ANNEE : 2003

THESE

présentée devant

l'université **Paul Sabatier de Toulouse** (sciences)

en vue de l'obtention

du grade de docteur de l'Université Paul Sabatier

spécialité : **Informatique**

par **Aura Nancy RODRIGUEZ**

ASSET :
Une architecture générale pour la télérobotique

Soutenue le 6 janvier 2003 devant le jury composé de :

Directeur de Thèse :	Jean-Pierre JESSEL	Toulouse
Rapporteurs :	Jacques LEMORDANT	Grenoble
	Jacques TISSEAU	Brest
Examinateurs :	Rachid ALAMI	Toulouse
	René CAUBET	Toulouse
	Yves DUTHEN	Toulouse
Invité :	Patrice TORGUET	Toulouse

Institut de Recherche en Informatique de Toulouse
Université Paul Sabatier – 118, route de Narbonne – 31062 Toulouse cedex

Resume

Pour être efficace, un système de téléopération doit fournir à l'opérateur toute l'information nécessaire à l'accomplissement de sa tâche. De plus, ces informations doivent lui parvenir sous une forme facile à interpréter et à travers un système de communication fluide. Pour ces diverses raisons, les environnements virtuels sont des solutions suscitant un très grand intérêt pour des applications de téléopération. La réalité virtuelle permet d'avoir une interface naturelle et intuitive, et d'intégrer différentes sources d'information pour améliorer la perception de l'opérateur. Dans le système ASSET (Architecture pour des Systèmes de Simulation et d'Entraînement en Téléopération) nous avons intégré les techniques de la réalité virtuelle pour concevoir et mettre en œuvre un environnement de développement de systèmes de téléopération. Cet outil est spécialisé dans la construction rapide de prototypes ou de programmes de tests pour de nouveaux dispositifs, modèles de simulation, techniques d'interaction et comportements d'entités autonomes.

L'objectif de notre travail est la construction d'un système qui facilite le développement de systèmes de télérobotique, en suivant la philosophie des systèmes de la réalité virtuelle et la simulation distribuée. Nous avons adopté pour ASSET un développement orienté-objet afin d'offrir un système modulaire, flexible et facile à utiliser. D'un autre côté, grâce aux langages utilisés pour la création de notre système (Java et Java3d) celui-ci est portable et de ce fait disponible pour plusieurs plates-formes matérielles. ASSET est indépendant des dispositifs utilisés et ne requiert pas une configuration matérielle particulière. Notre système peut donc être considéré comme une solution de réalité virtuelle accessible et utile pour la recherche en télérobotique.

abstract

Efficiency in teleoperation systems is strongly affected by the relationship between the operator and the remote environment. If the operator has access to easily comprehensible information, his capability for opportune decision-making increases. Virtual reality has shown to be an effective means of displaying information to the human operator. It allows the creation of three-dimensional and interactive environments, which can be explored and controlled in an intuitive way. In our system, called ASSET (Architecture for systems of Simulation and Training in Teleoperation), we have applied virtual reality techniques in the design and implementation of an environment for teleoperation systems development. This tool allows flexible customizing and can be used as a testbed for evaluating interaction techniques, devices, simulation models and autonomous agents behaviors.

In this work we aim at providing a tool for helping development of telerobotics systems, following philosophy of experimental platforms for virtual reality systems. We have also adopted distributed simulation techniques for minimizing the network bandwidth use and object-oriented development to offer a modular, flexible and easy to use system. Furthermore, because one of major limitations of actual systems is that they are available on just one or on very few platforms, we have chosen Java and Java3d to develop our system, so that it can run on any platform without further changes. Also, even if we can use high-end displays, we are using a conventional computer monitor to display the virtual world. Our system can thus be regarded as a low cost virtual reality solution that can be used for many applications in telerobotics research.

Remerciements

Je remercie Monsieur Luis Fariñas del Cerro pour m'avoir accueilli dans son laboratoire.

J'adresse mes remerciements à Monsieur le professeur René CAUBET pour m'avoir accueilli au sein de l'équipe Synthèse d'Images et Réalité Virtuelle de l'IRIT.

Je tiens également à exprimer mes remerciements à Monsieur Jean-Pierre JESSEL pour avoir accepté la responsabilité de directeur de thèse et pour la confiance qu'il m'a accordée tout au long de ces trois années.

Je souhaite adresser ma gratitude aux membres du jury de thèse. Je remercie mes rapporteurs : Monsieur Jacques LEMORDANT et Monsieur Jacques TISSEAU, pour le temps et les efforts que leur a demandé la lecture et l'analyse de ce travail. Je remercie également Messieurs Rachid ALAMI, René CAUBET, Yves DUTHEN et Patrice TORGUET pour avoir eu l'amabilité de participer au jury.

Je tiens à remercier tous les membres de l'équipe Synthèse d'Images et Réalité Virtuelle qui m'ont aidée dans mes recherches et spécialement Olivier Heguy, étudiant en thése de l'équipe, avec qui j'ai collaboré. J'accorde une attention particulière à Roger Pujado *per la seva amabilitat i el seu gran cor*.

Je remercie tous mes amis et ma famille qui m'a toujours encouragé même en étant de l'autre côté de l'Atlantique.

Finalement je remercie Christophe pour sa patience, son soutien inconditionnel et son aide pendant tous ces mois de travail et de remises en question…

Table de matieres

Table de Figures

INTRODUCTION

introduction

Contexte et problématique abordé par ce travail. Objectifs et plan de thèse.

La Téléopération

Un système de téléopération permet à un opérateur de réaliser une tâche à distance, en l'éloignant de l'environnement de travail et des machines qu'il contrôle. La téléopération élimine ainsi les risques relevant de travaux dangereux tels que l'exploration spatiale ou la manipulation de substances toxiques.

