité avec des navigateurs VRML 97 existants. Ainsi, un objet multi-utilisateurs conçu à l'aide de notre proposition reste exploitable (chargement, affichage et interaction) par un navigateur VRML 97 classique, bien évidemment sans gérer le partage. En effet, la norme VRML 97 spécifie comment réagir lors de la lecture d'un `EXTERNPROTO` qui n'est pas reconnu par le navigateur : celui-ci est alors remplacé par un nœud vide lors de son instanciation dans le graphe de scène.

Contrairement à la proposition (Core) Living Worlds, notre extension ne propose pas de notion de `Zone` permettant de partitionner l'espace 3D afin de réduire la charge réseau. Une telle fonctionnalité n'est pas nécessaire dans le cadre de notre activité où les réunions ne regroupent qu'un petit nombre d'acteurs dans une unique pièce.

6.2.1 Le concept de `SharedNode`

Nous devons être capables de spécifier quels sont les propriétés exposées par un objet VRML 97 partagé. Pour cela, nous proposons le concept de `SharedNode`. Dans notre *framework*, `SharedNode` est le terme générique que l'on donne aux nœuds de description du partage. Un tel nœud permet au concepteur de spécifier qu'un champ d'un objet VRML 97 est partagé, ie. que sa valeur est synchronisée entre les différentes instances de l'interface de travail. Les `SharedNodes` sont l'équivalent des `NetworkStates` de (Core) Living Worlds, des `NetNodes` de VSPLUS, et des `SharedEvents` de la technologie multi-utilisateurs de Blaxxun. Ainsi, comme nous l'avons vu précédemment, à chaque type de base VRML 97 (`SFBool`, `SFColor`, etc), on associe un `SharedNode` (`SharedSFBool`, `SharedSFColor`, etc).

```
EXTERNPROTO SharedXXX [
  field          SFString alias ""
  exposedField SFString mode "public"
  field          SFString how "action"
  field          SFString lockName ""
][urn:lifl:proto/sharedxxx]
```

```
EXTERNPROTO SharedSFBool [
  field          SFString alias ""
  exposedField SFString mode "public"
  field          SFString how "action"
  field          SFString lockName ""
][urn:lifl:proto/sharedsfbool]
```

FIG. 6.4 – *Prototype d'un nœud `SharedNode` (XXX remplace un type de base, comme par exemple `SFBool` sur la figure de droite)*

La figure 6.4 présente le prototype générique d'un `SharedNode`. Il est constitué des champs suivants :

– le champ `alias` permet de désigner le nom du champ à partager, ie. le champ d'un nœud

contenu dans la géométrie 3D. La valeur de la chaîne de caractères est du type `<nom du DEF>.<nom du champ>`. Notre terminal VRML 97 utilise cette information pour remplacer le champ pointé en son équivalent partagé,
- le champ `how` permet de spécifier le degré de cohérence à utiliser. Notre terminal utilise cette information pour déterminer le canal de communication à utiliser pour envoyer les mises à jour (voir ci-dessous pour les différentes valeurs que peut l'on peut affecter à ce champ),
- le champ `mode` permet de changer, à la volée, le partage. Ainsi, il peut très facilement être activé (dans ce cas, le champ `mode` a la valeur `"public"`) ou désactivé (dans ce cas, le champ `mode` a la valeur `"private"`),
- le champ `lockName` permet d'indiquer si la donnée est soumise à un contrôle d'accès afin d'éviter les conflits d'accès.

Pour le champ `how`, nous proposons actuellement trois valeurs possibles permettant de prendre en compte les différents canaux de communication proposés par notre plate-forme :

- la valeur `"action"` indique que la donnée correspond à une action ponctuelle. Le canal RMIOP de notre plate-forme de communication est utilisé pour transmettre les mises à jour aux postes distants,
- la valeur `"flow"` indique que la donnée est un flot d'informations infini dans le temps, comme par exemple le son ou la vidéo. Pour ce type de donnée, le canal de flot de données de notre plate-forme de communication est utilisé,
- la valeur `"animation"` indique que la donnée est un flot d'informations fini dans le temps, comme par exemple la manipulation interactive d'un objet (translation, rotation). Dans ce cas, les étapes intermédiaires de l'animation ne sont pas importantes, seul le résultat compte et permet de maintenir la cohérence du monde partagé. Pour ce type de donnée, nous utilisons une combinaison des deux canaux de communication offerts par notre plate-forme (flots de données, RMIOP).

Dans un nœud de partage, lorsque la valeur du champ `how` est `"flow"` ou `"animation"`, le champ désigné par la valeur du champ `alias` est alors considéré comme un producteur/consommateur de flot, ie. un `FlowDevice` vu dans le chapitre précédent (cf. 5.1.2.1, page 142). Le champ désigné par la valeur du champ `alias` est donc associé à un flot élémentaire (`FlowEndpoint`) via lequel il est capable d'envoyer ou recevoir des données.

Dans l'interface de travail, les modifications de données partagées se font à l'aide de `Sensor3D` placés sur les objets que le concepteur autorise à manipuler. Comme nous l'avons vu dans le chapitre précédent, un contrôle d'accès est réalisé au moment de la sélection et de l'activation

du `Sensor3D` choisi en utilisant le champ `lockName` défini dans le nœud `Sensor3D` (cf. 5.2.2.2, page 161). Comme nous le verrons par la suite, le champ `lockName`, spécifié au niveau du nœud de partage, est utilisé par les applications externes pour prévenir les accès concurrents quand elles veulent manipuler un champ partagé (cf. 7.2.4, page 192). Cela nécessite un travail de la part du concepteur d'objets 3D. Il doit respecter certaines règles lors de la conception multi-utilisateurs : il doit affecter au champ `lockName` d'un nœud de partage la même valeur que celle contenue dans le champ `lockName` du `Sensor3D` qui lui est relié par le mécanisme de `ROUTE`. Ainsi, lorsqu'un utilisateur manipule une donnée partagée depuis l'interface de travail via le `Sensor3D`, aucune application externe ne peut modifier la même donnée partagée : le jeton est déjà verrouillé, et l'application verra sa demande de jeton rejetée puisque elle cherchera à acquérir le même jeton. Cette solution doit être transitoire : dans la mesure où nous voulons faciliter la conception des objets 3D utilisés par notre interface de travail, nous devons à terme fournir un algorithme automatique permettant de remonter les `ROUTES` VRML 97. Pour un champ donné, l'algorithme doit être capable de retrouver le `Sensor3D` qui lui est relié, et par conséquent, le champ `lockName` contenu dans le `Sensor3D`. Il est à noter que lorsque la valeur du champ `lockName` est vide (`""`), aucun contrôle d'accès n'est réalisé. Au contraire, lorsque le concepteur spécifie une valeur, notre terminal utilise le service de contrôle d'accès avant de réaliser la modification.

6.2.2 Le concept de `SharedSet`

Avec la notion de `SharedNode`, nous pouvons spécifier les propriétés exposées par un objet VRML 97 partagé. Notre extension multi-utilisateurs VRML 97 inclut également le prototype de nœud `SharedSet` qui, d'un point de vue sémantique, peut être assimilé à un objet 3D partagé. Dans notre philosophie, un fichier VRML 97 contient un objet 3D de l'interface.

```
EXTERNPROTO SharedSet [
  exposedField SFString mode "public"
  exposedField MFNode children []
  eventIn MFNode addChildren
  eventIn MFNode removeChildren
][urn:lifl:externproto/sharedset]
```

FIG. 6.5 – *Prototype du nœud `SharedSet`*

La figure 6.5 présente la syntaxe du noeud `SharedSet`. Il est composé des champs suivants :

- le champ `children` contient différents nœuds. Le fonctionnement de ce champ est similaire à celui proposé par les nœuds de groupe classiques définis dans la norme VRML 97 (par exemple, `Group` et `Transform`) : les champs `addChildren` et `removeChildren` sont utilisés

pour modifier le contenu du champ `children`. Les fils d'un `SharedSet` sont des noeuds de partage (`SharedNodes`), ce qui d'un point vue sémantique est équivalent à dire qu'un objet 3D partagé expose plusieurs propriétés,

- le champ `mode` est très utile pour désactiver le partage, en une seule fois, de tous les fils d'un même `SharedSet`. Si la valeur de ce champ est égale à `"private"`, cela inhibe le partage des fils du `SharedSet`, et ce quelque soit la valeur de leur champ `mode`.

Un `SharedSet` permet de définir un objet 3D partagé. En reprenant la terminologie du chapitre précédent concernant les flux (cf. 5.1.2.1, page 142), le `SharedSet` est assimilé à un périphérique multimédia (`MMDevice`) auquel est associé un `StreamEndpoint`. Comme nous l'avons dit précédemment, les différents `SharedNodes` que contient un `SharedSet` et qui sont configurés pour une communication par flux multimédia (ie. leur champ `"how"` est positionné à `"flow"` ou `"animation"`) sont considérés comme des producteurs/consommateurs de flots élémentaires (`FlowDevices`) rattachés à un flot élémentaire (`FlowEndpoint`) contenu dans le flux (`StreamEndpoint`) associé au `SharedSet`.

On peut imaginer utiliser le nœud `SharedSet` pour gérer la persistance des objets partagés. En effet, à partir des informations contenues dans un nœud `SharedSet`, il est possible de déterminer les informations à envoyer lorsqu'un nouvel utilisateur entre en session ou bien les informations à sauvegarder pour conserver l'état d'un objet 3D partagé. Le nouveau terminal en session doit recevoir des autres terminaux la valeur des propriétés exposées par chaque objet partagé, ie. pour chaque `SharedSet`, la valeur courante de ses différents `SharedNodes` fils. Pour cela, nous pouvons utiliser un mécanisme de sérialisation appliqué à chaque `SharedSet`. Ce mécanisme de sérialisation appliqué à un nœud `SharedSet` peut également être utilisé pour sauvegarder l'état courant d'un objet 3D partagé afin de le recharger ultérieurement.

Tel que nous l'avons conçu, avec le nœud `SharedSet`, il est possible de créer une hiérarchie de partage. En effet, rien n'empêche de placer dans le champ `children` d'un nœud `SharedSet` d'autres `SharedSet`, ce qui, d'un point de vue sémantique, signifie qu'un objet 3D peut contenir d'autres objets 3D, comme c'est par exemple le cas lors d'un assemblage. Comme nous l'avons dit précédemment, chaque `SharedSet` est associé à un flux (`StreamEndpoint`) : lorsqu'un `SharedSet` est le fils d'un autre autre `SharedSet`, les deux `SharedSet` restent associés à leur flux respectif (un flux contient uniquement des flots élémentaires). Le champ `mode` peut toujours être utilisé : il permet de modifier, en une fois, le mode de partage de tous les fils (`SharedNodes` et `SharedSets`) d'un `SharedSet`.

```
#VRML V2.0 utf8

# Some external definitions
EXTERNPROTO ...

# Sharing description
SharedSet {
 children [
  SharedSFRotation {
   alias "S.rotation_changed"
   lockName "rotation"
}]}

# VRML97 scene graph
DEF T Transform {
 children [
  DEF S Sensor3D { lockName "rotation" },
  Shape {
   appearance Appearance { material Material { diffuseColor 1 0 0 }}
   geometry Box { size 1 1 1 }
}]}

ROUTE S.rotation_changed TO T.rotation
```

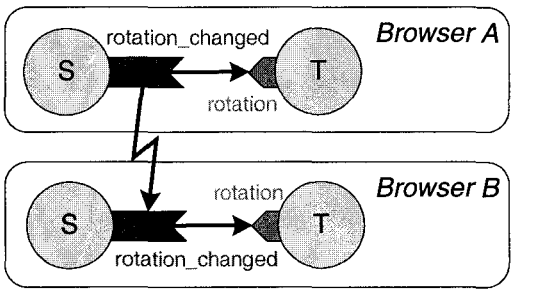


FIG. 6.6 – *Un exemple de contenu VRML 97 multi-utilisateurs créé avec notre extension (en gras, les parties provenant du contenu mono-utilisateur)*

6.2.3 Un exemple

La figure 6.6 montre un exemple de contenu multi-utilisateurs conçu avec notre proposition. Encore une fois, afin de pouvoir comparer plus facilement notre proposition (basée sur le paradigme de substitution) avec les autres (basées sur le paradigme d'insertion), nous avons conservé le même exemple que celui présenté précédemment (cf. figure 3.3, page 80, cf. figure 3.5, page 86 et cf. figure 3.4, page 84). La description du partage est placée en dehors de la description géométrique, en entête du fichier. Grâce au mécanisme de substitution, les `ROUTES` existantes sont conservées intactes. Pour pouvoir manipuler le cube dans notre interface de travail 3D, nous utilisons notre propre *sensor*, `Sensor3D`. Nous partageons le champ de sortie `rotation_changed` du `Sensor3D` afin de conserver la cohérence du `Sensor3D` dans les différents terminaux.

Comme nous l'avons vu sur la figure 3.4 (page 84), la proposition multi-utilisateurs de Blaxxun permet de gérer les conflits d'accès, mais cela nécessite de bonnes connaissances en programmation de scripts. Au contraire, afin de faciliter le développement de contenus multi-utilisateurs, notre approche reste purement descriptive et la gestion du contrôle d'accès est cachée du concepteur d'applications qui a uniquement besoin de renseigner les champs des différents `SharedNodes` et `SharedSets`.

6.2.4 Implémentation

Il est à noter que le mécanisme de substitution n'est pas réalisable avec un navigateur VRML classique. Le navigateur VRML 97 autour duquel est construite la plate-forme Spin-3D a été implémenté de façon à supporter cette fonctionnalité. Lorsque le navigateur VRML 97 étendu détecte qu'un champ est partagé, il est capable de remplacer dynamiquement l'objet représentant le champ par un autre objet similaire, et possédant la capacité de communiquer avec ses copies distantes. Le comportement de cet objet représentant un champ partagé est configuré dynamiquement en fonction des informations contenues dans le `SharedNode` associé. En VRML 97, la modification d'un champ est réalisée en appelant la méthode `setValue()` sur l'objet représentant le champ. Un objet représentant un champ partagé a une méthode `setValue()` dont le comportement est modifié dynamiquement en fonction du canal de communication spécifié dans le nœud de description du partage.

6.3 Bilan

Dans ce chapitre, nous avons présenté un nouveau paradigme permettant le partage de données. Le paradigme de substitution comme nous l'appelons, permet de résoudre les problèmes liés à l'utilisation du paradigme d'insertion (cohérence des `Sensors` et gestion du partage lors de manipulations depuis des applications externes). Nous avons également présenté la mise en œuvre du paradigme de substitution en utilisant le langage VRML 97. Notre extension multi-utilisateurs VRML 97 basée sur le paradigme de substitution, est conçue de façon à :

- respecter la syntaxe VRML 97 afin de ne pas dénaturer les contenus VRML 97 existants. Ainsi, des navigateurs VRML 97 standards sont capables de charger et d'afficher correctement des contenus multi-utilisateurs conçus à partir de notre extension (bien entendu sans aucun support du multi-utilisateurs),
- faciliter la conception de contenus multi-utilisateurs à partir de contenus existants,
- créer une hiérarchie de données partagées indépendamment de la description géométrique,
- supporter les différents canaux de communication proposés par notre plate-forme de communication.

Chapitre 7

Mécanisme de communication externe

Sommaire

La plate-forme de communication telle que nous l'avons présentée précédemment permet aux différents terminaux de s'échanger des informations afin de maintenir cohérentes les données partagées et de gérer la session de travail coopératif. Une telle plate-forme permet de réaliser des communications que nous appelons "internes". Dans cette section, nous nous intéressons à la définition d'une plate-forme de communication externe permettant la connexion d'applications existantes sur le terminal Spin-3D.

Les utilisateurs ont besoin d'outils pour pouvoir travailler dans l'interface et produire de nouveaux documents. Notre plate-forme fournit un mécanisme permettant de réaliser des applications dites "natives", intégrées directement dans l'interface de travail: c'est ce que nous appelons des applications "internes" . Un mécanisme de *plug-in* permet d'étendre les fonctionnalités de notre terminal. Des *plug-in* de visualisation permettent au terminal d'afficher des objets autres que des objets au format VRML 97; des *plug-in* de comportement agissent comme des contrôleurs d'interface. Cependant, ce mécanisme ne permet de réaliser que des applications coopératives "simples" (comme par exemple, un jeu de cartes). Nous voulons offrir aux concepteurs la possibilité de développer des applications plus complexes (comme par exemple, un outil de CAO 3D collaboratif). Cependant, il y est impossible de vouloir intégrer toutes les applications existantes dans l'interface de travail. Cela reviendrait à voir notre plate-forme comme un nouveau système d'exploitation. Dans le cadre d'un outil de CAO 3D multi-utilisateurs, nous n'avons pas pour but d'intégrer un moteur de CAO 3D dans notre interface; de même, nous ne voulons pas que le moteur de CAO 3D intègre directement le terminal de travail collaboratif. D'un point de vue IHM 3D, les applications doivent être adaptées à un fonctionnement en 3D, ce qui n'est pas envisageable dans tous les cas: du fait de l'utilisation de la troisième dimension, les différents mécanismes de présentation et d'interaction des applications actuelles sont complètement à revoir pour pouvoir être intégrées directement dans notre interface 3D.

Nous voulons permettre à des applications qui ne peuvent pas être directement incluses dans l'interface de travail 3D, de pouvoir interagir avec le terminal de travail coopératif: nous qualifions de telles applications d'"externes". Pour cela, nous proposons une interface de communication entre les applications existantes et notre terminal de travail [LDPDG+03]. Les utilisateurs conservent leurs applications existantes (leurs mécanismes de présentation et d'interaction) pour produire leurs documents et utilisent notre terminal de travail pour coopérer. On peut éventuellement imaginer voir travailler chaque utilisateur sur deux postes: un poste exécutant le terminal de travail, un autre poste exécutant l'application classique. Une interface de programmation doit permettre d'établir directement un canal de communication entre l'application externe et notre terminal.

Dans ce chapitre, nous étudions les mécanismes mis en œuvre pour la réalisation de l'interface

de communication externe, basée sur les spécifications de l'EAI VRML 97, ainsi que son fonctionnement. Ensuite, nous identifions différents services dont ont besoin les applications externes et présentons un ensemble d'extensions permettant d'offrir ces différents services. Puis finalement, nous présentons l'implémentation de notre interface de communication externe.

7.1 Définition de l'interface de programmation

Dans notre interface de travail, les différents objets 3D sont décrits en utilisant le langage VRML 97 : l'interaction et le partage sont décrits dans le même fichier que la géométrie 3D. Comme nous l'avons vu dans l'état de l'art, il existe une norme VRML 97 définissant une interface de programmation de haut niveau, l'EAI (External Authoring Interface), et permettant à des applications externes (ie. s'exécutant en dehors du navigateur VRML 97) de manipuler la scène contenue dans le navigateur. La spécification de l'EAI regroupe un ensemble de services permettant la manipulation des nœuds contenus dans le graphe de scène. L'EAI est implémentée par la majorité des navigateurs VRML 97 disponibles sur le marché. Cependant, ils n'offrent pas toutes les fonctionnalités que l'EAI est capable de supporter. Théoriquement, n'importe quel langage de programmation peut être utilisé pour développer une application externe utilisant l'EAI. Néanmoins, dans la pratique, les différentes implémentations de l'EAI n'offrent qu'une interface Java permettant uniquement le développement d'applets capables de communiquer avec le contenu du navigateur VRML 97. Les implémentations disponibles dans les navigateurs VRML 97 classiques permettent uniquement la communication entre une applet et une scène VRML 97 contenues dans la même page HTML, et par conséquent, l'applet et le navigateur VRML 97 s'exécutent sur la même machine, dans le contexte du processus du navigateur HTML. En effet, pour réaliser cela, ils utilisent uniquement les fonctionnalités (*LiveConnect* et *ActiveX/COM*) que proposent les navigateurs Internet (cf. annexe A.4, page 228) permettant de faire communiquer différents éléments s'exécutant au sein d'un même navigateur Internet.

Pour concevoir notre interface de communication entre les applications externes et notre terminal de travail, nous proposons de réunir deux standards : la spécification de l'EAI et la norme CORBA. C'est pourquoi, nous avons nommé notre interface de communication, DAI (*Distributed Authoring Interface*). L'utilisation de CORBA nous offre une infrastructure générique et distribuée pour réaliser les communications entre une application externe et le terminal : les applications externes interagissent de façon transparente avec des objets distants contenus dans notre terminal de travail. De plus, inhérent à l'utilisation de l'intergiciel CORBA, nous avons un support multi-plate-formes et multi-langages pour le développement des applications externes.

7.1.1 Architecture

Pour concevoir notre interface de programmation, nous utilisons la dernière spécification de l'EAI. En effet, elle contient une description IDL-OMG des différents services proposés (les interfaces `Browser`, `Node`, `Field`, ainsi que les différents observateurs d'évènements). À partir de cette description, nous sommes capables de générer les souches dans différents langages (C++, Java, etc.) pour la partie cliente et la partie serveur. Le navigateur VRML 97 étendu de notre terminal de travail joue le rôle du serveur, ie. la partie hébergeant les objets implémentant les squelettes générés à partir de la description IDL-OMG. L'application externe joue le rôle du client, elle utilise les souches clientes générées à partir de la description IDL-OMG qui permettent de masquer la distribution des objets manipulés. L'architecture générale de la DAI est présentée sur la figure 7.1.

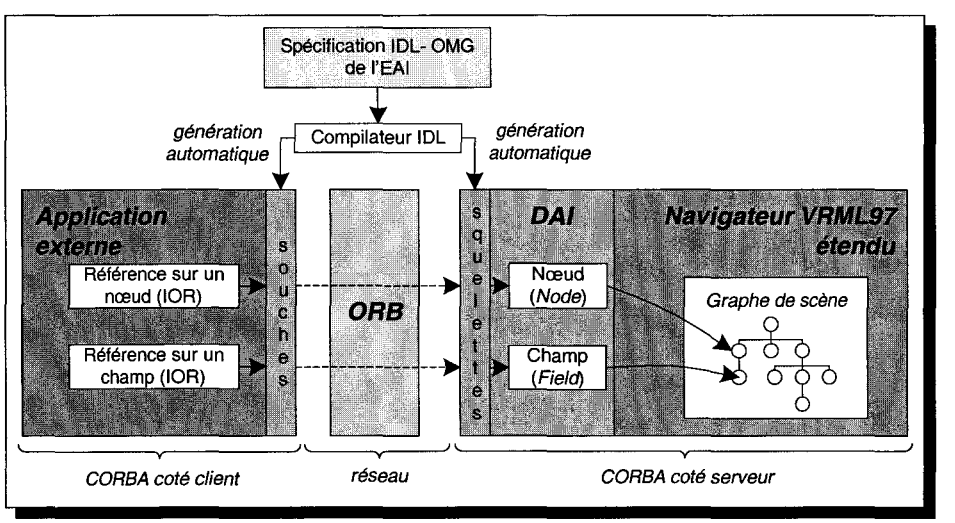


FIG. 7.1 – *Architecture de l'interface de manipulation externe (DAI) de notre terminal de travail*

7.1.2 Principe de fonctionnement

Une première solution, la plus naïve, est de connecter tous les noeuds et champs contenus dans le graphe de scène au bus CORBA. Une telle approche n'est pas envisageable : la connexion d'un grand nombre d'objets sur le bus CORBA risquent d'avoir un impact négatif sur ses performances. De plus, seulement quelques nœuds et champs sont susceptibles d'être utilisés par les applications externes.

Une solution moins coûteuse est de connecter uniquement les nœuds et champs nécessaires aux applications externes. Pour cela, l'interface de communication externe est capable de créer,

à la volée, des objets que nous appelons "passerelles" pour les nœuds et champs contenus dans le graphe de scène du navigateur VRML 97. Une "passerelle" est un objet qui permet de faire le lien entre la DAI et les objets du graphe de scène. L'interface d'une passerelle de nœud ou de champ est celle spécifiée dans la norme de l'EAI (respectivement, l'interface `Node` et `Field`).