Historique

La manipulation de produits dangereux a été à l'origine de la télémanipulation :

> *A partir des années 1940, la production d'électricité d'origine nucléaire nécessite la manipulation de substances radioactives à une échelle importante. Les premiers télémanipulateurs ont donc été conçus pour répondre au besoin d'intervention dans les centrales. Il s'agissait de systèmes simples, symétriques, composés d'un bras maître avec une poignée et d'un bras manipulateur muni d'une pince, reliés par des câbles et des poulies. La vision de l'espace de travail était directe à travers un hublot protecteur et le choix d'une réalisation purement mécanique lié au besoin de robustesse du système vis-à-vis des radiations.*
>
> - Espiau - [Espiau00]

L'histoire des télémanipulateurs commence avec le manipulateur maître-esclave développé par Ray Goertz au Argonne National Laboratory (ANL) en 1948 [Goertz52]. Ce système, conçu pour la manipulation de substances toxiques, permettait à un opérateur de manipuler les matériaux radioactifs placés dans une cellule. Dans ce premier système, le manipulateur maître possédait la

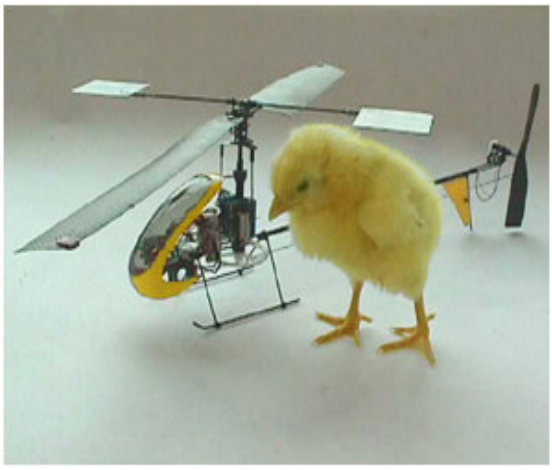
même structure mécanique et les mêmes propriétés cinématiques que le manipulateur esclave donc les limitations physiques de l'esclave étaient perçues naturellement par l'utilisateur. En 1954, l'ANL développe la deuxième génération de télémanipulateurs, électriques à retour d'effort [Goertz54]. Dans un système avec retour d'effort, toute force externe expérimentée par le manipulateur esclave est reproduite sur le manipulateur maître. Ceci est utilisé pour implanter le contrôle bilatéral : une force appliquée sur l'esclave (respectivement le maître) produira un mouvement du maître (respectivement de l'esclave). Les systèmes hydrauliques ont aussi été présents depuis le début de la télémanipulation. Le premier système de ce type est le « Handyman » de Ralph Mosher développé aux laboratoires de General Electric en 1958 [Mosher60]. Le Handyman, manipulateur à retour d'effort, consistait en deux bras hydrauliques à 10 degrés de liberté, 2 degrés de liberté pour chaque doigt.

Figure 0.1 : Environnement de télémanipulation au Argonne National Laboratory

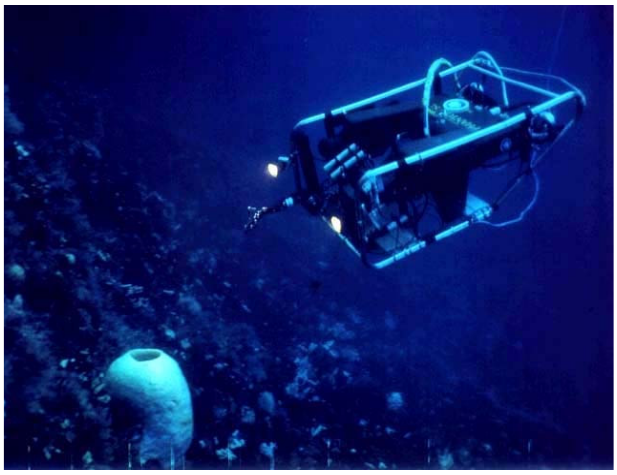
Au début des années 60 le champ d'application de la téléopération s'étend à l'exploration spatiale donc à la télémanipulation et au contrôle de véhicules avec des retards de communications [Ferrel65]. En 1970, l'exploration sous-marine devient l'un des principaux champs d'application des systèmes téléoperés. Dans les années 80, les véhicules opérés à distance ROV (Remotely Operated Vehicles) commencent à être amplement utilisés dans les industries du gaz et du pétrole. Les ROV sont utilisés pour la surveillance des oléoducs et pour l'inspection des structures artificielles sous-marines. La science s'est aussi intéressée aux possibilités de recherche

fournies par les ROV : ils sont utilisés pour l'observation de la flore, la faune et la géologie marine [Fong93, Sheridan92, Weast99].
Le domaine nucléaire reste un grand utilisateur de systèmes téléoperés. Dans les accidents de Three Mile Island et Chernobyl, des téléoperateurs ont été utilisés pour l'inspection et la décontamination des sites concernés. Les réalisations de Jean Vertut et son équipe au CEA montrent l'adaptation et l'évolution des teleoperateurs dans le domaine nucléaire : téléopérateurs montés sur porteurs, systèmes téléscopiques, robots longs et fins pour passer dans des canalisations, hexapodes, véhicules à roues, à chenilles, en trains...

L'apparition des ordinateurs et le progrès en matière de technologie ont permis le développement de systèmes de plus en plus performants dans de diverses fonctions : le dépannage de satellites, l'exploration de volcans, le déminage, l'aide aux personnes handicapés (avec, par exemple, les fauteuils roulants intelligents)… La recherche n'est pas limitée aux applications de robots manipulateurs et mobiles terrestres et aquatiques : les drones, petits véhicules volants partiellement téléopérés ou autonomes, suscitent un grand intérêt militaire et civil. En effet, les drones seraient des outils permettant la surveillance, le renseignement, la localisation d'accidentés en montagne, la cartographie automatique, etc.



a)



b)

Figure 0.2.a : Le mini hélicoptère Pixel 2000 [Pixel[1]]
Figure 0.2.b : ROV de la NASA pendant la réalisation d'une mission en Antarctique [Weast99]

L'intégration dans les systèmes robotiques de capacités avancées de perception et de décision a favorisé l'apparition de la téléopération coopérative. En effet, il est possible d'alléger la tâche de

[1] Les références marques avec le symbole sont présentées dans le chapitre Bibliographie rubrique URL.

contrôle et de supervision de l'opérateur en dotant les robots d'autonomie. Dans les systèmes de téléopération coopérative les utilisateurs reçoivent de l'assistance pendant l'exécution de leurs missions [Rogers95]. Ce type de système, reconnu depuis longtemps comme une technologie très utile pour l'exploration spatiale, a été intégré dans de nombreux contextes : la manipulation en centrales nucléaires, les opérations de sauvetage, les interventions en environnements hostiles, la rééducation et la téléchirurgie [Ottensmeyer96].

Problématique

Un système de téléopération consiste en trois composants principaux : l'interface de l'opérateur, (l'ensemble d'outils qui lui permettent de contrôler le système), le système distant qui réalise la tâche souhaitée et le schéma de communication qui relie les deux. L'interface de l'opérateur est située à une certaine distance de l'environnement de travail. Pour lui permettre de travailler d'une façon efficace, il est donc indispensable de lui présenter toute l'information nécessaire à l'accomplissement de sa tâche. De plus, ces informations doivent lui parvenir sous une forme facile à interpréter et à travers un système de communication fluide. Cette problématique particulière est représentée par trois aspects : la gestion de la latence, le contrôle de dispositifs et l'efficacité de l'interface homme-machine.

Latence

La latence est définie comme le temps que prend le système pour répondre aux commandes de l'utilisateur; en d'autres termes, le temps entre la communication des instructions et la visualisation des résultats. Des études sur la perception humaine montrent que pour établir une communication efficace avec l'utilisateur la latence d'un système interactif doit être inférieure à 50 ms [Ware94]. La latence dans les systèmes de téléopération est essentiellement due à la transmission d'information entre l'interface de l'opérateur et le système distant. Comme nous l'avons vu, dans la téléopération classique et dans le contrôle bilatéral, l'opérateur doit attendre le retour d'information (images vidéo et forces expérimentées par le manipulateur) du système distant. Les retards dans la communication augmentent le temps d'achèvement des tâches car l'utilisateur essaie de s'adapter à eux en utilisant des stratégies comme «déplacer et attendre» : réaliser une action très simple et attendre le retour pour continuer [Sheridan92]. En outre, lorsque les retards de communications sont importants, la perception du site de travail est dégradée et le système devient difficile à contrôler. Plusieurs solutions ont été proposées pour améliorer la performance des systèmes de téléopération en présence de retards : [Sheridan92,

Kosuge98, Leleve01]. L'approche adoptée généralement pour palier les effets de la latence est la fermeture de la boucle de contrôle dans le site local. Ceci peut être réalisé via une simulation graphique de l'action souhaitée, un simulateur prédictif ou la transmissions de séquences complètes de commandes. D'autres travaux utilisent des mécanismes dans le système distant de façon à incrémenter l'autonomie du système et diminuer les risques. La solution la plus adoptée jusqu'a présent est l'utilisation de simulations prédictives. La simulation prédictive permet la visualisation de l'état de l'environnement de travail sans attendre la réponse du système réel. Le nouvel état est calculé en appliquant les commandes de l'utilisateur aux modèles dynamiques qui représentent le robot et l'environnement. La simulation dans un système de téléopération présente aussi d'autres avantages comme la détection des collisions entre le robot et les objets du site de travail, le repérage des problèmes avant l'exécution des instructions et la visualisation du résultat des actions découplée du système réel.

Contrôle de dispositifs

Dans un système de téléopération, il est possible de distribuer les responsabilités entre l'opérateur humain et le système. Nous parlons de **contrôle direct** si l'opérateur prend en charge entièrement la réalisation de la tâche et de **contrôle automatique** quand le système exécute la tâche indiquée par l'opérateur. Il est aussi possible d'avoir un **contrôle partagé** : certains mouvements ou degrés de liberté peuvent être gérés par l'ordinateur de façon à permettre à l'opérateur de se concentrer sur l'essentiel de la tâche. Le **contrôle supervisé** est une stratégie intermédiaire entre le contrôle direct et automatique, résultat de la recherche dans le domaine de l'exploration spatiale. Le contrôle supervisé donne de l'autonomie au système distant pour permettre la réalisation de certaines tâches sans l'intervention humaine [Sheridan92]. Le contrôle supervisé est souvent employé dans la télémanipulation [Blackmon96] et dans une moindre mesure dans la téléopération de véhicules [Wettergreen95, Lin95, and Stone96]. Dans le **contrôle collaboratif** l'humain n'est plus le centre du système téléoperé [Fong00]. Dans ce type de contrôle, l'opérateur est utilisé comme une source limitée d'information à laquelle le robot peut faire des requêtes. Avec la **téléopération multi-opérateur**, plusieurs opérateurs partagent le contrôle. Dans le système créé par Cannon les opérateurs utilisent des « outils virtuels » pour définir des actions clefs à un niveau superviseur [Cannon97]. Un opérateur utilise ces outils graphiques pour diriger le robot en les plaçant sur la scène. Une interface distribuée permet aux différents opérateurs de partager le contrôle : un ensemble d'outils est affiché dans chaque site participant. La **téléopération coopérative** aide l'opérateur à réaliser sa tâche. Murphy décrit un système d'assistance qui combine un robot d'autonomie limitée avec un système assistant à base

de connaissances [Murphy96]. Le système fourni de « l'assistance stratégique » pour permettre à l'opérateur de coopérer avec le robot pour l'exécution de tâches complexes.

Interface homme-machine

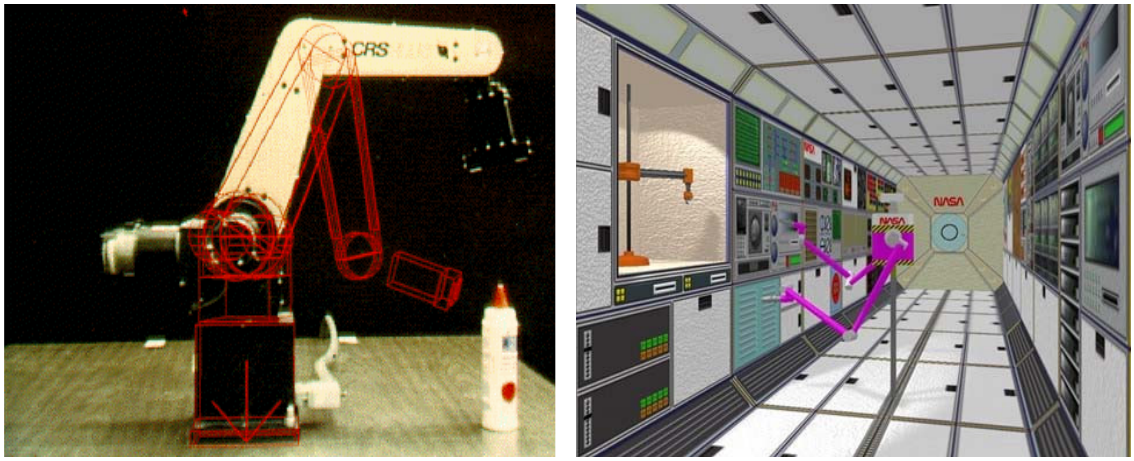
Dans les premiers télémanipulateurs, l'opérateur était séparé de l'environnement de travail par un hublot protecteur. Dans les années 50, l'inclusion d'un circuit fermé de télévision a permis à l'opérateur de s'éloigner beaucoup plus du site de travail. Les premiers systèmes de téléopération avaient donc une interface très simple : un dispositif pour guider le manipulateur distant et un ou plusieurs moniteurs avec des images vidéo de l'environnement de travail. Ensuite, l'utilisation des ordinateurs a facilité amplement le contrôle des dispositifs, mais pendant plusieurs années les interfaces ont été très complexes pour l'opérateur : à l'ensemble de moniteurs à surveiller s'ajoutent les consoles pour suivre l'état des dispositifs ou la valeur de certaines variables (température, pression,…). Dans ce type d'interfaces l'information provenant du système distant est dégradée car le champ de vision est limité et le retour visuel n'est pas assez riche. Dans ces conditions, l'exécution d'une tâche peut prendre très longtemps ou être impossible à réaliser.