Lorsqu'une application externe demande un objet nœud ou champ, elle obtient une référence (ie. une IOR en utilisant la terminologie CORBA) sur un objet passerelle de nœud ou de champ. Ensuite, à partir de la référence obtenue, l'application est capable d'invoquer des méthodes sur ces objets de façon transparente (comme si l'objet était présent dans son espace mémoire). En interne, une passerelle possède un pointeur vers un élément du graphe de scène (nœud ou champ). Lorsque la passerelle reçoit une invocation de méthode, elle délègue l'appel à l'objet qu'elle représente, et renvoie la réponse à l'application externe.

7.1.3 Utilisation des observateurs d'évènements

Dans le fonctionnement classique de l'EAI VRML 97, l'application externe est capable de placer des observateurs sur des champs afin d'être notifiée de leurs modifications. Nous proposons également un tel mécanisme : l'application externe crée un observateur de champ (`VrmlEventListener`), le connecte à l'ORB, et ensuite donne la référence (IOR) de l'observateur créé au terminal de travail Spin-3D en déclarant à un champ que l'application externe souhaite l'observer. Dès qu'un champ est modifié, le gestionnaire d'évènements de notre terminal VRML 97 étendu est alors en charge de prévenir tous les observateurs enregistrés auprès de ce champ : pour cela, le gestionnaire d'évènements utilise une invocation de méthode sur la référence de chaque observateur enregistré.

7.2 Définition d'extensions de l'EAI

Une application externe peut avoir besoin d'effectuer des opérations complexes sur le contenu de l'interface de travail. En effet, une application externe doit être capable de traverser le graphe de scène VRML 97 et doit être capable de récupérer une référence sur n'importe quel nœud contenu dans le graphe de scène afin de pouvoir le manipuler, ie. modifier les valeurs de ses champs. Par exemple, on peut imaginer qu'une application externe souhaite modifier la couleur d'un objet contenu dans l'interface : pour cela, elle doit être capable de récupérer une référence sur tous les nœuds `Material` correspondant à cet objet et modifier sur chacun les champs correspondant à la couleur. On peut également imaginer qu'une application externe ait besoin de

parcourir un morceau de la hiérarchie VRML 97 pour récupérer un objet contenu dans l'interface et le sauvegarder dans un fichier.

L'EAI telle qu'elle a été définie ne permet pas d'offrir de tels services. En effet, l'EAI n'autorise pas d'obtenir une référence sur n'importe quel nœud contenu dans le graphe : uniquement ceux possédant un `DEF` sont accessibles via l'EAI.

C'est pourquoi, nous proposons plusieurs extensions à l'EAI classique permettant d'offrir les services dont a besoin une application externe afin d'interagir avec notre terminal de travail. Aucune modification des interfaces existantes n'a été effectué, nos propositions sont des extensions des interfaces déjà existantes grâce au mécanisme d'héritage que propose le langage IDL-OMG.

7.2.1 La notion de `NodePath`

Comme nous l'avons déjà dit, l'EAI classique ne permet d'accéder qu'aux nœuds du graphe de scène sur lesquels le concepteur a placé un `DEF`. À partir du moment où un noeud possède un `DEF`, il est accessible via l'EAI en utilisant la méthode `getNode()` de l'interface `Browser`. Nous considérons que cela constitue une limitation et qu'une application externe doit avoir accès à tous les nœuds contenus dans le graphe de scène, pas uniquement les nœuds possédant un `DEF`.

Pour contourner cette limitation, nous proposons un mécanisme que nous appelons *NodePath*, inspiré de *XPath* [W3C99] de la norme XML. Un *NodePath* définit un chemin dans l'arborescence d'un objet VRML 97 permettant de désigner un nœud en particulier. Chaque nœud intermédiaire du graphe de scène qui permet de mener à un nœud précis apparaît dans le *NodePath* soit en utilisant son nom standard (ie. le type du nœud), soit en utilisant son `DEF` (si il en possède un) :

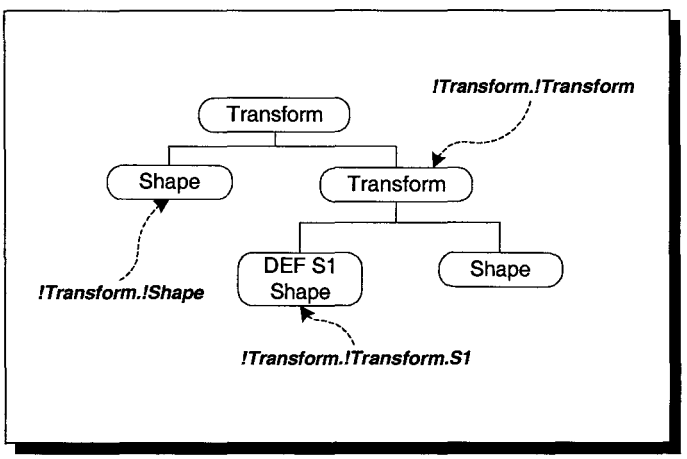
- si c'est le nom standard du nœud qui est utilisé, il apparaîtra sous la forme "`!<NomDuTypeDeNoeud>`" dans le *NodePath*. C'est le nom utilisé par défaut, un point d'exclamation ("!") suivi du nom du type de nœud. Par exemple, un nœud `Transform` apparaît sous la forme "`!Transform`" dans le NodePath,
- si le nœud possède un `DEF`, ce dernier peut être utilisé : il apparaîtra alors sous son nom de `DEF` ("`<NomDuDEF>`") dans le *NodePath*. Par exemple, un nœud Transform possédant un `DEF` (`DEF MonTransform Transform {...}`) apparaît sous la forme "`MonTransform`" dans le *NodePath*.

Le caractère "`.`" est utilisé comme séparateur de noms. Comme les chemins de fichiers utilisés dans les lignes de commandes des systèmes d'exploitation, nous autorisons l'utilisation de méta-

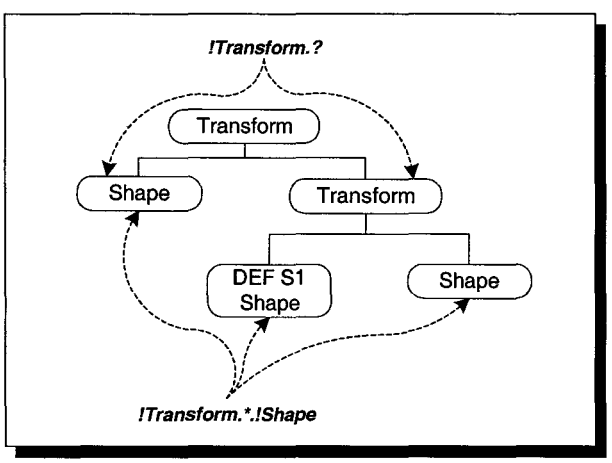
caractères afin de remplacer un ou plusieurs noeuds inconnus :

- le méta-caractère désigné par le symbole "?" remplace un et un seul nœud quelconque,
- le méta-caractère désigné par l'astérisque "*" remplace une séquence de nœuds de taille quelconques (de 0 à n).

La figure 7.2 présente plusieurs exemples d'utilisation de la notion de *NodePath*.



(a) exemple de NodePath complet



(b) exemple de NodePath avec méta-caractères

FIG. 7.2 – *Exemples d'utilisation de NodePath*

Il est à noter qu'un même *NodePath* peut désigner plusieurs nœuds du graphe de scène. Comme cela est possible avec XPath, nous autorisons la spécification d'index afin de désigner un nœud en particulier. L'index est placé entre crochets juste après le nom du nœud (il est à noter que c'est un parcours réalisé en profondeur d'abord). La figure 7.3 présente un exemple d'utilisation d'index dans un *NodePath*.

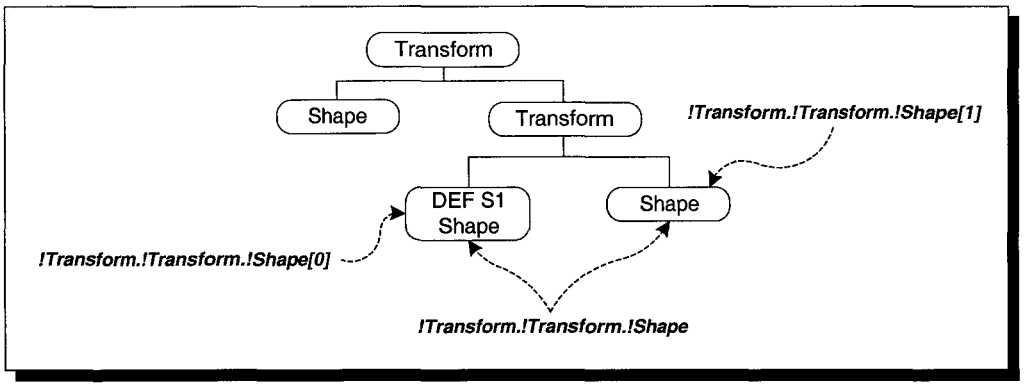


FIG. 7.3 – *Exemples d'utilisation d'index dans un NodePath*

Avec le terminal Spin-3D, chaque objet de l'interface 3D possède un identifiant unique. De plus, chaque objet 3D est placé dans un *widget*, comme par exemple la "table" ou le "bandeau" (cf. 4.1.2, page 116). Il est possible de faire figurer ces différentes informations dans un *NodePath* afin d'effectuer une recherche sur un objet en particulier. Par exemple, si un objet, possédant l'identifiant 1000, est présent sur la table, nous préfixons le *NodePath* avec `"Table.1000."` afin de désigner cet objet en particulier. Avec ce préfixe, il est possible de faire une recherche dans la géométrie d'un objet en particulier. Il est également possible d'utiliser les méta-caractères `"*"` et `"?"` dans ce préfixe, afin de remplacer le nom du *widget* et l'identifiant de l'objet.

7.2.2 L'interface `NodeSearcher`

Comme nous l'avons vu, l'EAI classique permet d'obtenir des références uniquement sur des noeuds possédant un `DEF`. Le concept de *NodePath* tel que nous venons de le présenter nous permet de supprimer une telle limitation. Nous avons également défini l'interface `NodeSearcher` permettant de retrouver des nœuds contenus dans le graphe de scène. Une instance d'une telle interface permet de retrouver tous les nœuds désignés par un *NodePath* donné : c'est une alternative à la méthode `getNode()` fournie par l'interface du `Browser` de l'EAI standard, une méthode ne permettant d'obtenir qu'un seul nœud à la fois.

Comme nous l'avons mentionné précédemment, un *NodePath* peut pointer sur plusieurs nœuds à la fois. C'est pourquoi, un objet `NodeSearcher` doit gérer une liste de nœuds, chacun d'entre eux étant pointé par le *NodePath* spécifié. Une fois la recherche effectuée (une recherche en profondeur d'abord est réalisée dans le graphe de scène) et les nœuds correspondant trouvés, il n'est pas envisageable de directement tous les connecter au bus CORBA. Seuls quelques noeuds sont susceptibles d'être utilisés par l'application externe, de plus l'application est susceptible de les utiliser séquentiellement, les uns après les autres. De plus, un trop grand nombre d'ob-

jets connectés au bus CORBA risque de réduire son efficacité. Par exemple, on peut imaginer qu'une application externe effectue une recherche avec le *NodePath* suivant, `"*"` ; dans une telle situation, le résultat de la recherche est tous les noeuds contenus dans le graphe de scène.

Nous proposons que notre interface `NodeSearcher` gère une liste contenant les nœuds résultats de la recherche, avec un curseur pointant à un instant donné sur un élément de la liste. L'application externe peut manipuler le curseur pour le faire pointer sur un autre élément de la liste. Elle doit pouvoir également demander différentes informations sur l'élément de la liste qui est pointé par le curseur afin de déterminer si elle est intéressée par ce nœud. Elle peut alors demander à l'objet `NodeSearcher` de connecter l'élément de la liste pointé par le curseur au bus CORBA afin d'obtenir une référence sur ce nœud pour pouvoir le manipuler.

La figure 7.4 présente la définition IDL-OMG de l'interface `NodeSearcher`. Nous allons maintenant présenter les différentes méthodes que propose cette nouvelle interface :

- les méthodes `previous()`, `next()` et `jump(i)` permettent de déplacer le curseur dans la liste. Juste après avoir effectué une recherche, le curseur est positionné sur le premier élément de la liste (la position du curseur étant alors égale à 0),
- la méthode `getSize()` permet de connaître le nombre d'éléments présents dans la liste résultat,
- les méthodes `getName()` et `getType()` permettent d'obtenir respectivement le nom et le type du noeud qui est pointé par le curseur de liste. Elles ont exactement le même comportement que les méthodes `getName()` et `getType()` de l'interface `Node` présente dans la spécification originelle de l'EAI. Cela permet d'obtenir des informations sur le nœud pointé par le curseur sans pour autant avoir à le connecter au bus CORBA quand cela n'est pas nécessaire,
- la méthode `getNodePath()` permet de récupérer le *NodePath* complet du nœud de la liste pointé par le curseur,
- la méthode `getNode()` permet de récupérer une référence sur le nœud pointé par le curseur de liste. Le nœud est alors connecté au bus CORBA. L'application externe est alors seule responsable pour détruire la référence créée en appelant la méthode `dispose()` de l'interface `Node`,
- la méthode `getNodeFromIndex(i)` permet de connecter au bus CORBA et récupérer une référence sur le $i^{ième}$ nœud contenu dans la liste. Dans ce cas, le curseur n'est pas déplacé,
- la méthode `searchNode()` permet d'effectuer une nouvelle recherche, ainsi un même objet `NodeSearcher` peut être utilisé à plusieurs reprises par l'application externe évitant ainsi de créer un nouvel objet `NodeSearcher` pour chacune des recherches. La liste courante contenue dans l'objet `NodeSearcher` est alors effacée et remplie avec le résultat de la nouvelle

recherche. Cependant, les références sur nœuds obtenues par l'application externe restent quant à elles toujours valides, l'application ayant seule la responsabilité de les détruire,

- la méthode `dispose()` permet de signaler au noyau de notre plate-forme que l'application n'est plus intéressée par l'objet `NodeSearcher`, et que, par conséquent il est libre de le détruire. La référence de l'objet `NodeSearcher` que possède l'application externe n'est alors plus valide.

```
module vrml {
 module eai {
  interface NodeSearcher {
   void   searchNode(in string npath);
   Node   getNode()
          raises (InvalidNodeException);
   Node   getNodeFromIndex(in long index)
          raises (InvalidNodeException,
                  ArrayIndexOutOfBoundsException);
   string getType()
          raises (InvalidNodeException);
   string getName()
          raises (InvalidNodeException);
   string getNodePath()
          raises (InvalidNodeException);
   long   getSize();
   void   previous()
          raises (InvalidNodeException);
   void   next()
          raises (InvalidNodeException);
   void   jump(in long index)
          raises (InvalidNodeException, ArrayIndexOutOfBoundsException);
   void dispose();
  }; // End of interface NodeSearcher
 }; // End of module eai
}; // End of module vrml
```

FIG. 7.4 – *Définition IDL-OMG de l'interface* `NodeSearcher`

Dans le but de supporter l'interface `NodeSearcher`, nous étendons, en utilisant la notion d'héritage du langage IDL-OMG, l'interface `Browser` déjà existante. Comme le montre la figure 7.5, l'interface `BrowserExt` étend l'interface `Browser` en ajoutant le support de l'interface `NodeSearcher`. Elle propose une méthode `getNodeSearcher()` permettant d'obtenir une référence sur un objet `NodeSearcher` qui peut ensuite être utilisé pour effectuer une recherche à partir d'un `NodePath`.

7.2.3 L'interface `NodeProxy`

Nous souhaitons également permettre aux applications externes de traverser la hiérarchie VRML 97 d'un objet de l'interface de travail. Les services offerts par la spécification classique de l'EAI ne permettent pas de réaliser facilement la traversée complète du graphe de scène. En effet, on pourrait imaginer utiliser la méthode `getNode()` de l'interface `Browser`, mais cela nécessite que tous les noeuds contenus dans le graphe de scène possèdent un `DEF`, et que la

```
module vrml {
 module eai {
  interface Browser {
   /* Classical methods to access services provided      */
   /* by the browser, such as: name, version, nodes, etc */

   Node getNode(in string name) raises (...);

   /* (...) */
  }

  interface BrowserExt : Browser {
   NodeSearcher getNodeSearcher()
                raises (ConnectionException);
  }; // End of interface BrowserExt
 }; // End of module eai
}; // End of module vrml
```

FIG. 7.5 – *Définition IDL-OMG de l'interface* `BrowserExt`

structure hiérarchique de graphe de scène soit connue à l'avance. Une telle méthode n'est pas envisageable dans le contexte distribué dans lequel nous sommes. En effet, le temps de réponse de l'application externe dépend, en partie, du temps de réponse du bus CORBA : le temps de création et de connexion d'un nouvelle passerelle n'est pas négligeable. Cela a pour conséquence de réduire l'interactivité de l'application externe.

Une autre approche serait d'utiliser un objet `NodeSearcher` comme nous l'avons défini précédemment : l'application externe peut effectuer une recherche en utilisant le NodePath `"*"`, dans ce cas la liste résultante de la recherche contenue dans l'objet `NodeSearcher` contiendra tous les nœuds du graphe de scène VRML 97. Cependant, cette solution a deux inconvénients :

- d'une part, la structure hiérarchique du graphe de scène est perdue,
- d'autre part, cette méthode impose à l'application externe de demander explicitement la connexion (en utilisant la méthode `getNode()` de l'interface `NodeSearcher`) et la déconnexion (en utilisant la méthode `dispose()` de l'interface `Node`) de chacun des nœuds. Dans un contexte distribué comme le notre où nous souhaitons permettre à l'application externe de s'exécuter sur une autre machine que celle exécutant le terminal de travail, il est important de limiter les temps de latence afin que l'utilisation de l'application externe reste interactive. Le temps de réponse lié aux communications réseau reste incompressible, il faut alors limiter le temps de traitement.

Nous préférons proposer une solution plus efficace : elle consiste à réutiliser une passerelle déjà existante et de modifier son état interne afin de la faire pointer sur un autre noeud du graphe de scène VRML 97 permettant ainsi de naviguer dans la hiérarchie. Un objet `Node` offrant de telles propriétés est ce que nous appelons un `NodeProxy`. Un tel mécanisme est inspiré des patrons de conception (*design patterns*) *Traversable* ou *Visitor* [GHJV95]. Un objet `NodeProxy` pointe à un

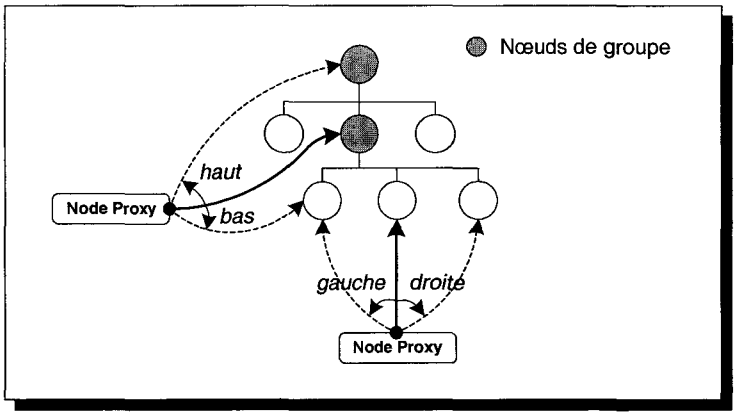


FIG. 7.6 – *Navigation d'un objet* `NodeProxy` *dans la hiérarchie VRML 97*

instant sur un nœud du graphe de scène, et contrairement à un objet `Node` de la spécification classique de l'EAI qui pointe de façon statique sur un nœud de la hiérarchie, il est possible de modifier dynamiquement le nœud pointé par un objet `NodeProxy`. La figure 7.6 montre que l'on peut déplacer un objet `NodeProxy` de haut en bas (ou de bas en haut) dans la hiérarchie VRML 97, ainsi que de droite à gauche (ou de gauche à droite) parmi les fils d'un nœud de groupe.

Comme le montre la figure 7.7, l'interface `NodeProxy` hérite de l'interface `Node` afin d'offrir les mêmes services que l'interface `Node` (comme les méthodes pour accéder aux champs, `getEventIn()` et `getEventOut()`, etc). L'interface `NodeProxy` propose également des méthodes pour naviguer dans la hiérarchie VRML 97. La propriété d'héritage nous permet de réutiliser les méthodes existantes : toute méthode retournant déjà un objet `Node` peut être utilisée pour retourner un objet `NodeProxy`. Ainsi, la méthode `getNode()` de l'interface `Browser` peut être utilisée pour obtenir la référence sur un objet `NodeProxy`, cela dépend uniquement de l'implémentation, de même pour les méthodes `getNode()` et `getNodeFromIndex()` de l'interface `NodeSearcher`.

La figure 7.8 présente le comportement des différentes méthodes de navigation que proposent l'interface `NodeProxy` sur son état interne. Il est à noter que les méthodes `getParent()`, `getChild(i)`, `getNext()` et `getPrevious()` ne modifient pas l'état interne de l'objet `NodeProxy`. En effet, ces quatre méthodes permettent à l'application externe d'obtenir, de façon explicite, une nouvelle passerelle pointant sur, respectivement, le parent du nœud N (en supposant que l'objet `NodeProxy` pointe sur le nœud N), le $i^{ième}$ fils du nœud N, le nœud "frère" à droite de N et le nœud "frère" à gauche de N.

```
module vrml {
 module eai {
  interface Node {
   /* Classical methods to access node members    */
   /* (type, name, fields), and release nterest. */

   eai::field::EventIn  getEventIn(in string name)
                        raises (...);
   eai::field::Eventout getEventOut(in string name)
                        raises (...);

   /* (...) */
  }

  interface NodeProxy : Node {
   void parent()
        raises (InvalidNodeException);
   Node getParent()
        raises (InvalidNodeException);
   void child(in long index)
        raises (InvalidNodeException,
                ArrayIndexOutOfBoundsException);
   Node getChild(in long index)
        raises (InvalidNodeException,
                ArrayIndexOutOfBoundsException);
   void next()
        raises (InvalidNodeException);
   Node getNext()
        raises (InvalidNodeException);
   void previous()
        raises (InvalidNodeException);
   Node getPrevious()
        raises (InvalidNodeException);
  }; // End of interface NodeProxy
 }; // End of module eai
}; // End of module vrml
```

FIG. 7.7 – *Définition IDL-OMG de l'interface* **`NodeProxy`**

Nom de la méthode	État interne (avant)	État interne (après)
parent()	le nœud N	le parent du nœud N
child(i)	le nœud N	le $i^{ième}$ nœud fils de N
next()	le nœud N	le nœud "frère" à droite de N
previous()	le nœud N	le nœud "frère" à gauche de N

FIG. 7.8 – *Comportement des différentes méthodes de l'interface* **`NodeProxy`** *permettant la navigation dans la hiérarchie VRML 97*

7.2.4 Gestion des champs partagés

Nous avons défini dans notre plate-forme le gestionnaire d'accès concurrents permettant d'éviter que plusieurs utilisateurs ne modifient en même temps, depuis l'interface de travail, une même donnée partagée. Dans l'interface de travail, les modification de données partagées se font à l'aide de `Sensor3D` placés sur les objets que l'on autorise à manipuler. Le contrôle d'accès (ie. l'acquisition de jeton) est réalisé au moment de la sélection et de l'activation du `Sensor3D` choisi en utilisant le champ `lockName` défini dans le nœud `Sensor3D` (cf. 5.2.2.2, page 161).

Or, comme nous l'avons dit, une application externe accède directement aux champs des noeuds. C'est pourquoi, avant de réaliser une modification sur un champ, l'application externe doit s'assurer que personne n'est déjà en train de manipuler le champ qu'elle souhaite modifier. Nous devons offrir aux applications externes la possibilité de demander à notre terminal de travail l'autorisation de modifier un champ quand elles en ont besoin.