Pour améliorer l'interface et la qualité de l'information présentée à l'opérateur, les systèmes de téléopération ont cherché à intégrer les différentes sources d'information en incorporant des techniques de réalité augmentée ou de réalité virtuelle. La réalité augmentée améliore la perception de l'opérateur en utilisant une superposition d'images de synthèse sur des images réelles ou de vidéo. Gradecki [Gradecki94] a définit la réalité virtuelle comme une « technologie qui permet à un utilisateur d'interagir avec les objets qui forment un environnement virtuel et de les regarder à partir d'un point de vue quelconque ». La réalité virtuelle offre aux utilisateurs la capacité de percevoir et de contrôler les systèmes complexes d'une façon naturelle, ce qui a conduit à son utilisation dans de nombreux domaines : entraînement militaire et médical, jeux, architecture, tourisme… Dans la téléopération, la réalité virtuelle permet à l'utilisateur d'interagir avec un monde entièrement généré par ordinateur qui représente le monde réel.

Téléopération et Réalité Virtuelle

Dans la téléopération, la qualité de la communication homme-machine est fortement liée à la performance du système. En effet, l'interface utilisateur est un facteur primordial dans la communication efficace de l'information [Fong00]. En utilisant un environnement virtuel comme interface utilisateur d'un système de téléopération, l'opérateur retrouve toute l'information spatiale dont il a besoin pour travailler, éliminant ainsi la nécessité d'avoir plusieurs images vidéo.

De plus, les interfaces basées sur la réalité virtuelle sont capables de compenser les délais très longs de communications car le contrôle abstrait est moins sensible aux latences de retour que le contrôle direct [Pook95]. Différents projets ayant suivi cette direction [Nguyen01] montrent que les interfaces de réalité virtuelle peuvent améliorer la connaissance de la tâche à réaliser et fournissent des outils considérables pour comprendre et analyser l'environnement distant. En outre, grâce à la simulation 3D interactive, l'opérateur peut expérimenter naturellement avec les machines et les objets de l'environnement distant sans s'inquiéter pour les dommages qui peuvent être provoqués par un usage inapproprié.



a) b)

Figure 0.3.a : Interface de réalité augmentée
Figure 0.3.b : Interface de réalité virtuelle

Objectifs et Plan de thèse

Un système de téléopération intègre donc des composants technologiques très variés : la simulation, l'affichage 3D, la création de scènes (géométrie et comportement), la gestion de dispositifs et les communications. De ce fait, les outils de développement de systèmes de téléopération doivent permettre aux chercheurs et développeurs d'explorer et de tester leurs idées de façon indépendante, sans affecter le processus d'intégration et l'évolution de l'application finale tout en évitant les développements inutiles. De plus, dans le cas d'applications coopératives où les robots d'assistance peuvent être utilisés et contrôlés de diverses manières, il est important pour le développeur d'avoir des systèmes de construction flexibles et d'utilisation simple pour concevoir et évaluer ses applications.

Il existe à présent un grand nombre d'architectures pour des systèmes robotiques. Cependant, la plupart de ces architectures ne traitent qu'un seul aspect de la problématique de la téléopération : le contrôle de dispositifs, l'autonomie ou l'interface utilisateur. Il existe très peu de projets fournissant une solution intégrale pour le développement d'un système de téléopération. Ce type de système existe, et en grand nombre, dans le domaine de la réalité virtuelle. Cependant, même s'ils veulent faciliter la construction de nouveaux systèmes, la complexité ou les ressources requises pour les utiliser limitent énormément leur emploi dans la construction rapide et simple de nouvelles applications. Le but de ce travail de recherche est donc de concevoir et mettre en œuvre un outil de développement et de test pour des systèmes de téléopération, basé sur les environnements virtuels. Le système ASSET (Architecture pour des Systèmes de Simulation et

d'Entraînement en Téléopération) est un outil spécialisé dans la construction rapide de prototypes, de programmes de tests pour nouveaux dispositifs ou modèles de simulation, et pour l'étude des différentes possibilités d'interaction dans la collaboration humain-robot. Notre système propose une solution aux différents aspects de la problématique des systèmes téléoperés permettant ainsi de produire facilement des applications fonctionnelles.

Ce document décrit le travail réalisé dans le développement de la plate-forme ASSET. La première partie introduit l'état de l'art dans les domaines abordés par cette thèse : la réalité virtuelle distribuée et la téléopération. L'étude de ces domaines est nécessaire pour clarifier la problématique traitée dans le système ASSET. Dans le domaine de la téléopération, nous nous intéressons notamment aux travaux réalisés en architectures d'applications de télérobotique et aux systèmes de téléopération sur Internet. Ces derniers, grâce à la facilité de mise en œuvre et la simplicité des interfaces, ont réussi à rendre accessibles de nombreux systèmes robotiques. Nous présentons ensuite, après la définition d'un ensemble de critères d'étude, les systèmes de développement d'applications de réalité virtuelle les plus significatifs. Puis, dans le dernier chapitre de cette partie, nous réalisons une analyse des caractéristiques souhaitées d'un environnement de développement pour la téléopération. La deuxième partie de ce document est structuré de la manière suivante : dans le chapitre 1, nous présentons l'architecture de téléopération définie par notre système, les modules qui la composent, et son fonctionnement. Le système ASSET est ensuite détaillé dans le chapitre 2 par rapport aux différents critères impliqués dans la construction d'un système de développement d'applications de téléopération. Le chapitre 3 présente les choix techniques que nous avons fait pour la mise en oeuvre de notre architecture. Enfin, dans le chapitre 4 nous décrivons plusieurs des applications réalisées avec le système. Nous présentons notamment le système ROVE, conçu pour l'étude de l'interaction homme-robot dans le cadre de la téléopération coopérative. Une dernière partie, présentant une discussion sur comment notre système satisfait les exigences d'un environnement de développement et les perspectives envisagées pour notre travail, clôt ce mémoire.

PARTIE 1

état de l'art

Etat des recherches dans les domaines de la téléopération et les systèmes de développement d'applications de réalité virtuelle

chapitre 1

Systèmes de Téléopération

La téléopération permet de réduire, pour les opérateurs humains, les risques associés aux travaux se déroulant dans des environnements hostiles. Les premiers systèmes de téléopération ont donc été développés pour la manipulation de produits radioactifs et l'exploration spatiale et sous-marine. Puis, grâce aux progrès en matière de technologie et à l'intégration de fonctions avancées (notamment la locomotion) le champ d'application de la téléopération s'est étendu à plusieurs domaines : l'intervention en sites contaminés, la surveillance, l'aide aux handicapés et le service.