Pour cela, nous avons étendu l'interface de chaque type de base VRML 97 définie dans la norme de l'EAI (`EventInSFBool`, `EventInSFColor`, etc.) pour offrir aux applications externes la possibilité de verrouiller/déverrouiller le jeton associé à une action sur un champ partagé (cf. figure 7.9). La méthode `lock()` permet à l'application externe de demander à verrouiller le jeton, ie. seule l'application externe aura le droit de modifier le champ partagé. Pour cela, pour un champ partagé, nous utilisons le champs `lockName` contenu dans le noeud de partage (`SharedNode`). La méthode `lock()` s'exécute alors comme présentée sur la figure 7.10. La méthode `unlock()` permet de déverrouiller le jeton.

```
module vrml {
 module eai {
  interface AccessControlField {
   boolean lock();
   void unlock();
  }; // End of interface AccessControlField

  interface EventInSFBoolExt : EventInSFBool, AccessControlField {}
  interface EventInSFColorExt : EventInSFColor, AccessControlField {}
  ...

 }; // End of module eai
}; // End of module vrml
```

FIG. 7.9 – *Extension de l'interface de chaque type de base VRML 97 pour gérer les accès concurrents*

Cela nécessite un travail de la part du concepteur d'objets 3D. Il doit respecter certaines règles lors de la conception multi-utilisateurs : il doit affecter au champ `lockName` d'un nœud de partage la même valeur que celle contenue dans le champ `lockName` du `Sensor3D` qui lui est relié par le mécanisme de `ROUTE`. Ainsi, lorsqu'un utilisateur manipule une donnée partagée depuis

```
Fonction booléen lock()
  Si le champ est partagé Alors
    jeton = la valeur du champ lockName du SharedNode correspondant au champ partagé
    Si acquérir le jeton est vrai Alors
      retourner vrai
    Sinon
      retourner faux
    FinSi
  Sinon
    retourner vrai
  FinSi
```

FIG. 7.10 – *Algorithme utilisé pour la méthode* `lock()`

l'interface de travail via le `Sensor3D`, aucune autre application externe, ni aucun autre utilisateur depuis l'interface de son terminal, ne peuvent modifier la même donnée partagée: le jeton est déjà verrouillé, et l'application ou le terminal verront sa demande de jeton rejetée puisque elle cherchera à acquérir le même jeton. Cette solution doit être transitoire: dans la mesure où nous voulons faciliter la conception des objets 3D utilisés par notre interface de travail, nous devons à terme fournir un algorithme automatique permettant de remonter les `ROUTES` VRML 97. Pour un champ donné, l'algorithme serait capable de retrouver le `Sensor3D` qui lui est relié, et par conséquent, le champ `lockName` contenu dans le `Sensor3D`.

7.3 Implémentation

Dans cette section, nous présentons différents aspects de notre implémentation de la DAI. Tout comme le navigateur VRML 97 étendu, la DAI a été implémentée en utilisant le langage C++. Comme pour l'implémentation de notre couche de communication RMIOP, nous utilisons le bus CORBA *Orbacus* développé par IONA pour implémenter la DAI[47].

Les objets gérés par la DAI sont connectés sur un bus indépendant du bus utilisé par le mécanisme de communication interne: les propriétés de ces deux bus sont différentes, par exemple, pour le mécanisme de communication interne, on a besoin de spécifier, pour chaque objet, l'identifiant du groupe auquel l'objet appartient.

47. Nous utilisons la version 4.0.5 du bus CORBA Orbacus.

7.3.1 Développement d'applications externes

7.3.1.1 Connexion au terminal Spin-3D

Au démarrage du terminal de travail Spin-3D, un objet `BrowserExt` est créé et est connecté au bus. Comme cela est spécifié dans la norme de l'EAI, l'application doit tout d'abord obtenir une référence sur l'objet `Browser` pour pouvoir effectuer des manipulations sur le graphe de scène (récupérer des références sur des nœuds, des champs, etc). Dans le cadre de la DAI, nous proposons d'utiliser la méthode `getBrowser()` définie dans la spécification standard de l'EAI : cette méthode permet d'obtenir une référence sur l'objet `Browser`. Ainsi, nous conservons la compatibilité avec le schéma de fonctionnement standard de l'EAI. En utilisant la propriété d'héritage, la méthode `getBrowser()` peut indifféremment retourner un objet `Browser` ou un objet `BrowserExt` : cela dépend uniquement de l'implémentation de la méthode `getBrowser()`. En utilisant les fonctionnalités offertes par CORBA, il existe plusieurs méthodes pour implémenter la méthode `getBrowser()` :

- une première solution est d'utiliser une référence IOR écrite sous la forme d'une chaîne de caractères. Le terminal de travail Spin-3D doit pour cela publier la référence IOR de l'objet `Browser` (ou de l'objet `BrowserExt`, pour bénéficier de nos extensions) sur un serveur HTTP ou un serveur FTP. Ainsi, les applications externes sont capables de récupérer la référence écrite sous la forme d'une chaîne de caractères et de l'utiliser pour appeler des méthodes sur l'objet `Browser` (ou `BrowserExt`),
- une deuxième solution est d'utiliser le mécanisme d'URL fourni par la norme du service de nommage [OMG01c] (en anglais, *Interoperable Naming Service*, INS). Les références écrites sous la forme d'une URL sont plus lisibles que les références écrites sous la forme de chaîne de caractères. Une URL contient plusieurs informations (le protocole, le nom de la machine serveur, le numéro du port, la clé de l'objet qui est un identifiant unique et connu, etc.) permettant au client de reconstruire la référence de l'objet et qu'il puisse l'utiliser pour appeler des méthodes. Avec cette méthode, l'objet `Browser` doit posséder une clé unique et connue de la part du client. Le client doit uniquement obtenir le nom de la machine et le numéro du port du terminal à contacter,
- la troisième méthode est d'utiliser le service de nommage défini dans la norme CORBA. Le terminal de travail doit enregistrer auprès du service de nommage la référence de l'objet `Browser` (ou l'objet `BrowserExt`) en spécifiant un identifiant symbolique. Les applications externes utilisent cet identifiant pour retrouver la référence de l'objet.

Dans le terminal de travail Spin-3D, les deux dernières méthodes ont été implémentées. Les identifiants utilisés dans l'URL et dans le services de nommage doivent être connus et fixes.

7.3.1.2 Implémentation d'une application externe

Bénéficiant des avantages offerts par l'intergiciel CORBA, les applications externes peuvent être développées dans n'importe quel langage de programmation (il suffit juste qu'une translation du langage IDL-OMG vers le langage de programmation cible soit spécifiée). L'OMG a déjà spécifié des translations du langage IDL-OMG vers de nombreux langages de programmation comme C, C++, Java, etc.

Il est à noter que le développement d'applications externes nécessite aux concepteurs de respecter les règles de programmation classiques liées à l'utilisation de CORBA. Des opérations complexes sont à réaliser par le concepteur, par exemple les opérations d'initialisation du bus. L'OMG propose d'utiliser le langage de script IDLScript [OMG01b] (cf. annexe B, page 234) pour faciliter le développement d'applications CORBA : toutes les opérations complexes (comme l'initialisation de l'ORB ou les opérations de *cast*) sont effectuées par l'interpréteur de script. Pour le développement d'applications externes, nous bénéficions nous aussi des avantages offerts par le langage IDLScript.

L'annexe G (page 260) présente une même application développée dans différents langage : C++, Java et IDLScript.

7.3.2 Aspects sécurité

N'importe quelle application externe est capable d'obtenir la référence de l'objet `Browser` et de lui faire réaliser des activités néfastes. Nous devons être capable de contrôler les connexions d'applications externes sur notre terminal de travail : seules les applications autorisées seront capables de manipuler le contenu de l'interface de travail.

Avec un ORB standard, les communications entre les applications externes et le navigateur VRML 97 étendu se font en utilisant le protocole IIOP (qui utilise TCP/IP). Une telle solution n'offre aucune garantie pour la sécurisation des communications : en effet, du fait du manque de sécurisation du protocole IP (IPv4), ni le protocole IIOP, ni le protocole TCP/IP ne sont des

protocoles sécurisés et n'offrent de mécanismes d'authentification[48].

L'idée est donc de remplacer le protocole non sécurisé IIOP par un protocole sécurisé. Comme nous l'avons vu précédemment, au travers de l'interface de programmation des OCI (cf. annexe E, page 250), le bus CORBA *Orbacus* nous permet d'utiliser d'autres protocoles que IIOP. IONA propose un nouveau protocole, *FreeSSL Inter-ORB Protocol* (FSSLIOP) [Iona], capable de remplacer le protocole IIOP d'*Orbacus* et permettant de sécuriser les échanges entre le client et le serveur. Le protocole FSSLIOP, s'intègrant à Orbacus comme un nouveau protocole de communication pouvant être utilisé à la place de IIOP, utilise, au niveau des communications réseaux, le protocole *Secure Socket Layer* (SSL)[49] pour sécuriser les échanges d'informations. Le protocole FSSLIOP est une solution non standardisée par l'OMG et totalement propriétaire à IONA et à *Orbacus*. Il est à noter que l'OMG est en train de spécifier le protocole sécurisé (Secure Inter-ORB Protocol) [OMG01a].

Une autorité de confiance (ou autorité de certification) est en charge de signer les certificats contenant la clé publique de chacune des parties. Le certificat est associé à une clé privée protégée par un mot de passe. Les données cryptées à l'aide de la clé privée peuvent être décryptées à l'aide de la clé publique. La signature des certificats permet de vérifier qu'ils ont bien été délivrés par une autorité de confiance. Ainsi quand une application externe cherche à se connecter sur le terminal de travail Spin, plusieurs opérations sont alors réalisées :

- un échange des certificats des deux parties est réalisé,
- une vérification est effectuée au niveau de la signature du certificat. On vérifie que le certificat délivré par l'autre partie est valide, ie. que la signature numérique contenue dans le certificat a bien été faite par l'autorité de certification,
- si les certificats sont valides, les deux parties peuvent alors s'échanger une clé symétrique tirée aléatoirement par une des parties. L'échange se fait à l'aide des clés asymétriques (clé publique/clé privée) : l'une des parties, A, tire, au hasard, une clé symétrique, l'encrypte avec la clé publique de l'autre partie B et envoie à B le message crypté. La partie B utilise sa clé privée pour décrypter le message reçu de la partie A et obtient la clé symétrique qui sera utilisée par la suite pour sécuriser les communications entre les deux parties. Chacune des parties est alors en possession de la même clé symétrique et est donc capable de crypter/décrypter les messages échangés entre les deux parties.

48. Il existe une extension IP, appelée *IPsec* permettant de sécuriser les paquets IP. Cette extension est à la base du mécanisme de sécurisation proposé dans la nouvelle version d'IP (IPv6).

49. `http://www.openssl.org`

7.4 Résultats expérimentaux

Nous avons réalisé une application de démonstration de notre interface de communication. Une application de contrôle d'un système de particules s'exécutant dans l'interface Spin a été développée. La figure 7.12 présente l'architecture générale de l'application réalisée. Le système de particules est implémenté dans l'interface en utilisant la notion de plug-in proposé par notre terminal :

- sur chaque machine, un plug-in de visualisation permet d'afficher le système de particules à partir d'un vecteur contenant la position (x,y,z) de chacune des particules. Le vecteur de position est partagé entre les terminaux en utilisant le mécanisme de communication interne présenté précédemment. La figure 7.11 présente le fichier VRML 97 décrivant le système de particule,

```
#VRML V2.0 utf8

EXTERNPROTO ParticleDisplay [
  exposedField MFVec3f position
]["urn:file:ParticleDisplay.dll"]

SharedSet {
  mode "public"
  children [
    SharedMFVec3f {
      alias "ParticleDisp.position"
      mode "public"
      how "flow"
    }
]}

Transform {
  children [
    DEF ParticleDisp ParticleDisplay {}
]}
```

FIG. 7.11 – *Définition VRML 97 du système de particules*

- sur une des machines, un plug-in de comportement permet de calculer l'animation pour chacune des particules. Il modifie directement les valeurs de position contenues dans le vecteur. Comme le vecteur de position est partagé, les terminaux distants sont notifiés des modifications.

L'application externe est connectée au terminal qui est en charge de calculer l'animation des particules. Elle permet de contrôler les différents paramètres du système de particules (comme par exemple, la gravité, le nombre de particules, etc.). L'application externe s'exécute sur un PDA (PocketPC) et est écrite en Java. Un réseau sans fil (*wifi*, IEEE 802.11b) assure la communication entre le PDA et le terminal Spin-3D contenant le plug-in de calcul de l'évolution des particules. Les différents éléments du dispositif expérimental sont présentés sur la figure 7.13.

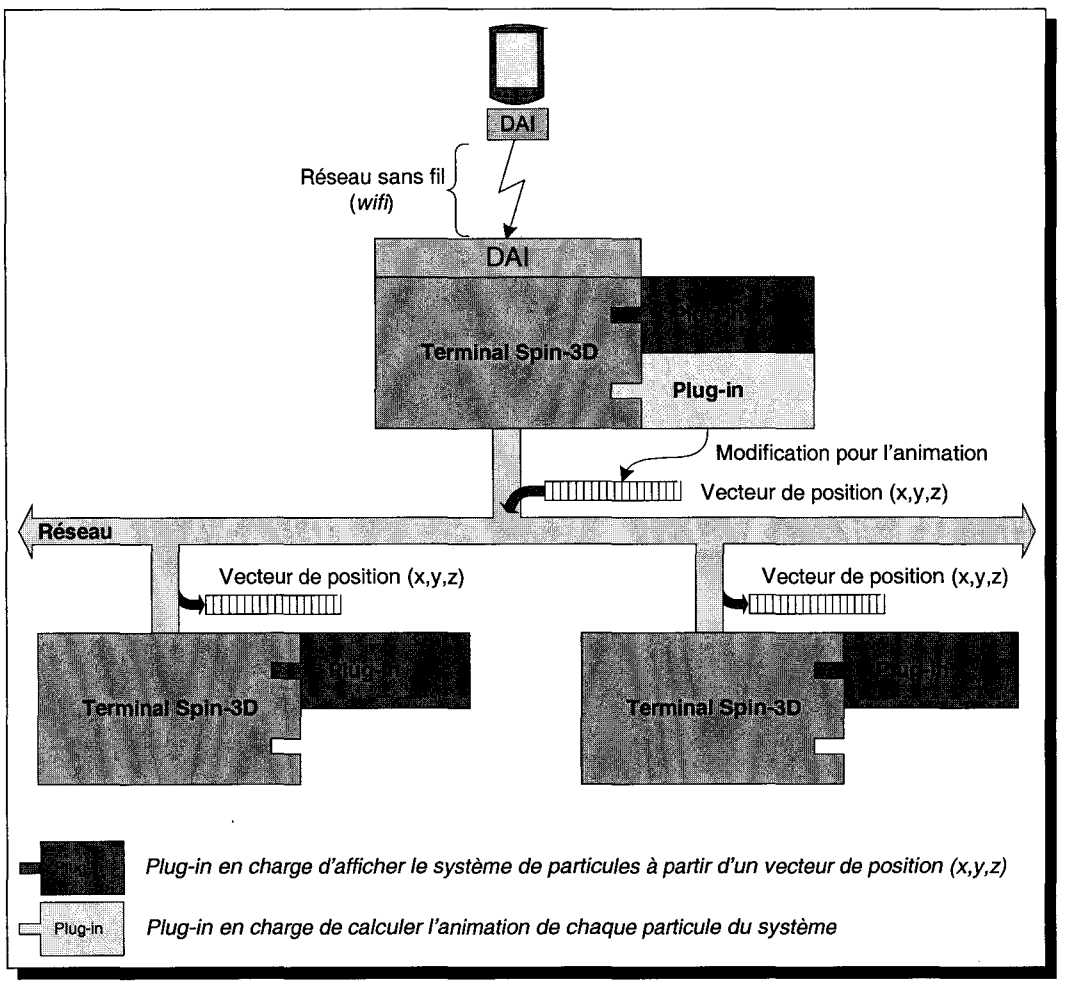
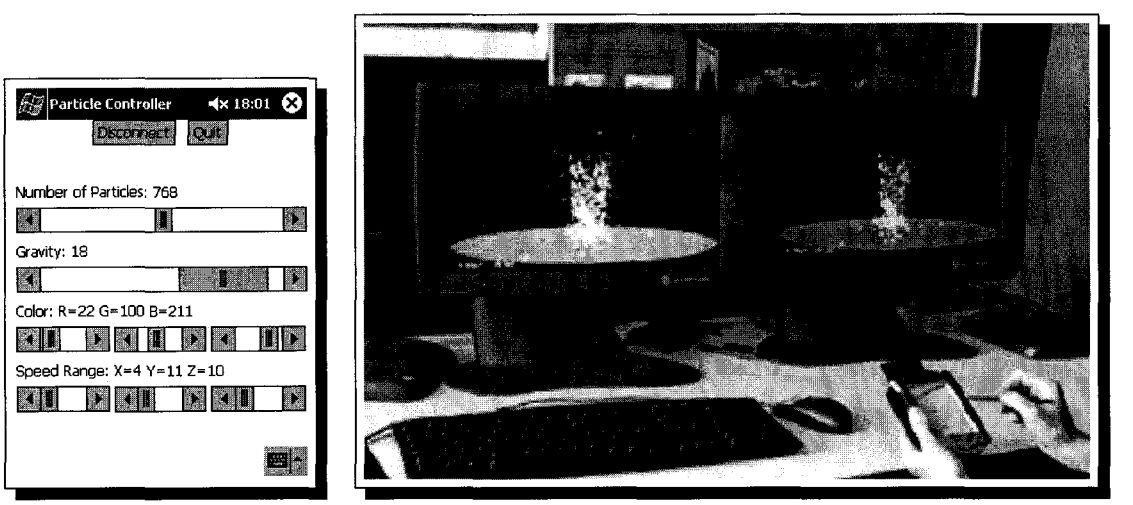


FIG. 7.12 – *Architecture de l'application du système de particules*



(a) interface du contrôleur s'exécutant sur le PDA

(b) interfaces du terminal de travail (le poste de gauche est en charge de calculer l'animation)

FIG. 7.13 – *Captures d'écran des différents éléments de l'application du système de particules*

7.5 Bilan

Dans ce chapitre, nous avons présenté l'interface de programmation que nous avons intégré à notre terminal de travail coopératif. Notre interface de programmation s'appuie sur l'EAI définie dans la norme VRML 97. La DAI permet de fusionner deux standards, d'un coté l'EAI VRML 97 classique, et de l'autre la technologie CORBA. L'utilisation de CORBA pour l'implémentation de notre interface de programmation nous permet de réaliser très simplement les supports multi-plateformes et multi-langages : des applications, écrites dans n'importe quel langage de programmation et s'exécutant à distance et sur n'importe quel système d'exploitation, sont capables se connecter sur notre terminal de travail coopératif et de manipuler le contenu de l'interface de travail. Les supports multi-plateformes et multi-langages sont une avancée par rapport aux implémentations classiques de l'EAI. L'EAI, telle que les navigateurs VRML 97 classiques l'implémentent, permettent uniquement à des applets Java de communiquer avec le navigateur VRML 97. Les applets Java et le navigateur VRML 97 s'exécutent sur la même machine, dans le même contexte, celui du navigateur HTML.

Nous avons présenté de nouvelles interfaces intégrées à l'EAI sous la forme d'extensions. Ces nouvelles interfaces offrent de nouveaux services pour les applications externes :

- le concept de `NodePath` permet de pouvoir pointer sur n'importe quel nœud contenu dans le graphe de scène,
- l'interface `NodeSearcher` permet de chercher des nœuds à partir de leur `NodePath` et d'obtenir leur référence,
- l'interface `NodeProxy` permet de parcourir le graphe de scène.

Nous avons également abordé les problèmes de sécurisation des communications entre une application externe et le terminal de travail. La solution proposée se place à bas niveau : elle permet uniquement de sécuriser les communications entre les applications externes et le terminal de travail. Cependant, on peut imaginer la sécurité en d'autres termes qu'un simple cryptage des paquets échangés. En effet, un système de plus haut niveau, plus sophistiqué, de permissions accordées aux applications externes permettrait de définir plusieurs niveaux d'accès au graphe de scène.

À partir de l'interface de programmation que nous avons définie, nous pouvons envisager de développer de multiples applications multi-utilisateurs, comme par exemple un outil de CAO 3D.

La figure 7.14 présente le schéma global de l'application coopérative :

- chaque utilisateur dispose de deux postes : un poste sur lequel s'exécute le terminal de coopération, et un autre poste exécutant l'application de CAO 3D. Les deux postes sont reliés en réseau : l'application de CAO 3D est connectée au terminal de travail coopératif en utilisant la DAI,
- lorsqu'un utilisateur manipule l'objet 3D depuis l'interface de l'application de CAO 3D, les modifications sont alors transmises via la DAI au terminal de travail coopératif afin qu'il mette à jour l'objet 3D présent dans l'interface de travail coopératif,
- l'objet 3D présent dans l'interface de travail coopérative est partagé, les terminaux distants sont prévenus des modifications opérées et mettent à jour l'objet 3D présent dans l'interface,
- en utilisant la DAI, l'objet 3D présent dans l'interface de l'application de CAO 3D peut également être mis à jour.

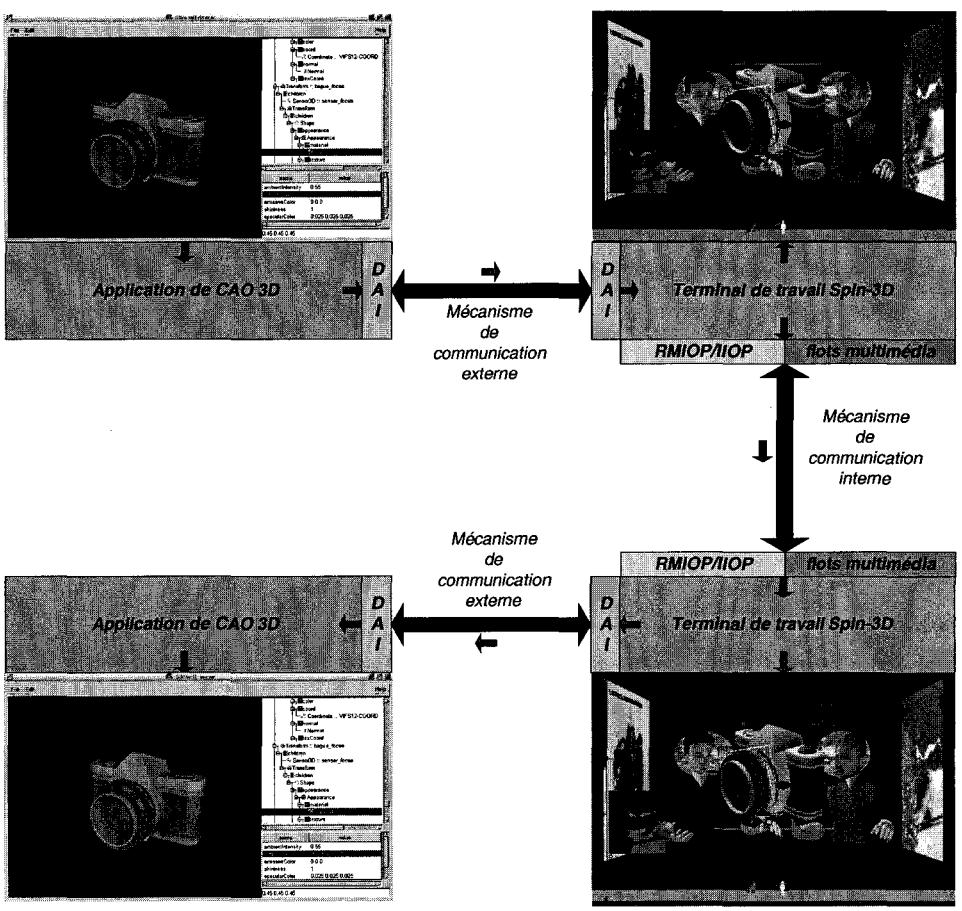


FIG. 7.14 – *Architecture générale d'une application de CAO 3D coopérative développée au dessus de la plate-forme Spin-3D*

Chapitre 8

Conclusion

8.1 Rappel des travaux réalisés

Dans cette thèse, nous nous sommes intéressés aux mécanismes de partage mis en oeuvre dans le terminal de travail coopératif Spin-3D afin de permettre aux utilisateurs de pouvoir coopérer au travers du réseau informatique. Le projet Spin-3D a pour but de définir une plate-forme pour le développement d'applications coopératives. Ainsi, toute application développée avec la plate-forme Spin-3D permet à un petit groupe de personnes de se réunir (virtuellement) et de coopérer pour réaliser une tâche définie à l'avance. Les différents protagonistes de la réunion, éloignés géographiquement, sont placés devant leur machine exécutant le terminal Spin-3D. Il s'agit de recréer virtuellement, au travers de l'interface, les conditions de la réunion réelle. Les utilisateurs du système doivent être capable d'agir sur des documents afin de les présenter, de les modifier ou même d'en produire de nouveaux.