Un système de téléopération actuel peut intégrer des composants technologiques très variés : des interfaces haptiques sophistiquées (exosquelettes), des robots avec une certaine capacité de décision, des techniques de visualisation avancées (réalité virtuelle) et divers outils (schémas de contrôle partagé, planification). Ceci, et les conditions héritées des premières applications auxquelles ils étaient destinés, font que la construction d'un système de téléopération est coûteuse en ressources et en temps de développement. Depuis plusieurs années, des efforts ont été faits pour que la téléopération devienne plus accessible. Plusieurs projets cherchent à diminuer le coût et la complexité du développement de ce type de systèmes. Dans ce chapitre, nous présentons les deux principales tendances regroupant ces efforts : la téléopération sur Internet et la définition d'architectures génériques pour les systèmes de téléopération.

Téléopération sur Internet

De très nombreux systèmes robotiques opérés à travers le World Wide Web ont été développés ces dernières années. Ces systèmes requièrent une infrastructure très simple pour leur déploiement et sont accessibles dans le monde entier. De plus, puisque l'interface est facile à comprendre et à maîtriser, l'utilisateur ne nécessite pas d'entraînement. Mais surtout, la téléopération sur Internet ouvre de très importantes possibilités de collaboration et de partage de ressources entre les différentes équipes de recherche disséminées dans le monde.

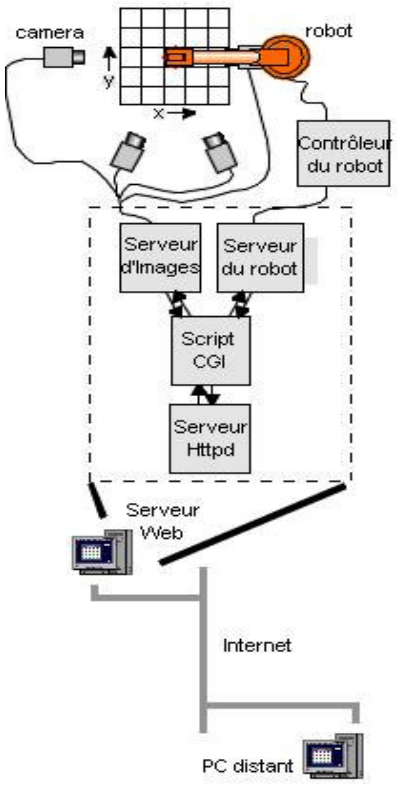


Figure 1.1.1 : Architecture du contrôle d'un manipulateur via le World Wide Web

Le premier dispositif « distribué » sur Internet a été le « Cambridge Coffee Pot ». Ce système utilisait une webcam pour transmettre des images d'une cafetière placée dans le laboratoire d'informatique de l'Université de Cambridge [Stafford-Fraser95]. Le projet Mercury [Goldberg95] a été mis en ligne en août 1994. Il permettait aux utilisateurs d'attraper et de manipuler différents objets en utilisant un bras robotique. En septembre 1994, un autre telemanipulateur a été mis ligne cette fois dans l'Université de Western Australia [Taylor95]. Dans la première version de ce système (figure 1.1.1), les utilisateurs devaient saisir des coordonnées spatiales pour spécifier les

mouvements du manipulateur. Plusieurs autres interfaces ont été utilisées ensuite pour simplifier le contrôle du robot [Taylor97, Dalton98].

L'idée de rendre accessible un dispositif en utilisant une interface web a été suivie par de nombreux projets, parmi lesquels le télescope robotique de Cox [Cox1994], le telerobot mobile « FortyTwo » à Manchester [Nehmzow96], le système DIGIMUSE, conçu pour la visualisation interactive des objets d'art [Goldberg98] et l'interface web pour la mission Pathfinder de la NASA [Backes98] qui permettait aux chercheurs de collaborer entre eux et travailler dans la mission sans se déplacer au centre de contrôle en Californie. D'autres projets se sont intéressés au contrôle distant de robots mobiles comme les systèmes KhepOnTheWeb [Michel97], le robot Xavier de l'Université Carnegie Mellon [Simmons98], le projet Museum Tour-guide [Burgard98] et le projet WebPioneer [Fong00]. Ces systèmes permettent à l'utilisateur de contrôler un robot soit dans un environnement statique, soit dans l'exploration d'environnements dynamiques.

La plupart des systèmes de contrôle sur Internet ont été construits en utilisant des pages HTML statiques et des programmes CGI (Common Gateway Interface). La page HTML permet à l'utilisateur de connaître l'état du système et de générer les ordres à exécuter. Les ordres sont traités et exécutés ensuite par le programme CGI qui contrôle le robot, sans supervision ni intervention de l'utilisateur. Enfin, lorsque l'ordre est achevé ou qu'une erreur s'est produite, une nouvelle page HTML est générée. L'interaction avec l'utilisateur est donc limitée car il est impossible de récupérer les consignes de l'utilisateur et de lui présenter l'information de retour en même temps. Pour résoudre ce problème, l'utilisation de plusieurs programmes CGI associés aux différents cadres d'une page HTML a été proposée. Cependant, puisqu'un programme CGI établit une nouvelle connexion à chaque fois qu'il est invoqué, leur multiplication augmentait le temps de réponse du système. L'alternative adoptée pour la gestion de l'interface a été l'utilisation d'applets Java à la place de HTML/CGI. Avec une applet Java, la connexion entre l'interface Web et le contrôleur du robot est établie une seule fois et la transmission de données peut donc se réaliser à tout moment dans les deux directions. Le système WebDriver [Grange00] est un exemple d'application de cette technologie. Etant le successeur du système WebPioneer, WebDriver est conçu pour la téléopération de robots mobiles évoluant dans des environnements dynamiques inconnus. L'interface utilisateur de WebDriver (figure 1.1.2) est une applet Java qui reçoit les commandes de l'utilisateur et affiche l'information des capteurs du robot.