Dans cette thèse, nous proposons tout d'abord les mécanismes bas-niveau de communication qui permettent aux différents terminaux présents à la session de travail virtuelle de dialoguer entre eux afin de maintenir cohérent l'état du monde virtuel partagé. Cela définit un mécanisme de communication que nous qualifions d'*interne*. Pour permettre aux utilisateurs de coopérer, lorsqu'un utilisateur agit sur un objet partagé, la manipulation doit être interactive. Pour cela, l'architecture de gestion de l'EVC 3D est décentralisée. Chaque poste exécutant le terminal de travail dispose de la base de données des objets 3D utilisés dans l'interface. Lorsqu'un utilisateur manipule une donnée partagée, son terminal est en charge de propager les modifications effectuées aux terminaux distants. À chaque donnée partagée, nous associons un groupe "logique" constitué de n copies, chacune d'elles étant gérée par un terminal de travail. Ensuite, nous utilisons un mécanisme de communication de groupe afin de maintenir la cohérence au sein de chaque groupe "logique". Nous offrons deux canaux pour réaliser les communications au sein d'un même groupe "logique" :

- un premier canal proposant une cohérence forte. Il repose sur le paradigme d'appel de méthodes à distance et utilise l'intergiciel CORBA. Pour les communications de groupe en CORBA, nous utilisons le nouveau protocole de communication MIOP dont nous avons fiabilisé la couche de transport à l'aide d'un protocole multipoint fiable qui utilise une hybridation de messages `NACK` et `ACK`,

- un deuxième canal proposant une cohérence faible. Il repose sur des flots multipoint établis entre les membres d'un même groupe "logique". Pour contrôler les flots, nous utilisons le service de flux multimédia, défini par l'OMG, dont nous avons adapté le fonctionnement à notre architecture de gestion décentralisée.

Pour gérer l'environnement 3D partagé, nous avons défini deux services décentralisés afin de tenir compte de notre architecture de gestion :

- un premier service est en charge de gérer les utilisateurs présents à la réunion virtuelle de travail. Ce service est utilisé à la fois à bas niveau, par notre protocole RMIOP pour fiabiliser les communications, et également à haut niveau, par le terminal de travail pour présenter les utilisateurs distants dans l'interface de travail. Le service de contrôle de groupe que nous proposons permet uniquement de gérer les problèmes liés au terminal lui même (pannes matérielles ou logicielles). Nous avons soulevé les autres problèmes, ie. ceux liés à l'infrastructure réseau, qui peuvent survenir (cf. 5.2.1.1, page 155), comme par exemple le problème du partitionnement du réseau ou encore les problèmes de visibilité. Pour ces différents problèmes complexes, il n'existe pas de solution générique : c'est à l'application utilisant notre plate-forme de résoudre ces différents problèmes,
- un deuxième service fournit les mécanismes utilisés par le terminal pour assurer qu'une donnée partagée n'est manipulée que par un seul utilisateur à la fois. Ce service est utilisé par le mécanisme d'interaction de l'interface de travail lorsqu'un utilisateur effectue une action sur une donnée partagée.

La conception d'une application coopérative utilisant la plate-forme Spin-3D doit rester simple et les différents problèmes liés à la distribution géographique des participants et au maintient de la cohérence des données partagées doivent être cachés au concepteur d'applications coopératives. La conception d'une application doit se limiter, dans la majorité des cas, à la création de contenus partagés (ie. les différents documents de la réunion coopérative). C'est pourquoi, nous proposons de décrire les différents objets 3D présents dans l'interface de travail en utilisant le langage VRML 97. Pour les objets 3D partagés, nous intégrons la description du partage au fichier VRML 97 contenant déjà la description de la géométrie et de l'interaction. La description du partage est une abstraction de haut niveau du mécanisme de communication interne utilisé pour maintenir la cohérence du monde 3D partagé. Notre mécanisme de partage VRML 97 est innovant dans le paradigme de partage utilisé. Nous sommes capable de remplacer n'importe quel champ d'un noeud VRML 97 par son équivalent partagé, ie. dont le comportement a été modifié afin de transmettre à ses copies distantes les modifications subies en utilisant le mécanisme de communication interne : c'est ce que nous appelons le paradigme de substitution. L'utilisation de ce nouveau paradigme facilite la conception d'objets VRML 97 partagés à partir d'objets déjà existants, et permet de tenir compte de l'évolution dynamique du partage d'un objet dans une application coopérative développée à partir de notre plate-forme.

Ce mécanisme VRML 97 de description d'objets 3D partagés permet de réaliser des applications coopératives intégrées directement à l'interface de travail. Pour concevoir des applications

coopératives plus complexes, nous offrons un mécanisme de communication que nous qualifions d'*externe* : des applications (comme par exemple celles dont le mécanisme d'interaction ne se prête pas à la 3D) sont capables de communiquer avec le terminal de travail coopératif afin de modifier son contenu. L'interface de programmation que nous proposons reprend les bases de l'EAI définie dans la norme VRML 97 et les étend à un contexte distribuée en utilisant l'intergiciel CORBA : les applications utilisant cette interface de communication s'exécutent en dehors de l'espace mémoire du terminal de travail coopératif, elles peuvent même s'exécuter sur une autre machine et communiquer avec le terminal au travers du réseau informatique. Notre interface de programmation intègre également de nouveaux services afin de permettre aux applications externes d'accéder, de parcourir et de modifier, plus facilement qu'avec l'EAI classique, le contenu du graphe de scène VRML 97.

8.2 Limitations et perspectives

Dans cette thèse, nous avons proposé un protocole multipoint fiable adapté à nos besoins. Du fait de l'utilisation de messages `ACK`, la scalabilité du protocole peut être remise en question. Cependant, la scalabilité n'est pas un de nos pré-requis : en effet, la plate-forme de développement Spin-3D doit permettre de mettre en relation un petit groupe de personnes (environ 5). Pour des raisons ergonomiques, il ne peut exister une réelle coopération entre les utilisateurs que lorsque leur nombre est réduit.

L'utilisation du multipoint peut freiner l'utilisation de notre terminal de travail coopératif : à l'heure actuelle, le multipoint n'est supporté sur Internet qu'au travers du réseau du MBone. Bien que le multipoint soit intégré nativement dans la prochaine version d'IP (IPv6), nous devons être capable de nous adapter aux contraintes techniques actuelles. Ainsi, tout en conservant une architecture de gestion décentralisée, nous offrons la possibilité de remplacer la couche de transport utilisée pour faire communiquer les différents terminaux entre eux. Dans le cadre de la démonstration "Argonaute 3D" [50], France Télécom a démontré la faisabilité d'une telle adaptation : la couche de transport multipoint a été remplacée par une passerelle unipoint sur laquelle les différents terminaux sont connectés. Lorsque la passerelle reçoit un message d'un terminal, elle est en charge de le diffuser aux autres terminaux de la session de travail.

Au niveau du protocole multipoint fiable, différents paramètres (période de ré-émission des fragments soumis à `ACK`, fragmentation des messages), doivent être réglés afin que le protocole multipoint fiable que nous proposons fonctionne dans les meilleures conditions possibles. Ces

50. `http://www.rd.francetelecom.fr/fr/galerie/navig_argonaute3D.htm`

paramètres sont à adapter en fonction des paramètres du réseau (charge, pertes, bande passante, latence). Pour cela, nous devons réaliser des tests en grandeur réelle de notre protocole multipoint fiable, notamment sur le réseau MBone. Il peut être également intéressant d'étudier un mécanisme d'adaptation automatique de ces différents paramètres (comme le propose par exemple le protocole RTP avec l'utilisation de RTCP).

Après avoir adopté le protocole MIOP, l'OMG s'intéresse maintenant à sa fiabilisation en ayant diffusé, à la fin de l'année 2002, un RFP [OMG02]. Le protocole *Reliable Ordered Multicast Inter-ORB Protocol* (ROMIOP) devrait permettre d'offrir très facilement un système à tolérance de panne. Les principes de communications de groupe introduit avec le protocole MIOP sont toujours valables. Cependant, avec ROMIOP, les invocations de méthodes sur un groupe d'objets sont transmis de manière fiable et sont exécutées de façon totalement ordonnée sur chacun des objets du groupe. L'évolution de cette future norme est donc à suivre. La norme ROMIOP couvre des problèmes beaucoup plus larges que ceux que nous avons abordés : dans la mesure où la norme ROMIOP se propose de respecter une *transmission totalement ordonnée*, les messages émis par une ou plusieurs sources à destination de plusieurs groupes "logiques" doivent être exécutés dans le même ordre sur tous les sites hébergeant des membres de ces groupes "logiques".

Dans cette thèse, nous n'avons que très peu abordé le problème de la sécurité des communications. Pour protéger la session de travail, il nous semble important de sécuriser les communications entre les différents terminaux, et entre un terminal et les différentes applications qui lui sont connectées. Des données sensibles peuvent être manipulées.

En ce qui concerne les communications entre les applications externes et le terminal de travail, il existe des solutions simples à mettre en œuvre comme le protocole FSSLIOP. Cependant, cela reste une solution propriétaire au bus CORBA Orbacus. Ici encore, il faut suivre l'évolution de la norme, CSIv2 [OMG01a], proposée par l'OMG à ce niveau.

Pour sécuriser les communications entre les différents terminaux, dans le contexte technique dans lequel nous nous plaçons, cela revient à sécuriser les communications multipoint. Pour cela, comme il est mentionné dans [MRR91], il faut assurer que :

- si un utilisateur entre dans la session de travail, il ne puisse lire les messages émis avant son arrivée,
- si un utilisateur quitte la session de travail, il doit être incapable de lire les messages émis après son départ,
- les messages émis au groupe doivent être authentifiés.

Pour les deux premiers points, une approche est d'utiliser une clé symétrique comme le propose la couche sécurisée IP (IPsec) : à chaque entrée ou sortie d'un utilisateur, la clé symétrique doit être modifiée et transmise aux différents membres authentifiés du groupe multipoint. Cela nécessite un service centralisé en charge de gérer les authentifications des utilisateurs et de transmettre aux utilisateurs authentifiés la clé symétrique en utilisant des certificats et des clés asymétriques. Des solutions industrielles, intégrant ces différents problèmes, commencent à émerger. On peut notamment citer comme exemple UPcrypt proposé par UDcast[51]. Cependant, toute personne ayant accès à la clé symétrique est alors capable d'envoyer des messages sur le canal multipoint sécurisé. Le dernier point permet donc aux récepteurs de s'assurer qu'un message a bien été émis par un émetteur authentifié. Dans la mesure où l'OMG propose CSIv2 permettant de sécuriser le protocole IIOP, nous pouvons très facilement imaginer que l'OMG va s'intéresser à la sécurisation des protocoles MIOP et ROMIOP.

De nouveaux services peuvent être développés en utilisant les mécanismes de communication proposés. Ces nouveaux services doivent tenir compte de l'architecture de gestion décentralisée :

- un service pour la gestion des identifiants des groupes logiques. À chaque groupe logique, nous associons un identifiant unique utilisé par le protocole RMIOP et par le flot multimédia. Nous proposons l'utilisation du service du nommage défini par l'OMG. Il serait utilisé comme une base de données permettant à partir d'un nom symbolique (celui de la donnée partagée) d'obtenir l'identifiant du groupe logique auquel elle appartient. Nous utilisons le même schéma de conception que pour le service de groupe ou le service de jetons : dans chaque terminal s'exécute un gestionnaire de nommage, les différents gestionnaires formant un groupe logique représentant le service de nommage. Dès qu'un terminal modifie le contenu de son gestionnaire de nommage (comme par exemple, l'ajout d'une nouvelle entrée), il prévient les gestionnaires distants de la modification. L'objet gestionnaire de nommage est implémenté sous la forme d'un objet CORBA et communique avec ses copies distantes en utilisant le protocole de communication de groupe RMIOP,
- un service pour la gestion de la persistance. Lorsqu'un utilisateur entre en session, le système doit fournir l'état courant des différents objets partagés. Pour cela, il est possible d'utiliser les informations contenues dans un nœud `SharedSet`. À son entrée dans la session, le nouvel arrivant peut demander la valeur courante des différentes propriétés exposées par un objet 3D partagé. En utilisant un mécanisme de sérialisation appliqué à chaque nœud `SharedSet`, un terminal, arrivant en session, est capable de recevoir des autres terminaux, déjà présents à la session, l'état courant des objets 3D partagés. De même les objets chargés par les utilisateurs au cours de la session doivent être transmis aux nouveaux arrivants.

51. `http://www.udcast.com`

À haut niveau, nous pouvons imaginer intégrer d'autres formats de fichiers 3D, comme par exemple X3D, le nouveau format du consortium Web3D. En proposant une description géométrique sous la forme d'un graphe hiérarchique, similaire au graphe VRML 97, notre terminal de travail peut très facilement être adapté afin de gérer, en natif, des fichiers au format X3D. Nous devons donc étudier comment intégrer le mécanisme de description de partage en utilisant la syntaxe proposée par le format X3D.

Le concepteur d'applications coopératives se doit de respecter quelques règles de conception, notamment pour la définition des verrous utilisés par le mécanisme d'interaction afin d'assurer à un utilisateur qu'il est le seul à pouvoir réaliser une action sur une donnée partagée. Dans le système que nous proposons actuellement, à chaque noeud de partage (`SharedNode`), le concepteur doit spécifier le même nom de verrou que celui utilisé au niveau du (des) `Sensors3D` susceptible(s) de le modifier depuis l'interface. Ainsi, pour limiter ces contraintes au moment de la conception des objets 3D partagés, nous devons proposer un algorithme permettant de remplir, de façon automatique, le champ `lockName` d'un `SharedNode` donné.

Chaque objet 3D chargé par notre terminal est décrit dans un fichier VRML 97 contenant la géométrie, l'interaction ainsi que, pour les objets 3D partagés, la description du partage. Le problème de cette approche est de lier un objet abstrait à son implémentation. L'idée serait d'abstraire complètement les objets 3D de l'interface afin de séparer un objet, et ses fonctionnalités, de son implémentation : c'est la notion de modèle de composant 3D. L'abstraction entre l'interface d'un objet et l'implémentation permet alors au concepteur de composer très facilement son application. Il lui est possible de changer un composant par un autre composant, offrant la même interface, sans pour autant revoir toute son application. Avec une architecture à composants, nous pouvons par exemple très facilement remplacer, de façon transparente, des objets dont le comportement est émulé par des objets équivalent mais dont le comportement est simulé.

La définition d'un composant 3D implique de pouvoir spécifier son interface, ie. l'abstraction nécessaire au client pour détacher le composant de son implémentation. Pour fonctionner, un composant peut avoir besoin d'autres composants : pour cela, un composant offre un ou plusieurs services utilisables par d'autres composants. Dans son interface, un composant expose également des propriétés de configuration ainsi que plusieurs entrées/sorties : ainsi, le concepteur d'applications est capable à la fois de configurer le composant selon ses besoins et également de connecter les sorties d'un composant aux entrées d'autres composants afin de composer son application. Une solution permettant cette abstraction est l'utilisation du modèle MVC, Modèle/Vue/Contrôleur : le Modèle est la logique de l'objet, la Vue est la représentation d'un objet dans l'interface, et enfin le Contrôleur permet à l'utilisateur d'agir sur le modèle au travers de la vue.

Bibliographie

[Ara98] Y. ARAKI – "VSPLUS: A High-level Multi-user Extension Library for Interactive VRML Worlds", *Actes de VRML'98*, Février 1998, p. 39–47.

[Ban96] S. BANGAY – "Modelling Parallel and Distributed Virtual Reality Systems for Performance Analysis and Comparison", Thèse, Department of Computer Science, Rhodes University, 1996.

[BF93] S. BENFORD et L. E. FAHLÉN – "A Spatial Model of Interaction in Large Virtual Environments", *Actes de European conference on Computer Supported Collaborative Work (ECSCW)*, Kluwer Academic, 1993, p. 107–.

[BGB+95] S. BENFORD, C. GREENHALGH, J. BOWERS, D. SNOWDON et L. E. FAHLÉN – "User Embodiment in Collaborative Virtual Environments", *Actes de Computer Human Interface (CHI) conference*, ACM Press, 1995, p. 242–248.

[BPP95] G. BELL, A. PARISI et M. PESCE – *The Virtual Reality Modeling Language version 1.0 specification*, 1995, `http://www.vrml.org/VRML1.0/`.

[BPPT01] C. BOURAS, D. PSALTOULIS, C. PSAROUDIS et T. TSIATSOS – "A Platform for Sharing Educational Virtual Environments", *Proceedings of the 9^th International Conference on Software Telecommunications and Computer Networks (SoftCOM 2001)*, Octobre 2001, p. 659–666.

[Bro98] W. BROLL – "DWTP – An Internet Protocol for Shared Virtual Environments", *Actes de VRML'98*, Février 1998, p. 49–56.

[BS98] W. BROLL et D. SCHICK – "DWTP – A Basis for Networked VR on the Internet", *Actes de Electronic Imaging '98*, Janvier 1998, p. 370–380.

[Bux92] W. A. S. BUXTON – "Telepresence: Integrating Shared Task and Person Spaces", *Actes de Graphics Interface*, 1992, p. 123–129.

[BvR94] K. BIRMAN et R. VAN RENESSE – *Reliable Distributed Computing With The ISIS Toolkit*, IEEE Computer Society Press, 1994.

[BWA96] J. W. BARRUS, R. C. WATERS et D. B. ANDERSON – "Locales: Supporting Large Multiuser Virtual Environments", *IEEE Computer Graphics and Applica-*

tions **16** (1996), no. 6, p. 50–57.

[BZWM97] D. BRUTZMAN, M. ZYDA, K. WATSEN et M. MACEDONIA – "Virtual Reality Transfer Protocol (VRTP) Design Rationale", *Actes de IEEE Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE'97)*, Juin 1997, p. 179–186.

[CBM97] R. CAREY, G. BELL et C. MARRIN – *ISO/IEC 14772-1:1997 Virtual Reality Modeling Language (VRML97)*, 1997, `http://www.web3d.org/Specifications/VRML97/`.

[CC99] J. A. CARSON et A. F. CLARK – "Multicast Shared Virtual Worlds using VRML97", *Actes de VRML'99*, Février 1999, p. 133–140.

[CDG+93] J. CALVIN, A. DICKEN, B. GAINES, P. METZGER, D. MILLER et D. OWEN – "The SIMNET Virtual World Architecture", *Actes de IEEE Virtual Reality Annual International Symposium*, Septembre 1993, p. 450–455.

[CH93a] C. CARLSSON et O. HAGSAND – "DIVE – A Multi User Virtual Reality System", *Actes de IEEE Virtual Reality Annual International Symposium (VRAIS)*, Septembre 1993, p. 394–400.

[CH93b] C. CARLSSON et O. HAGSAND – "DIVE – A Platform for Distributed Multi-User Virtual Environments", *Computer & Graphics* **17** (1993), no. 6, p. 663–669.

[CLW98] Y. HONDA AND MITRA – *Core Living Worlds*, 1998, `http://www.web3d.org/WorkingGroups/living-worlds/CoreLW/`.

[CNT+97] T. K. CAPIN, H. NOSER, N. M. THALMANN, I. S. PANDZIC et D. THALMANN – "Virtual Human Representation and Communication in VLNet", *IEEE Computer Graphics and Applications* **17** (1997), no. 2, p. 42–53.

[Dee89] S. DEERING – *Host extensions for IP multicasting*, 1989, RFC 1112.

[DM96] T. DUVAL et D. MARGERY – "Building objects and interactors for collaborative interactions with GASP", *Actes de IEEE VRAIS'96*, Janvier 1996, p. 214–221.

[DPD00] C. DUMAS, P. PLÉNACOSTE et S. DEGRANDE – "Les indices dynamiques dans les interfaces 3d – Au delà des ombres portées", *Actes de Ergo-IHM 2000*, Octobre 2000, p. 64–68.

[DSM+97] T. K. DAS, G. SINGH, A. MITCHELL, P. S. KUMAR et K. MCGEE – "NetEffect: A Network Architecture for Large-scale Multi-user Virtual Worlds", *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST'97*, Septembre 1997, p. 157–163.

[Dum97] C. DUMOULIN – "Dream : Une mémoire partagée répartie à cohérence programmable", Thèse, Université des Sciences et Technologies de Lille, 1997.

[Dum99] C. DUMAS – "Un modèle d'interaction 3D : Interactions homme-machine et

homme-machine-homme pour les interfaces 3D pour le TCAO synchrone", Thèse, Université des Sciences et Technologies de Lille, 1999.

[EGR91] C. A. ELLIS, S. J. GIBBS et G. REIN – "Groupware: some issues and experiences", *Communications of the ACM* **34** (1991), no. 1, p. 39–58.

[Fen97] W. FENNER – *Internet Group Management Protocol version 2*, 1997, RFC 2236.

[FFH+99] F. FAURE, C. FAISSTNAUER, G. HESINA, A. AUBEL, M. ESCHER, F. LABROSSE, J.-C. NEBEL et J.-D. GASCUEL – "Collaborative Animation over the Network", *Actes de Computer Animation'99*, Avril 1999, p. 107–116.

[FJL+97] S. FLOYD, V. JACOBSON, C. LIU, S. MCCANNE et L. ZHANG – "A Reliable Multicast Framework for Light–Weight Sessions and Application Level Framing", *IEEE/ACM Transactions on Networking* **5** (1997), no. 6, p. 784–803.

[FS98] E. FRÉCON et M. STENIUS – "Dive: A scaleable network for distributed virtual environments", *Distributed Systems Engineering Journal (special issue on istributed Virtual Environments* **5** (1998), no. 3, p. 91–100.

[Fun95] T. A. FUNKHOUSER – "RING: A Client-Server System for Multi-User Virtual Environments", *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, Avril 1995, p. 85–92.

[Fun96] T. A. FUNKHOUSER – "Network Topologies for Scalable Multi-User Virtual Environments", *Actes de IEEE VRAIS'96*, Avril 1996, p. 222–229.

[FvDFH90] J. D. FOLEY, A. VAN DAM, S. K. FEINER et J. F. HUGHES – *Computer Graphics: Principles and Practice (2^{nd} edition)*, Addison–Wesley Publishing Compagny, 1990.

[GB94] C. GREENHALGH et S. BENFORD – "Real-time groupware as a distributed system: Concurrency control and its effect on the interface", *Actes de CSCW'94*, Octobre 1994, p. 207–217.

[GB95a] C. GREENHALGH et S. BENFORD – "MASSIVE: a Collaborative Virtual Environment for Tele-conferencing", *ACM Transaction on Computer Human Interfaces (TOCHI)* **2** (1995), no. 3, p. 239–261.

[GB95b] C. GREENHALGH et S. BENFORD – "Massive: A distributed virtual reality system incorporating spatial trading", *Actes de IEEE DCS'95*, 1995, p. 27–35.

[GBD+94] A. GEIST, A. BEGUELIN, J. DONGARRA, W. JIANG, R. MANCHEK et V. SUNDERAM – *PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, 1994.

[GD98] L. GAUTIER et C. DIOT – "Design and evaluation of MiMaze, a multi-player game on the internet", *Actes de IEEE Multimedia Systems*, Juin 1998, p. 233–236.

[GG99] C. GRANSART et J. M. GEIB – "Using an ORB with Multicast IP", *Proceedings of PCS'99 (Parallel Computing Systems)*, Août 1999.

[GHJV95] E. GAMMA, R. HELM, R. JOHNSON et J. VLISSIDES – *Design Pattern, Elements of Reusable Object-Oriented Software*, vol. ISBN 0-201-63361-2, Addison-Wesley, 1995.

[GTP99] K. GREIN, O. THUNIN et P. PLÉNACOSTE – *Tests ergonomiques de l'application SPIN*, Mai 1999, Note Technique NT/CNET/6238.

[Hag95] O. HAGSAND – *SID2 Interface Specification*, Août 1995, Swedish Institute of Computer Science, `http://www.sics.se/ olof/sid2.html`.

[Hag97] O. HAGSAND – "Using Spatial Techniques to Decrease Message Passing in a Distributed VE System", *Actes de VRML'97*, Février 1997, p. 7–15.

[HC86] D. A. HENDERSON et S. J. CARD – "Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface", *ACM Transactions on Graphics* **5** (1986), no. 3, p. 211–243.

[HHBC92] O. HAGSAND, H. HERZOG, K. BIRMAN et R. COOPER – "Object-Oriented Reliable Distributed Programming", *Proceedings of the* 2^{nd} *International Workshop on Object-Orientation in Operating Systems*, Septembre 1992, p. 180–188.

[HLA00] ELECTRICAL AND ELECTRONICS ENGINEERS, INC. – *IEEE Standard for Modeling and Simulation (M&S) - High Level Architecture (HLA)*, Septembre 2000, IEEE Standard 1516.

[HMRL95] Y. HONDA, K. MATSUDA, J. REKIMOTO et R. LEA – "Virtual Society: Extending the WWW to Support a Multi-user Interactive Shared 3D Environment", *Actes de VRML'95*, Décembre 1995, p. 109–116.

[HW96] J. HARTMAN et J. WERNECKE – *The VRML 2.0 Handbook, Building Moving Worlds on the Web*, Addison–Wesley Publishing Compagny, 1996.

[HW99] R. HAWKES et M. J. WRAY – "LivingSpace: A Living Worlds Implementation using an Event-based Architecture", 1999, Hewlett-Packard Technical Report HPL-98-181.

[Iona] IONA – *The FreeSSL protocol homepage*, `http://www.ooc.nf.ca/fssl/`.

[Ionb] IONA – *Orbacus for C++ and Java*, Manuel de programmation, version 4.0.5.

[Ionc] IONA – *The Orbacus website*, `http://www.orbacus.com`.

[IST95] INSTITUTE FOR SIMULATION AND TRAINING – *Standard for distributed interactive simulation - application protocols*, 1995, IEEE 1278.

[JBBCN98] C. JUST, A. BIERBAUM, A. BAKER et C. CRUZ-NEIRA – "VR Juggler: A Framework for Virtual Reality Development", *Proceedings of the* 2^{nd} *Immersive Projection Technology Workshop (IPT'98)*, Mai 1998.

[Kar94] A. KARSENTY – "Le collecticiel : de l'interaction homme-machine à la communication homme-machine-homme", *Technique et Science Informatiques* **13** (1994), no. 1, p. 105–127.

[KBH00] G. KESSLER, D. BOWMAN et L. HODGES – "The Simple Virtual Environment Library: An Extensible Framework for Building VE Applications", *Presence – Teleoperators and Virtual Environments* **9** (2000), no. 2, p. 187–208.

[KG86] H. KRASNER et I. GREIF – "Actes de Conference on Computer–Supported Cooperative Work", ACM Press, 1986.

[KH00] G. D. KESSLER et L. F. HODGES – "A Network Communication Protocol for Distributed Virtual Environment Systems", *Proceedings of the third International Conference on Collaborative Virtual Environments (CVE'2000)*, Septembre 2000, p. 129–138.

[KS02] J. KELLER et G. SIMON – "Toward a Peer-to-Peer Shared Virtual Reality", *Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops (ICDCSW'02)*, Juillet 2002, p. 695–700.

[KZ96] A. KOIFMAN et S. ZABELE – "RAMP: A reliable adaptive multicast protocol Multicast Protocol", *Actes de IEEE INFOCOM'96*, Mars 1996, p. 1442–1451.

[LDPDG+01] S. LOUIS DIT PICARD, S. DEGRANDE, C. GRANSART, G. SAUGIS et C. CHAILLOU – "A CORBA Based Platform as Communication Support for Synchronous Collaborative Virtual Environments", *Actes de M3W (Mutimedia Middleware Workshop), en marge de la conférence ACM Multimedia*, Octobre 2001.

[LDPDG+02] S. LOUIS DIT PICARD, S. DEGRANDE, C. GRANSART, G. SAUGIS et C. CHAILLOU – "VRML Data Sharing in the Spin-3D CVE", *Actes de Web3D'02*, Février 2002, p. 165–172.

[LDPDG+03] S. LOUIS DIT PICARD, S. DEGRANDE, C. GRANSART, G. SAUGIS et C. CHAILLOU – "VRML97 Distributed Authoring Interface", *Actes de Web3D'03*, Mars 2003, p. 135–145.

[LGLA98] B. N. LEVINE et J. J. GARCIA-LUNA-ACEVES – "A comparison of reliable multicast protocols", *Multimedia Systems* **6** (1998), no. 5, p. 334–348.

[LHMM97] R. LEA, Y. HONDA, K. MATSUDA et S. MATSUDA – "Community Place - Architecture and Performance", *Actes de VRML'97*, Février 1997, p. 41–50.

[Li86] K. LI – "Shared Virtual Memory on Loosely Coupled Multiprocessors", Thèse, Université de Yale, 1986.

[Lia98] T. LIAO – *Lightweight Reliable Multicast Protocol Specification*, Octobre 1998, Internet-Draft, <`draft-liao-lrmp-00.txt`>, disponible à l'adresse `http://webcanal.inria.fr/lrmp/`.

[LJK03] J. M. LINEBARGER, C. D. JANNECK et G. D. KESSLER – "SSVE: A Framework and Interaction Mechanisms for Highly Dynamic Object-Focused Collaboration", *Submitted to ACM Symposium on Virtual Reality Software and Technology (VRST 2003)*, Octobre 2003.

[LM01] P. LE MER – "Modèle de communication Homme-Clone-Homme pour les Environnements Virtuels Collaboratifs non-immersifs", Thèse, Université des Sciences et Technologies de Lille, 2001.

[Loc95] J. LOCKE – "An Introduction to the Internet Networking Environment and SIMNET/DIS", Tech. report, Computer Science Departement, Naval Postgraduate School, 1995.

[LP96] J. C. LIN et S. PAUL – "RMTP: A Reliable Multicast Transport Protocol", *Actes de IEEE INFOCOM'96*, Mars 1996, p. 1414–1424.

[Lét00] E. LÉTY – "Une architecture de communication pour environnements virtuels distribués à grande-échelle sur l'Internet", Thèse, Université de Nice-Sophia Antipolis, 2000.

[Lév90] P. LÉVY – *Les technologies de l'intelligence*, Éditions La découverte, 1990.

[LW98] LIVING WORLDS WORKING GROUP – *Making VRML 97 Applications Interpersonal and Interoperable*, 1998, `http://www.web3d.org/WorkingGroups/living-worlds/`.

[MAC+02] D. MARGERY, B. ARNALDI, A. CHAUFFAUT, S. DONIKIAN et T. DUVAL – "OpenMASK: Multi-Threaded or Modular Animation and Simulation Kernel or Kit : a general introduction", *Actes de VRIC'02*, Juin 2002, p. 101–110.

[Mar01] D. MARGERY – "Environnement logiciel temps-réel distribué pour la simulation sur réseau de PC", Thèse, Université de Rennes 1, 2001.

[MF98] B. MACINTYRE et S. FEINER – "A Distributed 3D Graphics Library", *Actes de SIGGRAPH'98*, Juillet 1998, p. 361–370.

[MP4] THE MOVING PICTURE EXPERTS GROUP – *The website*, `http://mpeg.cselt.it`.

[MPB03] J.-E. MARVIE, J. PERRET et K. BOUATOUCH – *Remote Interactive Walkthrough of City Models Using Procedural Geometry*, Juillet 2003, Publication interne n°1546.

[MRR91] M. MOYER, J. R. RAO et P. ROHATGI – "A Survey of Security Issues in Multicast Communications", *IEEE Network* (1991), p. 12–23.

[MSS99] S. MUNGEE, N. SURENDRAN et D. C. SCHMIDT – "The Design and Specification of a CORBA Audio/Video Streaming Service", *Actes de HICSS-32 Hawaii International Conference on System Sciences* (Hawaii), Janvier 1999.

[MZ97] M. R. MACEDONIA et M. ZYDA – "A Taxonomy for Networked Virtual Environments", *IEEE Multimedia* **4** (1997), no. 1, p. 48–56.

[MZP+94] M. R. MACEDONIA, M. J. ZYDA, D. R. PRATT, P. T. BARHAM et S. ZESWITZ – "NPSNET: A Network Software Architecture for Large Scale Virtual Environments", *Presence: Teleoperators and Virtual Environments* **3** (1994), no. 4, p. 265–287.

[OMG99] OMG – "Unreliable Multicast Inter-ORB Protocol Request For Proposal", OMG Document orbos/99-11-14, Novembre 1999.

[OMG00] OMG – "Control and Management of Audio/Video Streams Specification", OMG Document formal/2000-01-03, Janvier 2000.

[OMG01a] OMG – "Common secure interoperability protocol version 2 (csiv2)", OMG Document ptc/01-06-17, Juin 2001.

[OMG01b] OMG – "CORBA Scripting Language Specification", OMG Document formal/01-06-05, Juin 2001.

[OMG01c] OMG – "Naming Service Specification", OMG Document formal/2001-02-65, Février 2001.

[OMG01d] OMG – "The Common Object Request Broker: Architecture and Specification Revision 2.5", OMG Document formal/01-09-01, Septembre 2001.

[OMG01e] OMG – "Unreliable Multicast Inter-ORB Protocol", OMG Document orbos/2001-03-01, Avril 2001.

[OMG02] OMG – "Reliable Ordered Multicast Inter-ORB Protocol Request For Proposal", OMG Document realtime/2002-11-28, Novembre 2002.

[OMIM94] K.-I. OKADA, F. MAEDA, Y. ICIKAWA et Y. MATSUSHITA – "Multiparty Videoconferencing at Virtual Social Distance: MAJIC Design", *Actes de Computer Supported Collaborative Work (CSCW) conference*, ACM Press, 1994, p. 385–393.

[PA94] P. PRIMET et S. AKKOUCHE – "Réflexion autour du concept de télépointeur et réalisation d'un outil coopératif de désignation", *Actes de la Conférence Interface Homme Machine (IHM)*, 1994, p. 176–182.

[RCRW99] G. REITMAYR, S. CARROLL, A. REITEMEYER et M. WAGNER – "DeepMatrix - An Open Technology Based Virtual Environment System", *The Visual Computer Journal* **15** (1999), no. 7/8, p. 395–412.

[RDS03] J. L. ROBINSON, S. DUMOULIN et J. A. STEWART – "MVIP II – A Protocol for Enabling Communication in Collaborative Virtual Environments", *Actes de Web3D'03*, Mars 2003, p. 155–160.

[RP94] J. REYNOLDS et J. POSTEL – *Assigned Numbers*, 1994, RFC 1700.

[RSL00] J. L. ROBINSON, J. A. STEWART et I. LABBE – "MVIP - Audio Enabled Multicast VNet", *Actes de VRML'00*, Février 2000, p. 103–109.

[Saa99] K. SAAR – "VIRTUS: A Collaborative Multi-user Platform", *Actes de VRML'99*, Février 1999, p. 141–152.

[Sau98] G. SAUGIS – "Interface 3D pour le travail coopératif synchrone, une proposition", Thèse, Université des Sciences et Technologies de Lille, 1998.

[SCFJ96] H. SCHULZRINNE, S. CASNET, R. FREDERICK et V. JACOBSON – *RTP: A Transport Protocol for Real-Time Applications*, 1996, RFC 1889.

[SDW92] W. STRAYER, B. DEMPSEY et A. WEAVER – *XTP: The Xpress Transfer Protocol*, Addison–Wesley, 1992.

[SK01] O. SCHREER et P. KAUFF – "An Immersive 3D Videoconferencing System–based on a Shared Virtual Environment", *Proceedings of the International Conference on Media Futures*, 2001.

[SSP+95] G. SINGH, L. SERRA, W. PNG, A. WONG et H. NG – "BrickNet: Sharing objects behaviors on the net", *Actes de IEEE Virtual Reality Annual International Symposium (VRAIS)*, Mars 1995, p. 19–25.

[SYW99] U. SUNG, J. YANG et K. WOHN – "Concurrency Control in CIAO", *Actes de IEEE VR'99*, Mars 1999, p. 22–28.

[TD01] D. TREGLIA et M. DELOURA – *Game Programming Gems 3*, vol. ISBN 1-58450-233-9, Charles River Media, 2001.

[Tra99] H. TRAMBEREND – "Avocado: A Distributed Virtual Reality Framework", *Actes de IEEE VR'99*, Mars 1999, p. 14–21.

[VDMKCS+99] F. J. VANDERLEI DERIGGI, M. MASSAKUNI KUBO, A. CARLOS SEMENTILLE, J. R. FERREIRA BREGA, S. GARCON DOS SANTOS et C. KIRNER – "CORBA Platform as Support for Distributed Virtual Environments", *Actes de IEEE VR'99*, Mars 1999, p. 8–13.

[Ver99] R. VERTEGAAL – "The GAZE Groupware System: Mediating Joint Attention in Multiparty Communication and Collaboration", *Proceedings of the Computer Human Interface (CHI) conference*, ACM Press, 1999, p. 294–301.

[W3C99] W3C – *XML Path Language (XPath) version 1.0*, Novembre 1999, `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[WAB+96] R. C. WATERS, D. B. ANDERSON, J. W. BARRUS, D. C. BROGAN, M. A. CASEY, S. G. MCKEOWN, T. NITTA, I. B. STERNS et W. S. YERAZUNIS – "Diamond Park and Spline: A Social Virtual Reality System with 3D Animation, Spoken Interaction, and Runtime Modifiability", Tech. Report TR96-02a, Mitsubishi Electric, 1996.

[WAS97] R. C. WATERS, D. B. ANDERSON et D. L. SCHWENKE – "Design of the in-

teractive sharing transfer protocol", *Proceedings of WET ICE'97 - IEEE Sixth Workshops on Enabling Technologies for Collaborative Enterprises: Infrastructure for Collaborative Enterprises*, Juin 1997, p. 140–147.

[Wex93] A. WEXELBLAT – *The Reality of Cooperation: Virtual Reality and CSCW*, Academic Press, 1993.

[Whi98] S. F. WHITE – *VRML Interchange Protocol – Specification*, 1998, `http://www.csclub.uwaterloo.ca/u/sfwhite/vnet/VIP.html`.

[WKM94] B. WHETTEN, S. KAPLAN et T. MONTGOMERY – "A High Performance Totally Ordered Multicast Protocol", *Dagstuhl Seminar on Distributed Systems*, Septembre 1994, p. 33–57.

[WS] S. F. WHITE et J. SONSTEIN – *VNet – A multi-user VR system.*

[WW92] A. H. WATT et M. WATT – *Advanced Animation and Rendering Techniques: Theory and Practice*, Addison–Wesley Publishing Compagny, 1992.

[X3D] THE WEB3D CONSORTIUM – *X3D Architecture and API - ISO/IEC FCD 19775:200x*, `http://www.web3d.org/fs_specifications.htm`.

[YGS95] R. YAVATKAR, J. GRIFFOEN et M. SUDAN – "A Reliable Dissemination Protocol for Interactive Collaborative Applications", *Actes de ACM Multimedia'95*, 1995, p. 333–344.

Index des références

[LDPDG+03], 180
[LGLA98], 68
[LHMM97], 51, 81
[LJK03], 105
[LM01], 4, 121
[LP96], 69
[LW98], 77
[Lét00], 67
[Lév90], 28
[Li86], 55
[Lia98], 69
[Loc95], 108
[MAC+02], 58, 106
[MF98], 43
[MP4], 26, 91
[MPB03], 44
[MRR91], 205
[MSS99], 61, 236
[MZ97], 43, 50, 52
[MZP+94], 17, 23, 41, 45, 51, 108
[Mar01], 58, 106
[OMG00], 61, 236
[OMG01a], 196, 205
[OMG01b], 195, 236
[OMG01c], 61, 194, 236
[OMG01d], 60, 234
[OMG01e], 63
[OMG02], 205
[OMG99], 63
[OMIM94], 32
[PA94], 30
[RCRW99], 51, 81
[RDS03], 81
[RSL00], 81
[SCFJ96], 67
[SDW92], 69
[SK01], 35
[SSP+95], 51
[SYW99], 47
[Saa99], 23, 41
[Sau98], 4, 29, 115, 116
[TD01], 153
[Tra99], 23
[VDMKCS+99], 104, 105
[Ver99], 35
[W3C99], 184
[WAB+96], 41, 54
[WAS97], 58
[WKM94], 69
[WS], 81
[WW92], 23
[Wex93], 28
[Whi98], 81, 89
[X3D], 26
[YGS95], 71

Annexes

Annexe A

Le langage VRML

Le format VRML [BPP95][HW96][CBM97] s'est imposé depuis quelques années comme le format universel de description de données 3D sur Internet. La plupart des modeleurs 3D sont capables d'exporter les données 3D qu'ils produisent en VRML. La hiérarchie de la scène 3D et les différents objets qui la compose sont décrits en utilisant le langage VRML. Le codage ascii utilisé est à la fois un avantage pour la facilité d'édition, mais aussi un inconvénient pour la taille des fichiers générés. Cependant, les différents navigateurs VRML du marché supportent le format VRML compressé (fichier VRML ayant subi une compression *zip*). D'autres formats propriétaires offrent les mêmes fonctionnalités que le format VRML, le principal avantage étant d'offrir une taille de fichier beaucoup plus faible que VRML ascii et VRML zippé. On peut citer, entre autres, *o2c* (*Object to See*) de mb Software[52] ou *Metastream* de Viewpoint Corporation[53]. Cependant, leur utilisation reste très marginale.

Nous allons maintenant présenter le système VRML et plus particulièrement la deuxième version de la norme (VRML 97), ses concepts et ses fonctionnalités.

A.1 Historique

En 1993, suite à la présentation des travaux sur les interfaces de Réalité Virtuelle de M. Pesce et T. Parisi devant le consortium du World Wide Web (W3C), il est apparu nécessaire de définir un langage commun permettant de définir des scènes 3D. Le format proposé doit répondre à certaines contraintes : le format doit être indépendant de la plate-forme (Windows, Unix, etc), il doit être facilement extensible et il doit fonctionner avec une faible bande passante (modems).

Les choix se sont rapidement portés sur le format ascii de *Open Inventor*, un format développé par Silicon Graphics Inc. permettant la description de scènes 3D complètes avec rendu des objets sous forme de polygones, le traitement de l'éclairage, l'application d'une texture à un objet, etc. Le choix d'un format ascii peut paraître contradictoire avec la contrainte du passage sur des lignes bas débit, cependant il ne nécessite pas d'éditeur spécifique et est directement et facilement compréhensible. De plus, le format ascii est dans la politique des différents langages proposés par le W3C (comme par exemple, HTML). En 1995, la norme Virtual Reality Modelling Language 1.0 (VRML) [BPP95] a été adoptée. Le format VRML 1.0 a connu une large diffusion, et de nombreux outils sont disponibles pour le développement.

Rapidement pourtant, le langage VRML 1.0 montre ses limites : la scène 3D décrite est complètement statique et l'utilisateur ne peut pas interagir avec le monde. En février 1996, le consortium VRML entreprend de faire évoluer la norme VRML 1.0 pour prendre en compte la dynamique des mondes virtuels. En août 1996, les spécifications définitives de la norme VRML 2.0 sont publiées [HW96]. En 1997, la norme VRML 2.0 a fait l'objet d'une procédure de normali-

52. `http://www.o2c.de`

53. `http://www.viewpoint.com`

sation ISO [CBM97] (norme VRML 97, ISO/IEC n°14772-1:1997).

Cette deuxième version de la norme prend en compte l'animation de la scène, la gestion de l'interaction utilisateur, améliore le réalisme du monde 3D et ajoute la notion de prototypage.

Pour visualiser une scène VRML (1.0 ou 2.0) et pouvoir explorer le monde virtuel, il suffit d'un navigateur web auquel on a ajouté un plug-in capable d'interpréter la scène et de l'afficher dans le navigateur web même. Sur le marché, on trouve différents navigateurs VRML. On peut entre autres citer CosmoPlayer de SGI, Blaxxun Contact de Blaxxun, et Cortona de ParallelGraphics.

A.2 Description hiérarchique du modèle 3D

Le langage VRML permet de décrire la géométrie 3D du monde sous la forme d'un arbre. L'arbre est composé de *nœuds*, on peut distinguer deux types de nœuds : premièrement des nœuds permettant de créer la hiérarchie 3D (ce que l'on appelle des *grouping node*, comme par exemple les nœuds `Group` ou `Transform`), deuxièmement, des nœuds permettant de spécifier le rendu (sonore, visuel, géométrique). Le langage propose différents nœuds permettant de décrire des formes géométriques de base, comme par exemple un cube, une sphère, un cylindre, etc. Pour créer des objets plus complexes, VRML 97 offre également la possibilité de décrire les données 3D sous une forme "facettisée". VRML gère également des effets de rendu de plus haut niveau tels que le plaquage de textures, les sources lumineuses, etc. L'arbre créé à l'aide de ces différents nœuds est couramment appelé "graphe de scène" [HW96].

Chaque nœud expose un certain nombre de propriétés (appelées "champs" dans la norme VRML), ces propriétés permettent de configurer le comportement du nœud. Par exemple, le nœud `Box` (une boîte 3D) dispose d'une propriété *size* permettant de spécifier la taille de la boîte 3D.

Les figures A.1 et A.2 montrent des exemples de contenu VRML (1.0 et 2.0) et l'arbre associé.

A.3 Animation et Interaction

Comme nous l'avons vu précédemment, chaque noeud VRML 97 expose un certain nombre de propriétés. Ces différentes propriétés permettent au concepteur de spécifier le comportement du nœud, mais elles lui permettent également de connecter différents nœuds entre eux. Ces propriétés peuvent être de trois types :

- `EventIn`, la propriété est en entrée : elle est utilisée pour recevoir des informations de la part d'autres nœuds,
- `EventOut`, la propriété est en sortie : lorsqu'elle est modifiée, le nœud peut prévenir d'autres nœuds,
- `ExposedField`, la propriété est à la fois `EventIn` et `EventOut`.