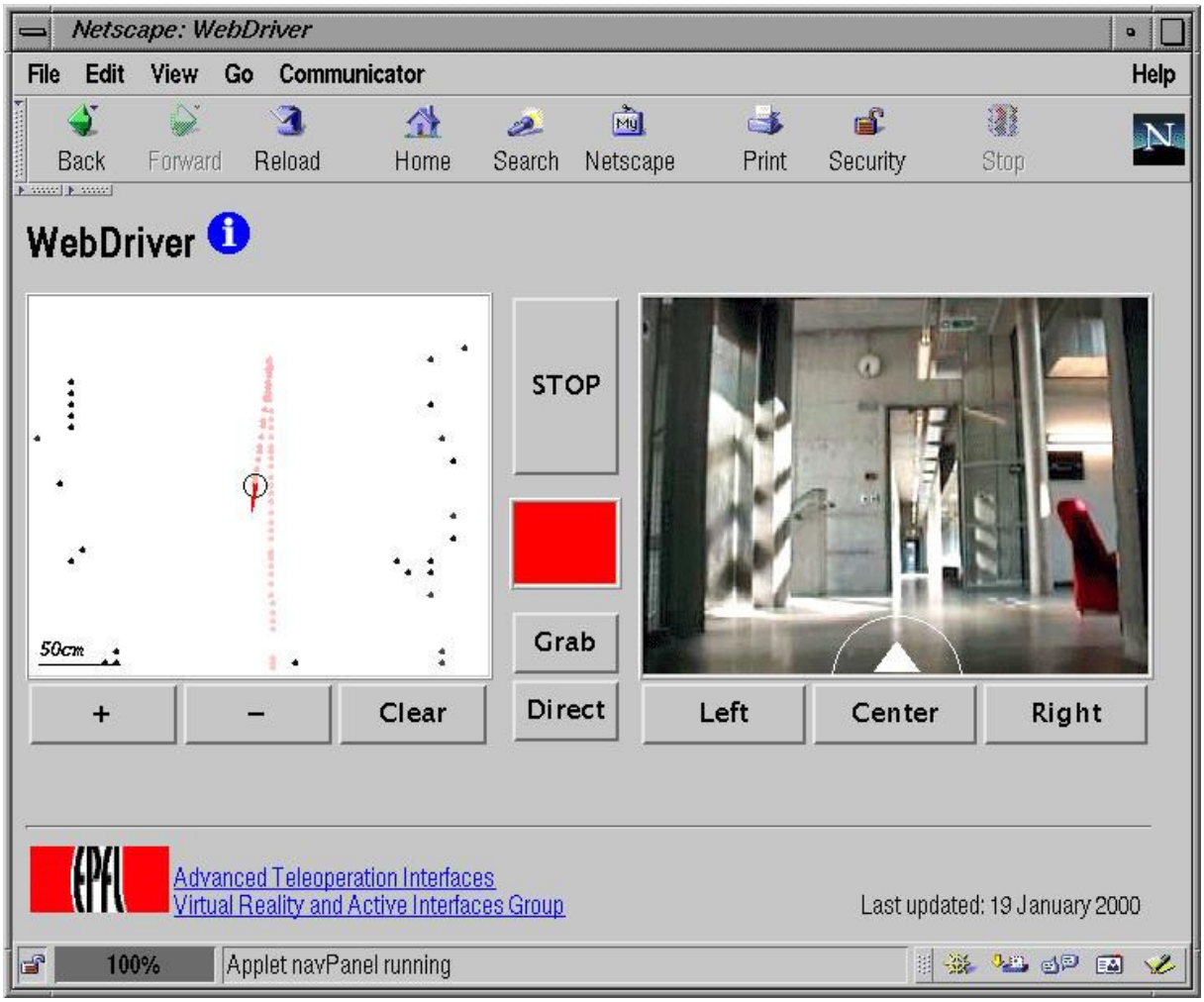


Figure 1.1.2 : Interface Utilisateur de WebDriver

WebDriver met à jour les images de l'interface utilisateur lorsque certains événements (comme la détection d'un obstacle) se produisent. Ce schéma permet de réduire le trafic du réseau généré par un serveur vidéo - très utilisé car facile à mettre en œuvre. Cependant, en limitant l'information du site de travail aux seules images vidéo, l'utilisateur a une perception limitée de l'environnement. Pour améliorer l'information de retour et gérer les délais de transmission, il est possible de réaliser une simulation graphique (2D ou 3D) de l'action souhaitée. La simulation est ensuite superposée aux images réelles pour aider l'utilisateur à réaliser sa tâche [Rastogi95]. Une autre approche consiste à prendre en compte les délais pour contrôler le système téléoperé. Ceci requiert de la part d'une simulation dynamique un predicteur de l'état actuel et futur du système distant [Kosuge98]. Ces idées ont été utilisées dans la construction du système décrit par Leleve[Leleve98], pour la téléopération d'un véhicule portant un manipulateur. Dans ce système, l'association entre le bloc de *prédiction* et le *modèle dynamique* permet d'effectuer des pré-simulations et d'afficher une prédiction de l'état du système distant avant de le recevoir. L'utilisation d'une

simulation 3D permet aussi le contrôle d'un telerobot lorsque les délais de communications sont importants. Aditya et al. [Aditya00], ont utilisé Java3D pour créer et manipuler un robot. La simulation du robot substitue les images réelles afin de réduire les délais de communication et le trafic du réseau. Hirukawa et al. [Hirukawa97] décrivent des interfaces web qui permettent à l'opérateur de manipuler des objets en utilisant une simulation graphique 3D contenue dans le navigateur. Ces interfaces suivent une approche de télé-programmation. En effet, les tâches sont d'abord testées dans le simulateur et ensuite, la séquence d'actions à réaliser est transmise au robot réel. Dans le système décrit par Finke [Finke99], un robot mobile autonome est connecté à un environnement virtuel. L'utilisateur peut non seulement envoyer des consignes au robot mais aussi influencer son comportement en plaçant des obstacles virtuels. L'information des capteurs du robot ainsi que les déductions réalisées par le robot sont affichées dans l'environnement virtuel, ceci permettant à l'utilisateur de valider les décisions prises par le robot.

Pour faciliter la mise en ligne des dispositifs, Ghiasi et al [Ghiasi99] a proposé un système réutilisable, conçu pour permettre la manipulation de dispositifs via le World Wide Web. Le système, construit avec Java et Python, cherche à réduire le nombre et le niveau des compétences requises pour déployer un dispositif téléoperé. Il fournit des mécanismes pour interagir avec le dispositif, pour construire des interfaces utilisateurs et pour réaliser des extensions et des modifications aux composants déjà existants. D'autres travaux utilisent des technologies de distribution comme Jini [Jini] pour permettre le partage et le contrôle d'objets et de dispositifs dans un environnement virtuel multi-utilisateur. Dans le système décrit par Inostroza [Inostroza02], l'API d'accès au graphe de scène est publiée comme un service Jini, permettant ainsi aux participants d'intervenir et à d'autres services (disponibles sur Internet) de collaborer dans la composition dynamique de l'environnement 3D. D'autres travaux montrent actuellement l'intérêt de l'utilisation des technologies middleware pour l'interconnexion des différents composants d'un système robotique [Gill02]. CORBA, par exemple, a été utilisé dans la construction d'un système d'assistance pour personnes à mobilité réduite [Jia01], l'implantation d'un laboratoire distribué [Paolini97] et le développement d'un système d'exploration et de reconstruction d'environnements distants [Aleotti02]. L'interconnexion des systèmes hétérogènes peut aussi être facilitée par l'adoption d'un langage standard. Ceci a été suggéré par divers travaux utilisant XML (Extensible Markup Language) pour la description des services offerts par les dispositifs et l'écriture des instructions [Blanks99, Salmi00].

L'USARC (Universities Space Automation/Robotics Consortium) a développé TCS (Telerobotics Construction Set), un outil de construction de systèmes distribués. Ce système utilise un modèle de distribution des objets pour permettre la mise en œuvre de systèmes télérobotiques [Kondraske93]. Les interfaces fournies par le TCS pour le contrôle du robot et la gestion de la vidéo et des données de l'application, permettent l'intégration de ressources hétérogènes dans le système. Un mécanisme appelé TSM (Telerobotics Session Manager) permet à l'opérateur de reconfigurer dynamiquement les flux de données et les interconnexions du système développé.

Les derniers systèmes cités représentent une partie des travaux entrepris pour donner aux développeurs des outils pour la construction de systèmes téléoperés. Cet intérêt est partagé par les architectures proposées pour la téléopération que nous présentons dans le paragraphe suivant. Ces architectures, qui ne sont pas limitées à la téléopération sur Internet, cherchent à faciliter la conception et la construction des systèmes de téléopération, notamment dans le domaine du contrôle des dispositifs.

Architectures pour la téléopération

Il existe un très grand nombre d'architectures pour la télérobotique. Cependant la plupart d'entre elles sont des architectures de contrôle de dispositifs, et seulement quelques projets ont cherché à structurer les systèmes de téléopération pour faciliter leur développement.

Les architectures de contrôle les plus significatives ont été proposées par le Jet Propulsion Laboratory. Un des résultats de leurs recherches a été le Telerobotic Testbed, conçu pour le développement des systèmes autonomes, multi-robot, pour la révision de satellites [Balaram92]. Le Telerobotic Testbed était un système très complexe dont la structure logicielle n'a pas survécu à la fin du projet. Plus tard, plusieurs projets de recherche sur des manipulateurs en orbite ont été mis en place : le projet Remote Surface Inspection écrit en C sur VxWorks, le projet Satellite Servicing développé en C et assembleur, et le projet MOTES écrit en Ada sur VxWorks [Backes93, Bejczy91, Volpe94]. Ces projets ont été implantés séparément, sans partage de code, même s'ils fournissaient une fonctionnalité similaire. Le même scénario s'est déroulé avec les projets JPL contrôlant des robots mobiles. Les différences d'architecture et d'outils de développement ont empêché l'utilisation de ces nombreux projets dans d'autres missions. Par exemple, le vehicule Sejourner – participant à la mission Pathfinder - a été programmé avec un

système entièrement nouveau. Après le Sejourner, la recherche d'une infrastructure logicielle modulaire, reconfigurable et réutilisable a commencé au JPL. Plusieurs systèmes avec cet objectif ont été produits : ControlShell, FIDO, DARPA TMR, Nanorover [Dvorak00]. Pour recentrer les efforts et profiter des expériences acquises sur les schémas de contrôle et l'autonomie dans les projets RAX (Remote Agent Experiment), réalisé en collaboration avec le Centre de Recherche Ames de la NASA [Muscettola98], et le projet MDS (Mission Data System), le projet CLARAty a été créé. CLARAty (Coupled Layer Architecture for Robotic Autonomy) est une architecture conçue pour le contrôle de robots autonomes [Volpe01]. La structure de CLARAty contient une couche de Décision et une couche Fonctionnelle. La couche Fonctionnelle est une interface à toute l'architecture matérielle du système et de ses services. La couche de Décision, elle, utilise les capacités de la couche Fonctionnelle pour réaliser les activités lui permettant d'atteindre ses objectifs. Une première implantation de cette architecture est en train d'être réalisée afin de l'intégrer dans les plate formes de recherche des véhicules d'exploration de Mars, Rocky 7 et 8. En outre, une collaboration avec la NASA a été instaurée dans le but d'avoir une implantation complète de l'architecture dans le courant 2007.

Le système VEVI (Virtual Environment Vehicle Interface), développé par le Centre de Recherche Ames de la NASA, est une interface utilisateur modulaire pour la téléopération directe et le contrôle supervisé de véhicules robotiques [Piguet95]. VEVI présente une scène 3D constituée par les modèles du véhicule contrôlé et de l'environnement dans lequel il opère. L'interface permet de connaître l'état du système, de planifier et d'examiner des actions, et d'envoyer les consignes devant être exécutées par le véhicule. VEVI a été conçu et implanté pour être modulaire, distribué et facile à opérer. VEVI peut être configuré en ajoutant des modules d'entrée/sortie qui communiquent avec le mécanisme robotique à contrôler. La communication est réalisée suivant un protocole de communication spécifique. Le mécanisme robotique est constitué d'un ensemble d'objets affichés dans l'environnement 3D de VEVI. L'information géométrique de ces objets et leurs relations cinématiques sont décrites dans les fichiers de configuration de VEVI. Le système VEVI a été utilisé dans plusieurs projets de recherche, parmi lesquels le robot sous-marin TROV (Telepresence Remotely Operated Vehicle) travaillant dans l'Antarctique [Hine94] et le robot à 8 pattes DanteII employé pour la recherche scientifique dans le cratère du volcan Mount Spurr en Alaska [Wettergreen95, Bares97].