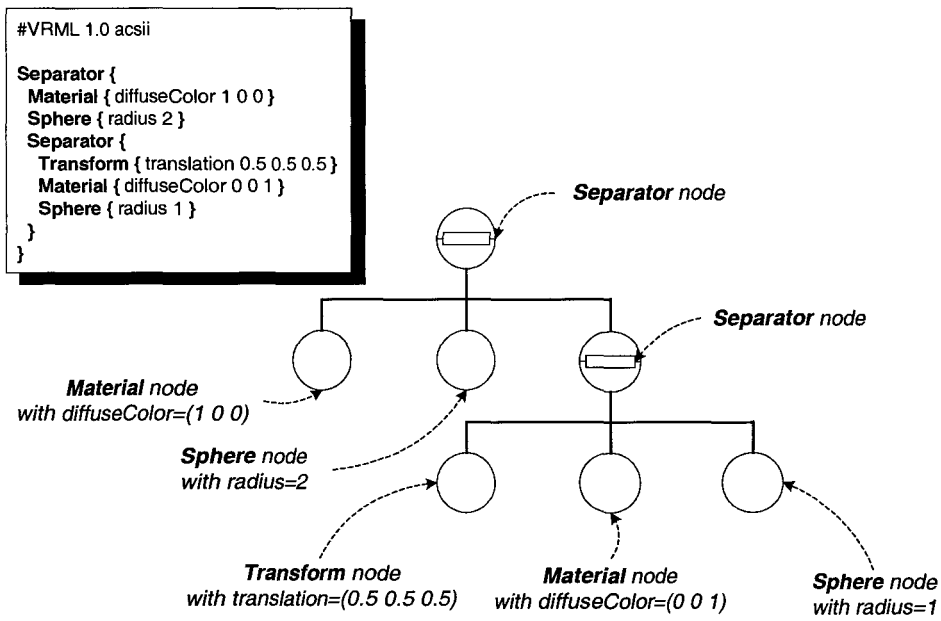


FIG. A.1 – *Exemple de contenu VRML 1.0 et graphe de scène associé*

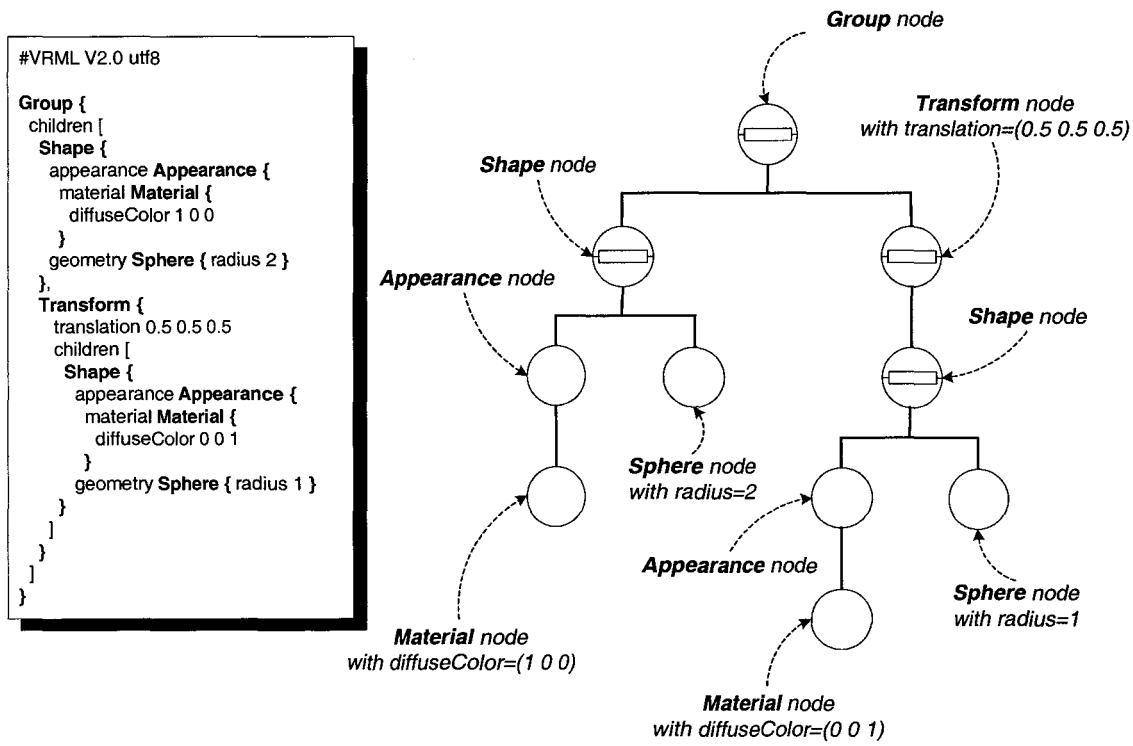


FIG. A.2 – *Exemple de contenu VRML 97 et graphe de scène associé*

La dynamique du monde 3D est obtenue en connectant des `EventOut` à des `EventIn` (de même type) en utilisant le mécanisme de `ROUTE`. L'évolution du monde produit des changements de valeurs (par exemple à l'aide de timers) qui sont répercutés grâce à l'exposition des `EventOut`. Tous les `EventIn` connectés à ces `EventOut` sont prévenus du changement de valeurs, ce qui déclenche alors une cascade d'évènements.

Grâce à la technique du *keyframing*, le concepteur peut spécifier la dynamique de son monde 3D. À la description géométrique de la scène, le concepteur ajoute des noeuds permettant de gérer l'animation : comme le montre la figure A.3, il connecte, à la l'aide du mécanisme de `ROUTE`, les sorties de *timers* (`TimeSensor` générant un top d'horloge à intervalle régulier) aux entrées de différents interpolateurs (`ColorInterpolator`, `PositionInterpolator`, `OrientationInterpolator`, etc). À l'aide de valeurs clés, l'interpolateur génère une valeur correspondant à la valeur courante du *timer*. Cette valeur de sortie peut être redirigée vers d'autres nœuds du monde. Le concepteur a ainsi créé une "chemin d'animation" [HW96].

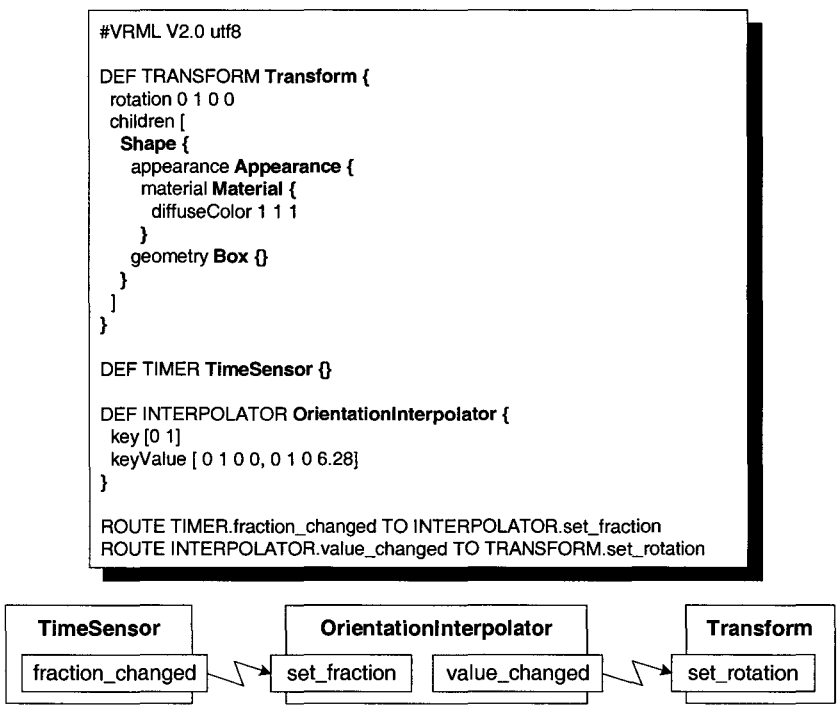


FIG. A.3 – *Exemple d'utilisation des interpolateurs*

Le concepteur peut également ajouter au graphe de scène différents noeuds (appelés `Sensors` dans la norme) permettant de gérer l'interaction avec l'utilisateur. Ces différents noeuds permettent de capter ce qui se passe dans le monde 3D, chaque noeud possède plusieurs propriétés (en sortie uniquement) de façon à réagir aux actions de l'utilisateur (cf. figure A.4).

Pour gérer des comportements plus complexes, VRML 97 permet d'inclure dans le graphe de scène des scripts. Tout comme les autres nœuds natifs du langage, le nœud `Script` expose différentes propriétés (définies par le concepteur) qui peuvent être connectées à d'autres nœuds

pour créer des chemins d'animation (cf. figure A.4). Un nœud `Script` contient du code compilé Java ou bien du code interprété écrit en ECMAScript (dérivé du langage JavaScript). Les scripts interagissent avec le contenu VRML 97 en utilisant la Browser Script Interface (BSI) [HW96] : cette interface permet, entre autre, aux scripts de retrouver des nœuds nommés explicitement (ie. possédant un DEF), de modifier les `ROUTES` existantes ou encore de modifier le contenu du navigateur VRML 97.

A.4 L'External Authoring Interface

A.4.1 Généralités

Comme nous venons de le voir, une première approche pour permettre à des programmes d'interagir avec la scène VRML 97 est l'utilisation de l'interface de scripting fournie par le navigateur (BSI, *Browser Script Interface*). Les scripts sont embarqués au sein même du graphe de scène et interagissent avec les autres nœuds en utilisant le mécanisme de `ROUTE`. Très souvent, les scripts embarqués dans le graphe de scène permettent de spécifier les comportements simples des objets contenus dans le monde.

Pour développer des comportements plus complexes, la norme VRML 97 propose l'External Authoring Interface (EAI). L'EAI est une API permettant à des programmes externes (i.e. situés hors du navigateur VRML 97) d'interagir avec le contenu du navigateur VRML 97. La spécification de l'EAI ne fait pas partie intégrante de la norme. En effet, c'est seulement une annexe. Cela signifie que les développeurs de navigateurs ont le choix de l'implémenter ou non. Néanmoins, de nombreux navigateurs VRML 97 comme Cosmoplayer ou Blaxxun Contact proposent cette fonctionnalité.

L'EAI définit un ensemble de services de haut niveau permettant à l'application externe d'accéder aux nœuds, à leurs propriétés et aux évènements de la scène VRML. La spécification de l'EAI décrit le comportement de ces services mais reste neutre quand à leur implémentation : aucun langage cible (ie. le langage utilisé pour le développement d'applications externes) n'est spécifié, une entière liberté est laissée aux développeurs. Néanmoins dans la pratique, tous les navigateurs VRML 97 supportent Java comme langage pour l'implémentation des applications externes. L'EAI est uniquement utilisée pour la communication entre une scène VRML et des applications externes (applets Java) incluses dans la même page HTML et s'exécutant sur la machine, dans le contexte du navigateur HTML. Cela limite grandement les possibilités. La raison est assez simple et purement liée à des problèmes techniques : les applications externes doivent pouvoir communiquer avec le navigateur VRML 97. Cela se fait par l'intermédiaire du navigateur web qui propose une API (*LiveConnect* pour Netscape, *ActiveX/COM* pour Internet Explorer) permettant la communication entre les applets Java et le navigateur VRML 97 embarqué dans le navigateur web (cf. figure A.5).

```
#VRML V2.0 utf8

DEF TRANSFORM Transform {
 rotation 0 1 0 0
 children [
  DEF SENSOR TouchSensor {}
  Shape {
   appearance Appearance {
    material Material {
     diffuseColor 1 1 1
    }
   geometry Box {}
  }
 ]
}

DEF TIMER TimeSensor { stopTime 1 }

DEF INTERPOLATOR OrientationInterpolator {
 key [0 1]
 keyValue [ 0 1 0 0, 0 1 0 6.28]
}

DEF SCRIPT Script {
 EventIn SFTime touch
 EventOut SFTime start
 EventOut SFTime stop
 Field SFBool isRunning FALSE
 url "javascript:
      function touch(value) {
       if (!isRunning) {
        start = value;
        isRunning = TRUE;
       } else {
        stop = value;
        isRunning = FALSE;
       }
      }"
}

ROUTE SENSOR.touchTime TO SCRIPT.touch
ROUTE SCRIPT.start TO TIMER.startTime
ROUTE SCRIPT.stop TO TIMER.stopTime
ROUTE TIMER.fraction_changed TO INTERPOLATOR.set_fraction
ROUTE INTERPOLATOR.value_changed TO TRANSFORM.set_rotation
```

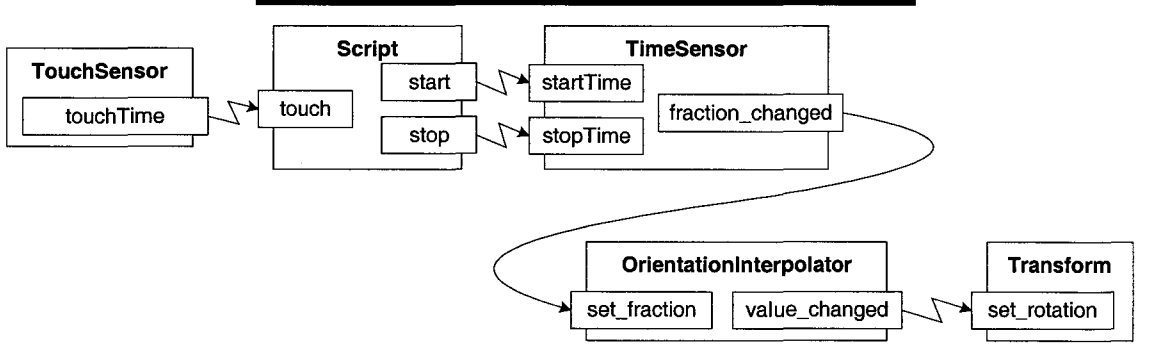


FIG. A.4 – *Exemple d'utilisation des sensors d'interaction, des scripts et des interpolateurs*

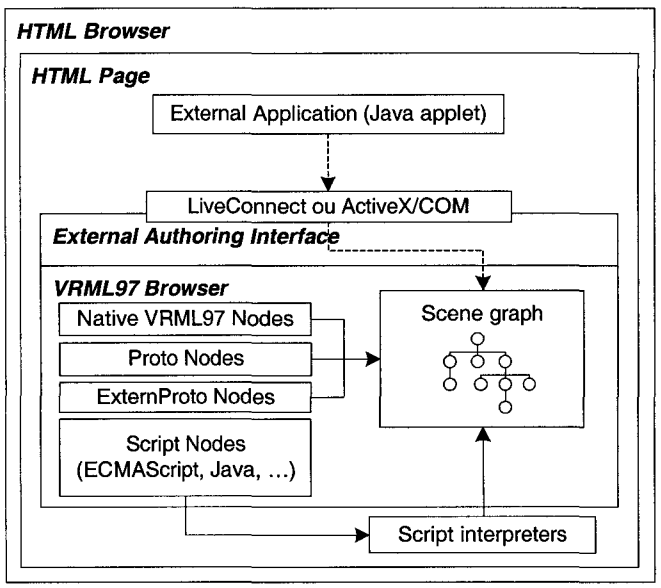


FIG. A.5 – *Architecture d'un navigateur VRML 97*

A.4.2 Exemples de code Java

Lors du développement d'une applet Java, la première opération à effectuer est d'établir une connexion vers un browser VRML. Cela s'effectue en récupérant la référence de l'objet `Browser` (cf. figure A.6). L'interface `Browser` offre un ensemble de methodes permettant d'accéder au graphe de scène VRML 97 contenu dans le navigateur :

- des méthodes pour manipuler le graphe de scène (ajout, retrait de nœuds, chargement d'une autre scène, etc.),
- des méthodes pour accéder aux nœuds contenus dans le graphe de scène.

```
// some modules import
import vrml.eai.*;
import vrml.eai.field.*;

public class VRMLApplet extends Applet {
 Browser browser;
 public void start () {
   ...
  browser = (Browser) Browser.getBrowser(this);
 }
}
```

FIG. A.6 – *Exemple de code EAI/Java : connexion vers le navigateur VRML 97*

Avant de pouvoir manipuler un nœud, l'applet Java doit récupérer sa référence. Cela s'effectue en utilisant la méthode `getNode()` de l'interface `Browser` (cf. figure A.7). Il est à noter que l'on peut uniquement récupérer les nœuds nommés du graphe de scène, ie. ceux possédant un `DEF`. On peut également manipuler les nœuds créés à la volée à partir du moment où l'application conserve la(les) référence(s) retournée(s) lors de la création. Une fois obtenue la référence sur

un nœud, le programmeur peut le manipuler, ie. écrire dans un de ses champs `EventIn`, obtenir la valeur de ses champs `EventOut` et enregistrer des observateurs sur ses champs (`EventIn` ou `EventOut`) (cf. figure A.8). Un observateur de champ permet à l'application d'être notifiée des changements de valeurs du champ.

```
Node transform;
try {
 transform = (Node) browser.getNode("TheTransform");
} catch (InvalidNodeException e) { ... }
```

FIG. A.7 – *Exemple de code EAI/Java : obtention de la référence d'un nœud possédant un DEF*

```
EventInSFRotation rotation;
float[] r = {0, 1, 0, 3.14};
try {
 rotation = (EventInSFRotation)
            transform.getEventIn("rotation");
 rotation.setValue(r);
} catch (InvalidEventInException e) { ... }
```

FIG. A.8 – *Exemple de code EAI/Java : écriture dans un champs*

Annexe B

L'intergiciel CORBA

CORBA (*Common Object Request Broker Architecture*) [OMG01d] est une norme de l'Object Management Group (OMG), un consortium regroupant plus de 800 membres de l'industrie de l'informatique : des fournisseurs de matériels comme Sun, HP, IBM ; des fournisseurs de logiciels comme Microsoft ; des grands utilisateurs comme Boeing, Alcatel ; des laboratoires de recherche dont le LIFL. L'OMG vise à promouvoir les technologies orientées objets pour la conception d'applications informatiques distribuées, interopérables et ouvertes. CORBA est l'acronyme de *Common Object Request Broker Architecture*, que l'on peut traduire en français de la façon suivante *Architecture Standardisée d'un Négociateur de Requêtes sur des Objets*. CORBA a pour but de masquer la distribution des objets : l'objectif est de permettre l'accès à des objets comme s'ils étaient présents localement, c'est le paradigme de la programmation distribuée orientée objet.

B.1 Les composants logicielles de l'architecture CORBA

La norme définit les différents composants logiciels permettant d'automatiser les tâches spécifiques à la distribution des objets sur le réseau : l'enregistrement des objets, leur localisation et leur activation ; l'encodage et le décodages des requêtes, des paramètres et des valeurs de retour ; l'invocation des méthodes. Pour cela, la norme CORBA comprend les composants logiciels suivants (cf. figure B.1) :

- l'ORB (*Object Request Broker*) : c'est l'entité principale par laquelle toutes les requêtes vont transiter. L'ORB est responsable de localiser un objet sur lequel l'on souhaite invoquer une méthode, de lui passer les paramètres, puis d'invoquer la méthode, et enfin de retourner le résultat.
- le langage OMG-IDL (*Interface Definition Language*) : c'est le langage permettant de spécifier l'interface des objets, ie. de décrire les méthodes, et leurs paramètres, que l'on va pouvoir invoquer. Un compilateur IDL génère à partir de la spécification OMG-IDL (appelée aussi "contrat IDL") des souches (*stubs*) pour le coté client et des squelettes (*skeletons*) pour le coté serveur dans le langage de son choix (C, C++, Java, etc). En conjonction avec l'ORB, les souches et les squelettes automatisent le codage/décodage des paramètres, l'enregistrement des objets, leur activation et leur localisation.
- le protocole GIOP (*General Inter-ORB Protocol*) : c'est le protocole de haut niveau permettant de coder les messages qui couvrent la sémantique des échanges requête-reponse (demande d'invocation, message résultat, etc). La CDR (*Common Data Representation*) permet de transcrire n'importe quel type de données défini en OMG-IDL en une représentation réseau. L'IOR (*Interoperable Object Reference*) permet de localiser les objets. Le protocole GIOP est conçu de manière à pouvoir fonctionner directement au dessus de n'importe quel protocole de transport. IIOP (*Internet Inter-ORB Protocol*) est une déclinaison de GIOP sur TCP/IP. Il spécifie comment les messages GIOP sont échangés sur les réseaux

TCP/IP.

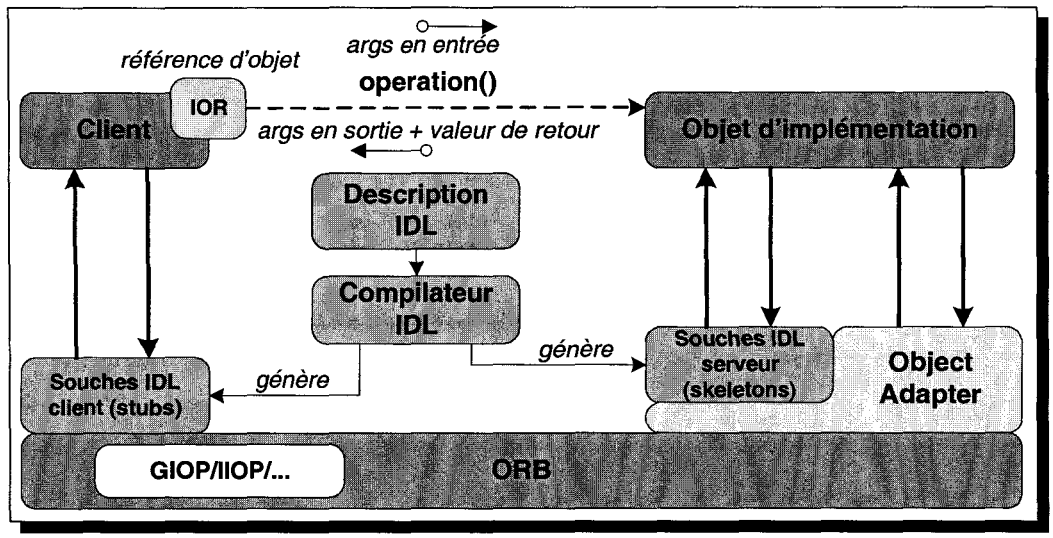


FIG. B.1 – *Architecture de l'infrastructure CORBA*

B.2 Développement d'applications distribuées

CORBA est un intergiciel "multi-plateformes" et "multi-langages'. Le langage OMG-IDL peut être projeté dans différents langages comme C, C++, Java, etc. Les clients et les serveurs peuvent être écrits dans des langages différents et être exécutés sur des plate-formes différentes. La norme CORBA et ses différents composants logiciels assurent *l'interoperabilité* : des clients et des serveurs conçus à partir de différentes implémentations d'ORB sont capables d'interagir entre eux.

Le fonctionnement général d'une invocation de méthode suit le schéma suivant (cf. figure B.1) :

1. le client appelle la méthode via le stub,
2. l'adaptateur d'objet (Object Adapter) fournit une abstraction de la notion d'objets d'implémentation et permet de faire le lien entre l'ORB et les objets d'implémentation. L'OA passe la requête d'invocation de la méthode à l'implémentation via le squelette,
3. l'implémentation retourne alors le résultat ou l'exception au client en utilisant le squelette, l'OA et l'ORB.

Classiquement, pour développer une application, le développeur choisit son langage d'implémentation, génère les souches et les squelettes et implémente son application cliente et serveur. Cela requiert une certaine connaissance des principes de base de CORBA et de bonnes connaissances en programmation. Pour simplifier le processus de conception, il est également possible de concevoir une application sans aucune phase de compilation en utilisant un langage de script adapté

à l'environnement CORBA. Pour cela, l'OMG a défini IDLScript [OMG01b] et une de ses implémentations CORBAScript [54]. Ce langage permet d'appliquer des invocations de méthodes sur n'importe quel objet CORBA. IDLScript facilite le développement de l'application : toutes les opérations complexes (comme l'initialisation de l'ORB ou les opérations de *cast*) sont effectuées par l'interpréteur de script.

B.3 Définition des services CORBA

En complément de la norme CORBA, l'OMG a spécifié également un ensemble de services, un service étant caractérisé par les interfaces qu'il fournit et par les objets qui fournissent ces interfaces. Ils proposent des outils pour faciliter l'utilisation courante des objets CORBA, comme par exemple, le service de nommage, le service d'évènements, le service vendeur, etc.

Le service de nommage [OMG01c] fournit un système de désignation pour retrouver les références d'objets (IOR) à partir de noms symboliques. Il peut être comparé à un annuaire de références où les applications serveurs peuvent enregistrer des références d'objets sous des noms symboliques, et où les applications clientes peuvent obtenir des références d'objets à partir des noms symboliques.

Le protocole GIOP n'a pas été conçu pour le transport de données temps réel (comme par exemple des données vidéo ou sonores) : en effet, l'encodage n'est pas assez efficace, des informations supplémentaires (IOR, nom de la méthode à invoquer, paramètres, etc.) doivent être ajoutées augmentant fortement la taille des paquets. Une solution est d'utiliser le service de flux multimedia [MSS99][OMG00] (norme *Audio/Video Streams*) défini par l'OMG. C'est un framework définissant un ensemble de services de haut niveau pour la mise en place et le contrôle de flux multimédia entre des producteurs et des consommateurs. La norme définit un ensemble d'interfaces de contrôle.

Comme le montre la figure B.2, un flux (*Stream*) est composé de plusieurs flots élémentaires (*Flows*). Dans la terminologie de la norme *Audio/Video Streams*, les terminaisons des flux s'appellent des `StreamEndpoints` et les terminaisons de flots élémentaires sont des `FlowEndpoints`. La norme définit également, via l'interface `MMDevice`, une abstraction des périphériques multimédia producteurs et consommateurs de données. Chaque périphérique multimédia regroupe plusieurs entités productrices ou consommatrices de flots (`FlowDevice`). Un dernier élément clé de la norme est la définition d'un service centralisé permettant de mettre en relation les producteurs et les consommateurs de flux (`MMDevices`), c'est l'interface du contrôleur de flux (`StreamCtrl`).

La figure B.3 présente le protocole de mise en relation. Le protocole tel qu'il est proposé dans la norme permet uniquement la mise en relation de deux périphériques multimédia. En effet, au cours de la mise en relation de deux périphériques multimédia (comme par exemple, une caméra avec un écran), un objet `VDev` est créé et permet aux différents `MMDevices` de configurer

54. `http://corbaweb.lifl.fr/CorbaScript/`

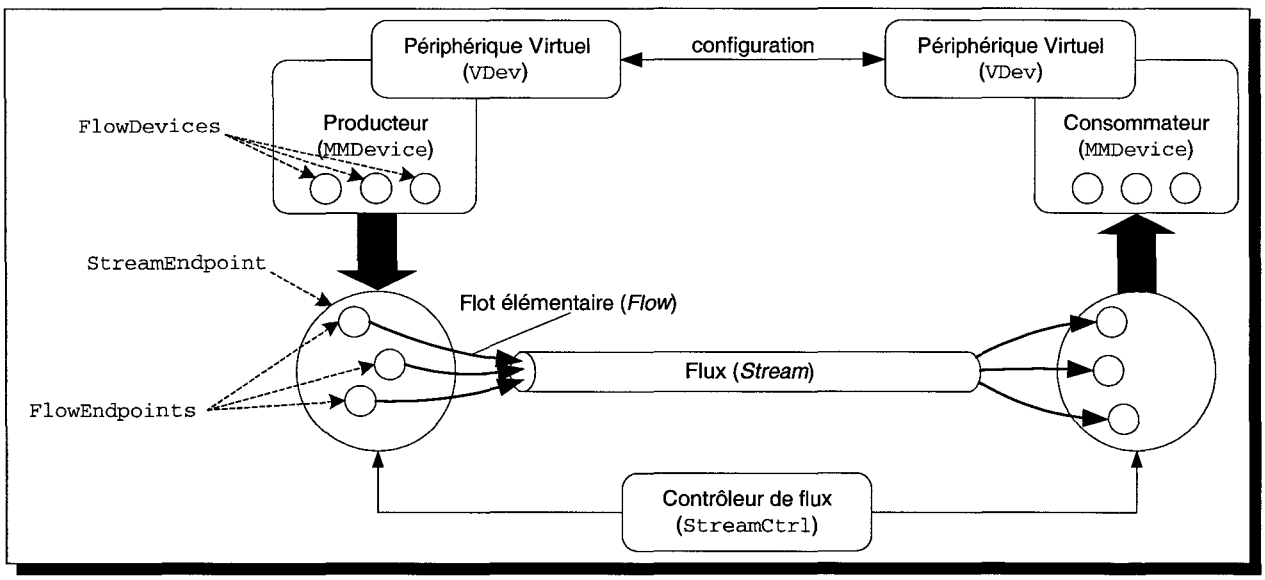


FIG. B.2 – *Terminologie des éléments de la norme Audio/Video Streams*

leurs entrées et sorties (comme par exemple, définir le format des données vidéos et sonores, le débit, etc.). De même, un objet `StreamEndpoint` est également créé permettant d'établir le flux entre les deux `MMDevices`. Le flux `StreamEndpoint` est composé de plusieurs de flots élémentaires (`FlowEndpoints`), chacun d'entre eux étant associé à un `FlowDevice` contenu dans le `MMDevice`. L'objet `StreamCtrl` est en charge de contrôler le flux établi entre les deux parties (démarrage, arrêt et destruction), ie. les deux `StreamEndpoint` créés par chacune des parties lors de la mise en relation. Ceci définit un premier profil de contrôle : uniquement les `StreamEndpoints` peuvent être contrôlés en utilisant l'interface du `StreamCtrl`. Néanmoins, les flots élémentaires (`FlowEndpoints`) peuvent être indirectement contrôlés via l'interface du flux (`StreamEndpoint`) auquel ils appartiennent en utilisant la notion de spécification de flot (`FlowSpec`) : une spécification de flot est une "adresse" assimilable à une URL qui est associée à un `FlowEndpoint`. Cette pseudo URL contient différentes informations comme le nom du flot associé, le protocole de transport utilisé, le type de données, etc. Pour plus de flexibilité dans le contrôle des flux, la norme définit un deuxième profil de contrôle autorisant le contrôle à la fois des `StreamEndpoints` ainsi que des `FlowEndpoints`. Ainsi avec ce deuxième profil, il est possible de connecter directement deux flots élémentaires ensemble.

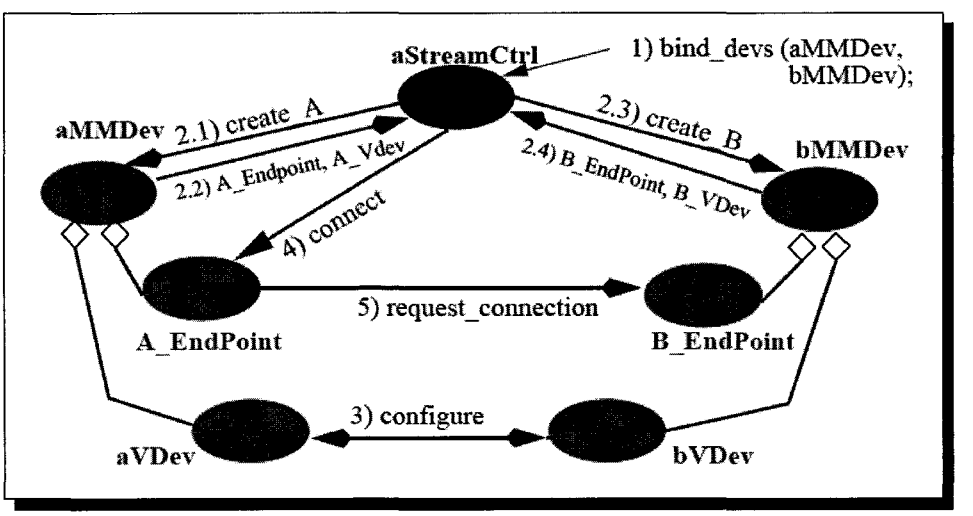


FIG. B.3 – *Protocole de mise en relation de deux* `MMDevices` *[OMG00]*

Annexe C

Partage d'un Sensor avec le paradigme d'insertion

Avec le paradigme d'insertion (cf. 3.1.1.1.1, page 78), il n'y a aucune cohérence entre les différents postes au niveau des capteurs de manipulation (`Sensors`). Étant les initiateurs de tout chemin d'animation, leurs sorties (`EventOut`) ne peuvent être synchronisées. L'état de chaque capteur de manipulation n'est donc pas cohérent avec l'état des variables qui lui sont reliées via le mécanisme de routes. Seul le poste à l'origine de la modification est cohérent ; sur les autres postes, cela va introduire des problèmes de compréhension de la part de l'utilisateur lors des futures manipulations (un saut brusque lors de l'animation pour retrouver un état cohérent entre la valeur de sortie du capteur et les variables qui lui sont connectées via le mécanisme de route).

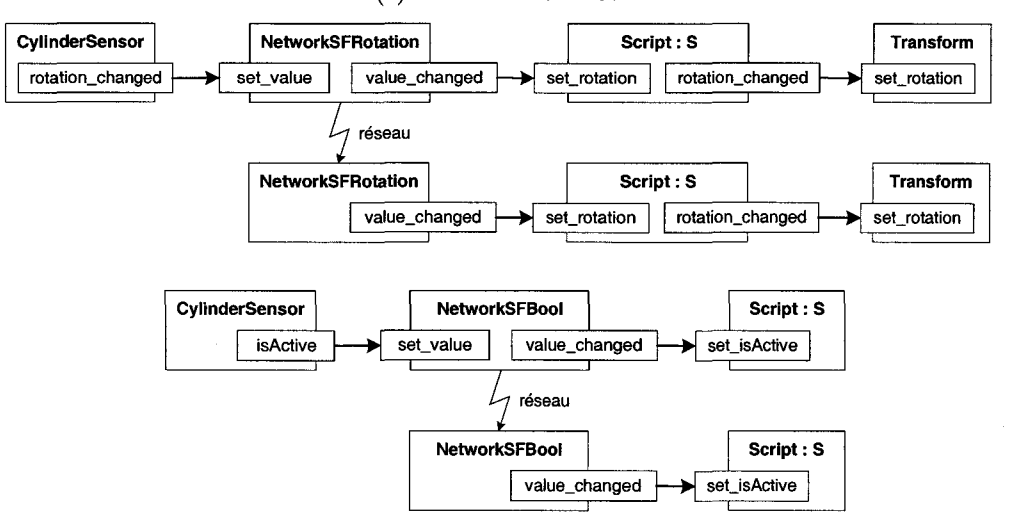
Dans cette annexe, nous présentons une solution technique permettant de compenser le manque de cohérence entre les copies d'un même `Sensor` inhérent à l'utilisation du paradigme d'insertion. Cette solution, utilisant la syntaxe proposée par Living Worlds, est présentée sur la figure C.1 :

- le Sensor doit être placé en mode relatif en plaçant le champ `autoOffset` à `FALSE`,
- un nœud Script est en charge de calculer le décalage que calculait automatiquement le `Sensor`. Pour cela, quand l'utilisateur arrête de manipuler (le champ `isActive` passe de `TRUE` à `FALSE`, le script récupère la dernière valeur émise par le `Sensor` ; cette valeur sera ajoutée aux prochaines valeurs générées par le `Sensor` lors de sa prochaine manipulation.
- des nœuds de partage sont insérés entre les nœuds `Sensor` et `Script`. Un nœud de partage permet de propager à distance les valeurs générées par le `Sensor` et utilisées pour l'animation. Un nœud de partage permet de propager à distance l'arrêt de l'état du `Sensor` local, cela permet aux scripts distants de déterminer le moment où ils doivent récupérer la valeur du décalage (ie. quand l'utilisateur relâche le `Sensor`).

Avec cette solution, on maintient cohérent l'état du nœud `Script` et, par conséquent, il n'y aura pas de brusques sauts lors de la manipulation d'un site à l'autre du même objet : la manipulation reprendra à partir de l'état courant contenu dans le nœud `Script` qui est synchronisé sur les différents postes. C'est une solution qui reste lourde à mettre en œuvre et nécessite de bonnes connaissances en VRML 97.

```
#VRML V2.0 utf8

# Some external definitions
EXTERNPROTO ...

# World description
DEF SO SharedObject {
 private PrivateSharedObject {
  state [
   DEF R NetworkSFRotation {},
   DEF C NetworkSFBool{}
  ]
  visibleDefinition DEF T Transform {
   children [
    DEF CS CylinderSensor { autoOffset FALSE },
    Shape {
     appearance Appearance { material Material { diffuseColor 1 0 0 }}
     geometry Box { size 1 1 1 }
    }
   ]
  }
}}

DEF S Script {
 eventIn SFRotation set_rotation
 eventIn SFBool set_isActive
 eventOut SFRotation rotation_changed
 field SFFloat angle 0.0
 url "javascript:
  function set_isActive(value) {
   if (!value)
    angle = rotation_changed[3]
  }
  function set_rotation(value) {
   rotation_changed = value
   rotation_changed[3] += angle
  }"
}

# Routes description
ROUTE CS.rotation_changed TO R.set_value
ROUTE CS.isActive TO C.set_isActive
ROUTE R.value_changed TO S.set_rotation
ROUTE S.rotation_changed TO T.rotation
ROUTE C.value_changed TO S.set_isActive
```

(a) le fichier VRML 97



(b) les chemins d'animation

FIG. C.1 – *Une solution pour résoudre les problèmes de cohérence des sensors en utilisant le paradigme d'insertion*

Annexe D

Un nouveau Sensor VRML 97 pour l'interaction 3D

Dans l'annexe A, nous avons vu que le langage VRML 97 propose différents noeuds `Sensor` (`CylinderSensor`, `SphereSensor`, etc.) permettant de capter les actions de l'utilisateur sur différents objets présents dans la scène VRML 97. Cependant, ces différents sensors sont adaptés à une interaction en 2D avec un dispositif de pointage 2D classique comme la souris. Or, dans le cadre de notre activité, nous proposons à l'utilisateur d'utiliser des périphériques 3D pour interagir avec les objets de l'interface 3D. C'est pourquoi, notre terminal de travail Spin-3D propose un nouveau type de `Sensor` VRML qui est plus adapté à une interaction en 3D bi-manuelle. Ce nouveau sensor permet de gérer deux périphériques virtuels :

- un périphérique de pointage à trois degrés de liberté permettant de sélectionner les objets, le sensor est alors capable de fournir la position du pointeur ainsi que l'état du bouton de sélection,
- un périphérique de manipulation à six degrés de liberté, le sensor est capable de fournir la valeur de la translation et de la rotation du périphérique.

Le navigateur *mappe* les périphériques virtuels sur des dispositifs physiques, par exemple l'*OWL* comme périphérique de désignation et la *SpaceMouse* comme périphérique de manipulation, ou bien encore la souris (désignation) et le clavier (manipulation) pour un terminal dépourvu de périphériques 3D.

Comme les autres extensions au langage VRML 97 que nous proposons, ce nouveau noeud est également intégré à notre navigateur VRML 97 étendu. La figure D.1 présente la description VRML 97 du nœud `Sensor3D`.

```
Sensor3D {
  enabled TRUE                  # exposedField SFBool
  relative TRUE                 # field SFBool
  raw FALSE                     # field SFBool
  position_changed              # eventOut SFVec3f
  translation_changed           # eventOut SFVec3f
  rotation_changed              # eventOut SFRotation
  eulerRotation_changed         # eventOut SFVec3f
  rotationThreshold 0 0 0       # field SFVec3f
  translationThreshold 0 0 0    # field SFVec3f
  rotationIncrement 0 0 0       # field SFVec3f
  translationIncrement 0 0 0    # field SFVec3f
  dirX TRUE                     # field SFBool
  dirY TRUE                     # field SFBool
  dirZ TRUE                     # field SFBool
  axisOx TRUE                   # field SFBool
  axisOy TRUE                   # field SFBool
  axisOz TRUE                   # field SFBool
  minX -999999.0                # field SFFloat
  maxX  999999.0                # field SFFloat
  minY -999999.0                # field SFFloat
  maxY  999999.0                # field SFFloat
  minZ -999999.0                # field SFFloat
  maxZ  999999.0                # field SFFloat
  minOx -999999.0               # field SFFloat
  maxOx  999999.0               # field SFFloat
  minOy -999999.0               # field SFFloat
  maxOy  999999.0               # field SFFloat
  minOz -999999.0               # field SFFloat
  maxOz  999999.0               # field SFFloat
  button FALSE                  # eventOut SFBool
  isActive FALSE                # eventOut SFBool
  lockName ""                   # SFString
}
```

FIG. D.1 – *Prototype VRML 97 du nœud* `Sensor3D`

Nom	Description
enabled	indique si le sensor prend en compte le périphérique de pointage, ie. s'il peut être activé
lockName	indique le nom du jeton à acquérir pour éviter les accès concurrents
relative	indique si les coordonnées du périphérique de pointage doivent être considérées dans le repère local de l'objet sur lequel se place le sensor (TRUE) ou dans le repère absolu de la scène (FALSE)
position_changed	position du pointeur dans le repère courant de la géométrie associée ou dans le repère de la scène globale (cf. le champ `relative`)
button	indique l'état du bouton du périphérique de sélection, enfoncé (TRUE) ou relâché (FALSE)
raw	indique si le périphérique de manipulation fournit des données brutes (TRUE) ou cumulées dans le temps (FALSE)
translation_changed	données de translation fournies par le périphérique de manipulation
rotation_changed	données de torsion fournies par le périphérique de manipulation
eulerRotation_changed	données de torsion fournies par le périphérique de manipulation sous forme de composantes d'euler (ie. un angle par axe de rotation)
dominant	pour le périphérique manipulation, seules les données du degrés de liberté sur lequel l'utilisateur exerce l'action la plus marquée seront prise en compte
dirX	indique si la composante de translation suivant l'axe Ox doit être prise en compte (TRUE) ou ignorée (FALSE)
dirY	indique si la composante de translation suivant l'axe Oy doit être prise en compte (TRUE) ou ignorée (FALSE)
dirZ	indique si la composante de translation suivant l'axe Oz doit être prise en compte (TRUE) ou ignorée (FALSE)
axisOx	indique si l'angle de rotation autour l'axe Ox doit être prise en compte (TRUE) ou ignorée (FALSE)
axisOy	indique si l'angle de rotation autour l'axe Oy doit être prise en compte (TRUE) ou ignorée (FALSE)
axisOz	indique si l'angle de rotation autour l'axe Oz doit être pris en compte (TRUE) ou ignorée (FALSE)
minX	borne inférieure des valeurs de translation suivant l'axe Ox
minY	borne inférieure des valeurs de translation suivant l'axe Oy
minZ	borne inférieure des valeurs de translation suivant l'axe Oz
maxX	borne supérieure des valeurs de translation suivant l'axe Ox
maxY	borne supérieure des valeurs de translation suivant l'axe Oy
maxZ	borne supérieure des valeurs de translation suivant l'axe Oz

Nom	Description
minOx	borne inférieure des valeurs de rotation autour de l'axe Ox
minOy	borne inférieure des valeurs de rotation autour de l'axe Oy
minOz	borne inférieure des valeurs de rotation autour de l'axe Oz
maxOx	borne supérieure des valeurs de rotation autour de l'axe Ox
maxOy	borne supérieure des valeurs de rotation autour de l'axe Oy
maxOz	borne supérieure des valeurs de rotation autour de l'axe Oz
translationIncrement	définit des pas discrets dans la translation envoyée. La valeur 0 0 0 indique une translation continue
rotationIncrement	définit des pas discrets dans la rotation envoyée. La valeur 0 0 0 indique une rotation continue
translationThreshold	définit un seuil en deçà duquel les données de translation ne sont pas envoyées. La valeur 0 0 0 indique une translation continue
rotationThreshold	définit un seuil en deçà duquel les valeurs de rotation ne sont pas envoyées. La valeur 0 0 0 indique une rotation continue

La table précédente présente les différents champs disponibles sur un noeud `Sensor3D`. Il est à noter que cette table est divisée en trois parties distinctes :

- une première partie contenant les champs relatifs au fonctionnement général du `Sensor3D`,
- une deuxième partie contenant les champs relatifs au périphérique de sélection,
- une troisième partie contenant les champs relatifs au périphérique de manipulation.

Pour que les utilisateurs puissent manipuler les objets 3D présents dans l'interface de travail, le concepteur doit placer ce nouveau `Sensor` sur les différents objets. Nous allons présenter un exemple : c'est le modèle d'un appareil photographique sur lequel l'utilisateur peut manipuler trois bagues de réglage. La figure D.2 montre le source VRML 97 (simplifié) : trois noeuds `Sensor3D` ont été ajoutés à la hiérarchie géométrique permettant ainsi de manipuler en rotation les trois bagues de réglage. Pour chacune des bagues, on spécifie au niveau du `Sensor3D` l'axe de rotation, ainsi que l'effet souhaité lors de la rotation. L'utilisation du champ `rotationIncrement` permet d'obtenir un effet cranté (pour les bagues de vitesse et de diaphragme), et les champs `min0?` et `max0?` permettent de fixer les limites de rotation.

```
#VRML v2.0 utf8
EXTERNPROTO Sensor3D [
  ...
]["urn:inet:lifl.fr:proto/Sensor3D"]

...
DEF bague_vitesse Transform {
  children [
    DEF sensor_vitesse Sensor3D {
      axisOx    FALSE
      axisOy    TRUE
      axisOz    FALSE
      rotationIncrement        0 30 0
      minOy     -150.0
      maxOy     180.0
    }
    ...
  ]
}
...
DEF bague_diaphragme Transform {
  children [
    DEF sensor_diaphragme Sensor3D {
      axisOx    FALSE
      axisOy    TRUE
      axisOz    FALSE
      rotationIncrement        0 30 0
      minOy     -60.0
      maxOy     270.0
    }
    ...
  ]
}
...
DEF bague_focus Transform {
  children [
    DEF sensor_focus Sensor3D {
      axisOx    FALSE
      axisOz    FALSE
      axisOy    TRUE
      minOy     -270.0
      maxOy     90.0
    }
    ...
  ]
}
...

ROUTE sensor_vitesse.rotation_changed TO bague_vitesse.set_rotation
ROUTE sensor_diaphragme.rotation_changed TO bague_diaphragme.set_rotation
ROUTE sensor_focus.rotation_changed  TO bague_focus.set_rotation
```

FIG. D.2 – *Exemple de contenu VRML 97 intégrant le nœud* `Sensor3D`

Annexe E

Fonctionnement du plug-in RMIOP

E.1 L'interface de programmation fournie par Orbacus

Le bus CORBA Orbacus développé par IONA propose une interface de programmation, appelée Open Communications Interface (OCI) [Ionb]. L'OCI fournit un ensemble d'interfaces permettant de développer de nouveaux protocoles de communications. Ces interfaces permettent de transférer un *buffer* d'octets d'un processus émetteur à un ou plusieurs récepteurs. Orbacus encode le message en utilisant GIOP et fournit un *buffer* contenant le message GIOP à l'OCI qui est en charge de le transporter jusqu'au(x) récepteur(s). Le développement des interfaces de l'OCI en utilisant TCP/IP permet d'implémenter IIOP. On peut imaginer utiliser d'autres protocoles de transport (comme par exemple, AAL5 – *ATM Adaptation Layer 5* –) permettant ainsi d'implémenter de nouveaux *plug-ins* de communication pour le bus CORBA Orbacus.

Dans l'OCI, on trouve les interfaces suivantes :

- l'interface `Buffer` permet de définir un objet contenant un ensemble d'octets et un curseur permettant de déterminer le nombre d'octets qui ont déjà été envoyés et le nombre qu'il reste a transmettre,
- l'interface `Transport` permet d'envoyer et de recevoir un ensemble d'octets en utilisant l'interface `Buffer`,
- les interfaces `Acceptor` et `Connector` sont des fabriques pour les objets `Transport`. L'interface `Connector` est utilisée par le client pour se connecter au serveur. L'interface `Acceptor` est utilisée par le serveur pour accepter les connections des clients,
- les interfaces des "fabriques" des objets `Acceptor` et de `Connector`,
- les *registries* pour les fabriques d'`Acceptor` et de `Connector` sont utilisées pour *plugger* le nouveau protocole au niveau du bus CORBA Orbacus,
- les interfaces `Infos` permettent d'obtenir des informations sur les objets `Transport`, `Acceptor` et `Connector`.

La figure E.1 présente les différentes interfaces définies dans l'OCI (exceptées les interfaces `Buffer` et `Infos`). L'OCI fournit des interfaces abstraites (`Acceptor`, `Acceptor Factory`, `Connector`, `Connector Factory`, `Transport`,) qui sont à étendre par héritage par les concepteurs de plug-in de communication. L'OCI fournit également l'implémentation des interfaces `Buffer`, `Acceptor Factory Registry` et `Connector Factory Registry`. Les implémentations spécifiques de l'interface `Acceptor Factory` (resp. `Connector Factory`) doivent être enregistrées auprès de l'interface `Acceptor Factory Registry` (resp. `Connector Factory Registry`).

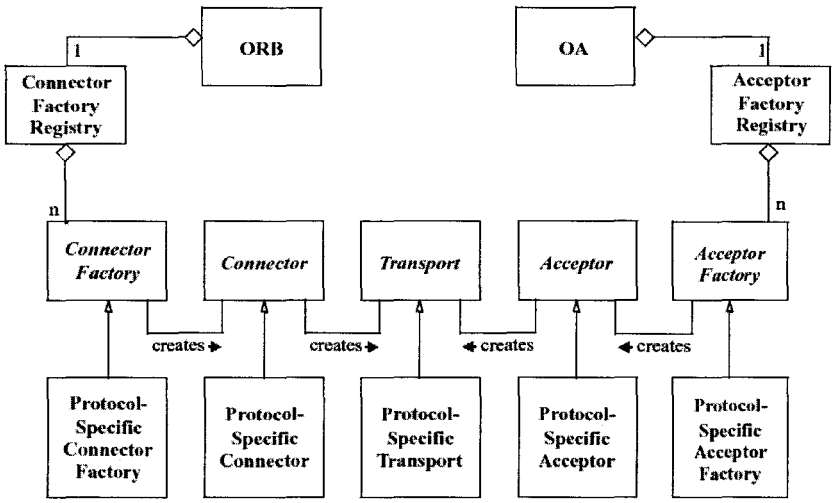


FIG. E.1 – *Diagramme de classes de l'OCI [Ionb]*

E.2 Schémas de fonctionnement du protocole RMIOP

Au cours de cette thèse, nous avons développé un plug-in de communication pour l'ORB Orbacus implémentant une version fiabilisée du protocole de communication de groupe MIOP. Pour cela, nous avons utilisé l'interface de programmation OCI.

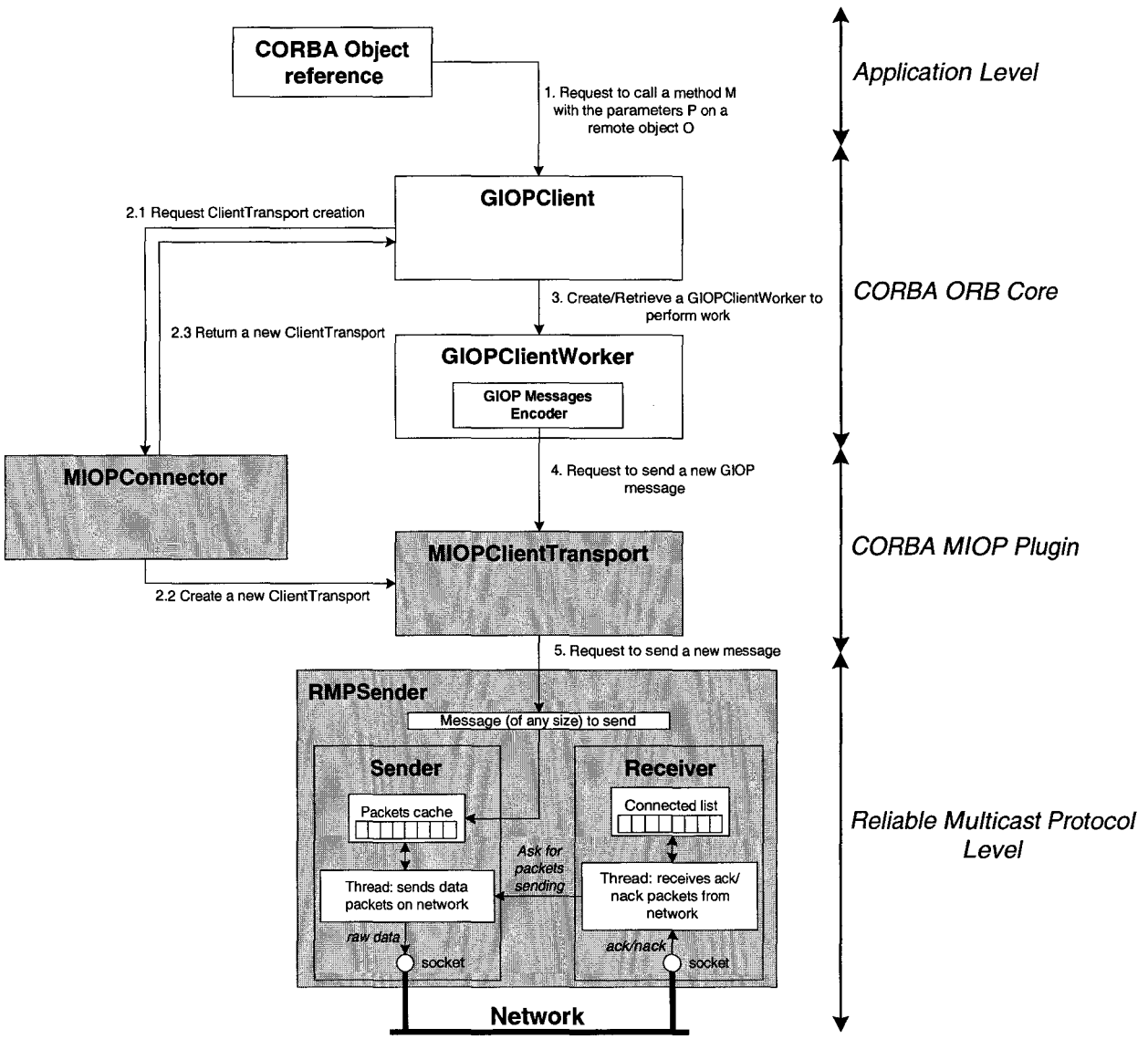


FIG. E.2 – *Fonctionnement du protocole RMIOP du coté du client (en grisé, les parties développées au LIFL)*

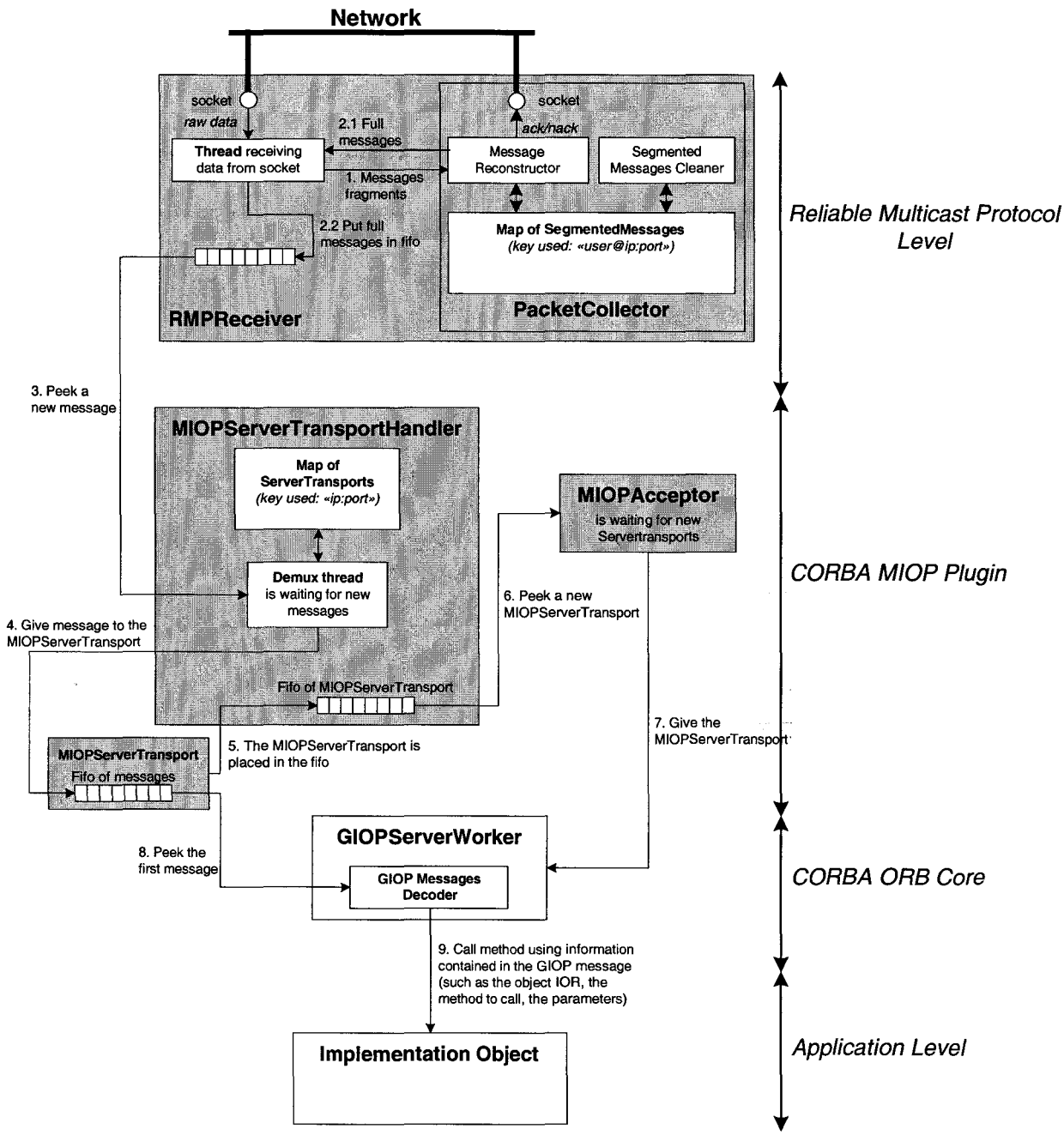


FIG. E.3 – *Fonctionnement du protocole RMIOP du coté du serveur (en grisé, les parties développées au LIFL)*

Annexe F

Automates du gestionnaire de jetons

Lors d'une demande d'acquisition de jeton, le terminal "demandeur" s'adresse aux gestionnaires de jetons local qui exécute alors l'automate présenté sur la figure F.1. Un dialogue s'établit avec les gestionnaires de jetons des terminaux de travail distants : à distance, lors de la réception d'une demande de jeton, l'automate présenté sur la figure F.2 est exécuté. Pour déverrouiller un jeton, l'automate de la figure F.3 est exécuté sur la machine possédant le jeton verrouillé.

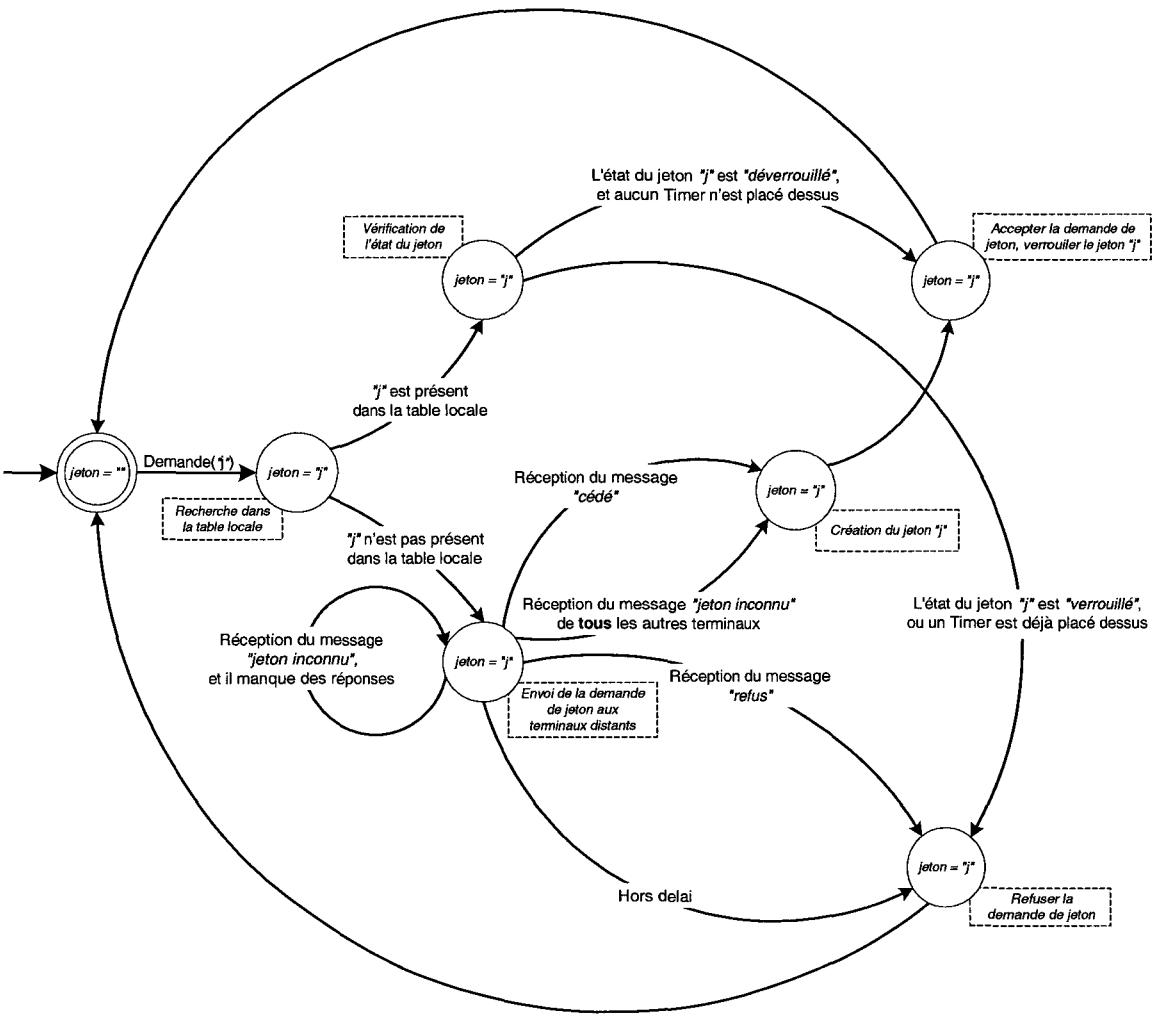


FIG. F.1 – *Automate exécuté par un terminal demandeur d'un jeton*

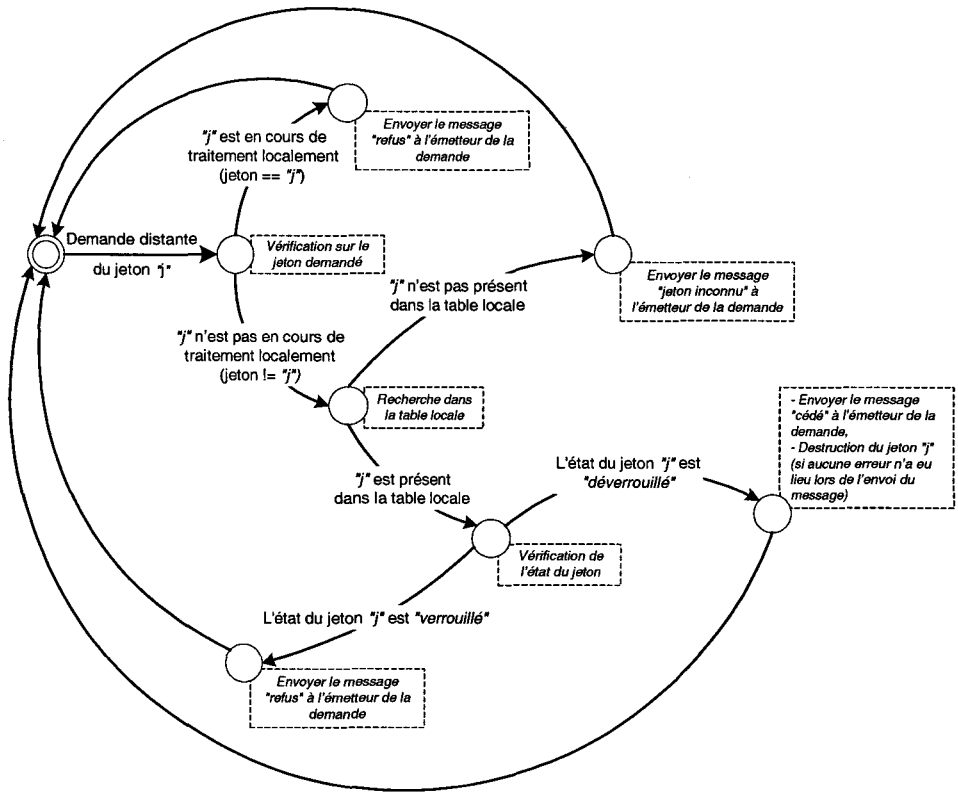


FIG. F.2 – *Automate exécuté lors d'une demande distante de jeton*

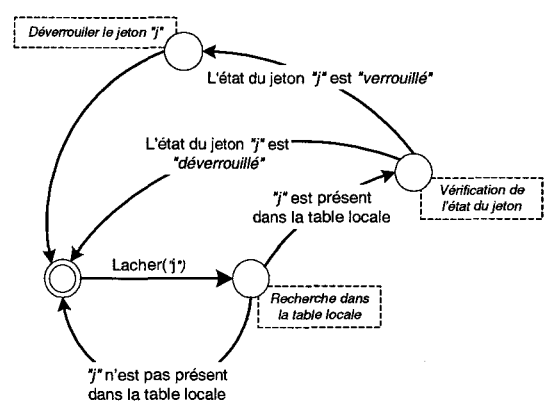


FIG. F.3 – *Automate exécuté lors d'un déverrouillage de jeton*

Annexe G

Exemple de programmes utilisant la DAI

Grâce aux avantages offerts par l'environnement CORBA, les applications externes se connectant à notre terminal en utilisant les mécanismes fournis par la DAI peuvent être écrites dans n'importe quel langage de programmation (à partir du moment où l'OMG a spécifié une translation du langage OMG-IDL vers le langage de programmation cible). Il existe de nombreuses translations du langage de spécification OMG-IDL vers des langages de programmation, comme par exemple Ada, C, C++, Java, Smalltalk, Lisp, etc.

Dans cette annexe, nous présentons un exemple d'application externe écrite dans les langages C++, Java et CorbaScript. Le but de cette application externe est d'interagir avec le contenu VRML 97 présenté sur la figure G.1 et de faire tourner la boite. Comme vont le montrer les différentes implémentations de l'application, l'interaction avec le contenu de notre navigateur VRML 97 étendu se fait de façon complètement transparente.

```
#VRML V2.0 utf8

DEF THEBOX Transform {
 children [
  Shape {
   appearance Appearance {
    material Material { diffuseColor 1 0 0 }
   }
   geometry Box {}
  }
 ]
}
```

FIG. G.1 – *Contenu VRML 97 avec lequel interagit notre exemple d'application externe*

G.1 En utilisant C++

Le code présenté sur la figure G.2 est celui de l'application écrite dans le langage C++. La syntaxe utilisée peut sembler obscure aux développeurs non familiarisés à la programmation CORBA. La spécification C++ ne définissant pas de "ramasse miette" (en anglais, *garbage collector*), la spécification CORBA utilise la notion de "*smart pointer*" afin de décharger le programmeur de la gestion mémoire des objets: les types possédant l'extension "*_var*" sont des *smart pointers*.

G.2 En utilisant Java

Le code présenté sur la figure G.3 correspond à l'application externe développée avec le langage Java. Il est à noter que la syntaxe est plus facile à appréhender qu'avec le langage C++. En effet, la spécification Java inclue un *garbage collector*, et par conséquent, les programmeurs n'ont pas à utiliser des types spéciaux permettant de simuler un *garbage collector* comme c'est le cas avec C++.

```
vrml::eai::Browser_var browser;
vrml::eai::Node_var transform;
vrml::eai::Field::EventIn_var event;
vrml::eai::Field::SFRotation_var rotation;
float v[4], angle;

// Initialize the ORB, FreeSSL plug-in somehow
(...)

// Retrieve references on objects contained in the browser
browser = vrml::eai::Browser::getBrowser();
transform = browser->getNode("THEBOX");
event = transform->getEventIn("rotation");
rotation = vrml::eai::Field::SFRotation::_narrow(event);
v[0] = 0.0f; v[1] = 1.0f; v[2] = 0.0f;
angle = 0.0f;

while(true) { // Main loop for the animation
 v[3] = angle;
 rotation->setValue(v);
 angle = angle + 0.01f;
}
```

FIG. G.2 – *Application externe implémentée en C++*

```
vrml.eai.Browser browser;
vrml.eai.Node transform;
vrml.eai.Field.EventIn event;
vrml.eai.Field.SFRotation rotation;
float v[4], angle;

// Initialize the ORB, FreeSSL plug-in somehow
(...)

// Retrieve references on objects contained in the browser
browser = vrml.eai.Browser.getBrowser();
transform = browser.getNode("THEBOX");
event = transform.getEventIn("rotation");
rotation = vrml.eai.Field.SFRotationHelper.narrow(event);
v[0] = 0.0; v[1] = 1.0; v[2] = 0.0;
angle = 0.0;

while(true) { // Main loop for the animation
 v[3] = angle;
 rotation.setValue(v);
 angle = angle + 0.01;
}
```

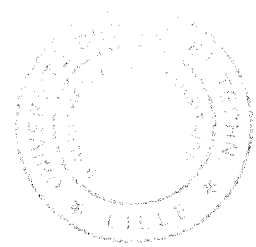
FIG. G.3 – *Application externe implémentée en Java*

G.3 En utilisant CorbaScript

En utilisant la spécification IDLScript (et son implémentation CorbaScript), les applications externes peuvent être développées plus facilement et plus rapidement qu'en utilisant les langages C++ et Java. La syntaxe proposée par CorbaScript est plus simple que celle des langages C++ et Java. Toutes les opérations complexes (comme l'initialisation de l'ORB, les opérations de conversion, etc.), qui avec les langages C++ et Java sont à la charge du programmeur, sont réalisées par l'interpréteur CorbaScript. CorbaScript est dédié à l'environnement de développement CORBA. Les scripts CORBA sont capables d'invoquer n'importe quelle méthode, de lire et écrire n'importe quel attribut de n'importe quel objet CORBA. La figure G.4 correspond à l'application externe développée sous la forme d'un script CORBA.

```
browser = vrml.eai.Browser.getBrowser()
node = browser.getNode("THEBOX")
rotation = transform.getEventIn("rotation")
angle = 0.0
while(true) {
 rotation.setValue([0.0 1.0 0.0 angle])
 angle = angle + 1.0
}
```

FIG. G.4 – *Application externe implémentée sous la forme d'un script CORBA*



Résumé

Le projet Spin-3D se propose de définir une plate-forme pour le développement d'applications coopératives synchrones permettant à un petit groupe de personnes, éloignées géographiquement, de se réunir, de travailler et de coopérer via une interface 3D. La médiatisation de l'activité implique de respecter certaines contraintes, notamment l'interactivité, afin d'offrir les meilleurs conditions de travail possibles aux utilisateurs. Dans cette thèse, nous nous sommes intéressés aux mécanismes de communication et de partage à mettre en oeuvre dans le terminal de travail Spin-3D.

Nous proposons de gérer la session de travail virtuelle de façon décentralisée. Chaque instance du terminal dispose d'une copie de la base de données 3D. Afin de conserver la cohérence des données partagées, chaque terminal est responsable de prévenir les terminaux distants des actions de son utilisateur. Pour cela, nous proposons deux canaux de communication offrant des niveaux de cohérence différents (cohérence forte et cohérence relâchée). Ces canaux de communication utilisent le multipoint IP et sont implémentés au dessus de l'infrastructure CORBA et de flots multimédia. Au dessus de ces mécanismes de communication, des services, basés sur un fonctionnement décentralisé, permettent de gérer la session de travail virtuelle.

Afin de faciliter le développement d'applications coopératives, nous masquons les différents problèmes techniques sous-jacents liés au maintien de la cohérence. Ainsi, nous proposons une abstraction de haut niveau des mécanismes de communication en utilisant le langage VRML97 pour la description des objets 3D, partagés ou non, présents dans l'interface de travail. Nous intégrons la description du partage de chaque objet 3D dans le fichier VRML97 contenant déjà la description de la géométrie et de l'interaction.

Pour développer des applications coopératives plus riches et dont le mécanisme d'interaction est inadapté à la 3D, nous proposons une interface de programmation, inspirée de l'EAI VRML97 à laquelle nous avons intégré de nouveaux services facilitant l'accès et le parcours du graphe de scène. Ainsi, des applications, qualifiées d'externes, sont capables de manipuler le contenu de notre terminal.

Mots clés : Environnement Virtuel Collaboratif, interface 3D, TCAO synchrone, réunion de petits groupes, multi-utilisateurs, architecture sans serveur, CORBA, flots multimédia, communication multipoint, VRML97, EAI.

Abstract

The Spin-3D project aims to design a complete platform for synchronous collaborative work with which it would be easy to create collaborative applications. Through a 3D interface, a small group of users is able to collaborate around 3D objects. We need to respect few requirements, such as interactivity, in order to enhance the feeling of collaboration within the 3D interface.

We use a serverless architecture in order to manage the virtual meeting. Each computer manages a copy of each shared object. Data are exchanged between the computers that participate in the collaborative session in order to maintain the overall coherency of duplicated objects. A communication platform providing different coherency levels supports network communication over IP multicast. The implementation is performed with the CORBA middleware and multimedia flows. Distributed services manage the virtual session.

We provide a high level abstraction in order to hide the underneath communication channels and the coherency problem. Designers are able to easily create multi-user objects, using VRML97 and our multi-user framework. These objects are presented within the 3D interface. The sharing description of each 3D object is embedded in the VRML97 file already containing the geometry and interaction.

We also provide an API, very close to the classical VRML97 EAI, in order to create complex collaborative applications. We enhance the standard EAI with new interfaces in order to easily traverse the whole VRML97 scene graph. Therefore, applications, running outside of the Spin-3D core, are able to manipulate objects contained within the Spin-3D browser.

Keywords : Collaborative Virtual Environment, 3D interface, synchronous CSCW, small group meetings, multi-user, serverless, CORBA, multimedia flows, multicast communication, VRML97, EAI.