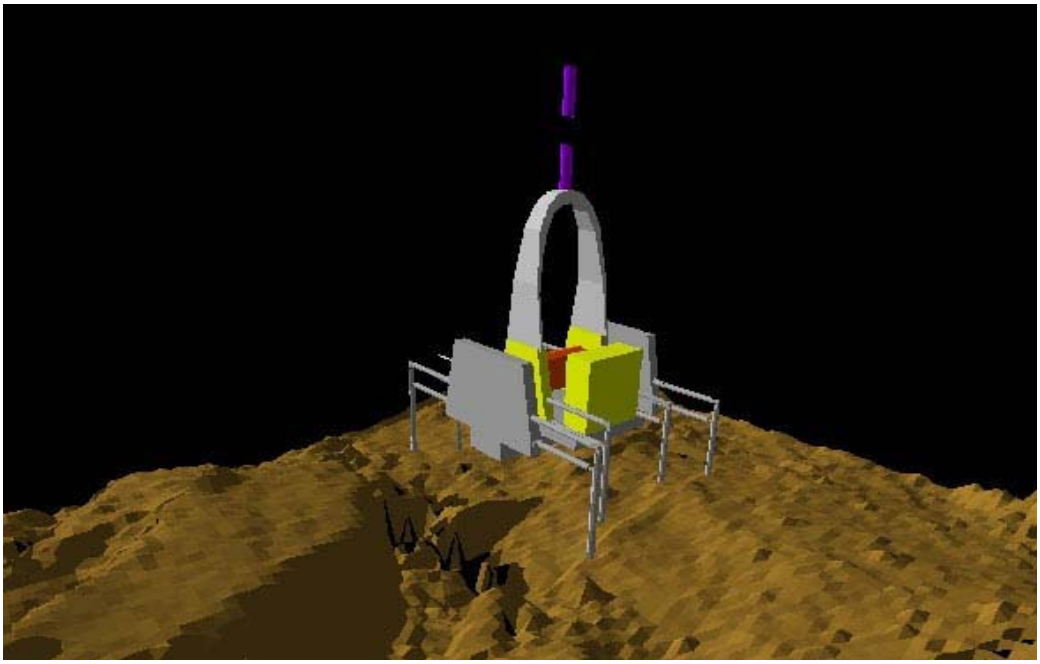


Figure 1.1.3 : Le robot DanteII dans le système VEVI

L'experience acquise dans le développement du système VEVI a été utilisé dans la mise en œuvre du projet Viz. Viz est une interface de réalité virtuelle, modulaire, flexible et facilement configurable [Nguyen00]. Viz a été conçu pour l'intégration simple des systèmes commerciaux existants et des technologies développées par la NASA. Cette stratégie cherche principalement à réduire le cycle de développement et permettre ainsi la construction rapide de prototypes.

Le module 3D de Viz est un serveur générique auquel les clients se connectent via le protocole TCP/IP. Une fois que le client est connecté au serveur, il peut modifier la scène dynamiquement en envoyant des messages. Le serveur interprète les messages des clients et réalise le rendu de l'environnement virtuel. En outre, une API pour l'envoi de messages est générée automatiquement en Java, C++ et Python à partir de la spécification des messages. En utilisant cette API, il est possible de transformer un module externe en un client de Viz. Viz peut être exécuté sur les plates-formes SGI et Linux, et la visualisation 3D du système est basée sur la bibliothèque graphique OpenInventor. Le système Viz a été évalué dans la préparation de la mission Mars Polar Lander (fig 1.1.4) pour intégrer dans une même interface les différents outils de planification, simulation et traitement de données développés par la NASA et le JPL.

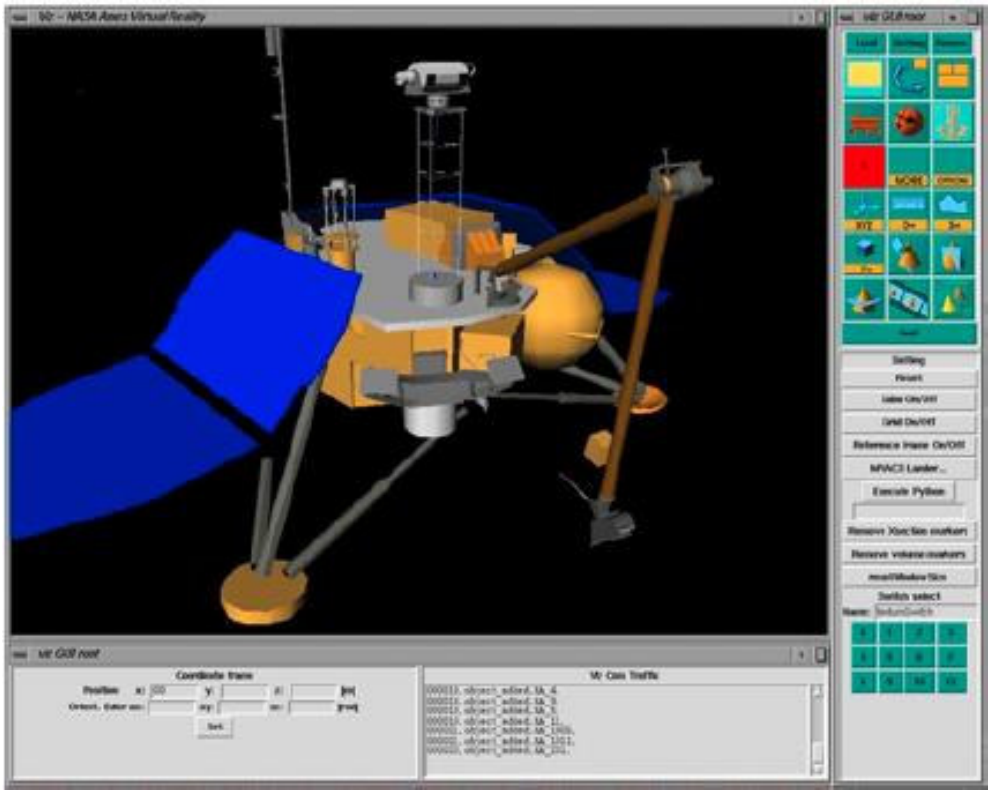


Figure 1.1.4 : Interface Viz pour la mission Mars Polar Lander

Les projets VEVI et Viz ont réalisé avec succès l'intégration d'interfaces de réalité virtuelle dans le contrôle de mécanismes robotiques. Cependant le contexte d'application de ces systèmes (exploration spatiale) fait que les besoins de performance, de sophistication et donc de coût des composants impliqués, sont très élevés. Pour cette raison, plusieurs projets proposant des plate-formes de test pour la téléopération à faible coût, ont été développés. Par exemple, Kaber et al [Kaber99] à l'Université de l'Etat du Mississippi, a développé une plate-forme de test pour la téléopération sur Windows NT/PC. Ce système, appelé VEMI (Virtual Environment Manipulator Interface) utilise un environnement virtuel pour contrôler un robot PUMA 560. Un modèle mathématique a été intégré dans VEMI pour que les mouvements du modèle virtuel correspondent à ceux du manipulateur réel. VEMI communique avec le contrôleur du robot en utilisant le protocole TCP/IP, ce qui permet de contrôler le robot au travers d'un réseau local ou d'Internet. Des plates-formes de test pour des robots mobiles ont aussi été implantés, comme le système KITE (Khepera Integrated Testing Environment) développé pour l'évaluation d'algorithmes de localisation et navigation [Sahin99], et l'environnement de test pour robots autonomes réalisé par Finke [Finke99].

D'autres projets, comme le projet CINEGEN [Flückiger98] travaillent sur le prototypage rapide de robots manipulateurs. CINEGEN propose un outil, simple d'utilisation et rapide à mettre en

œuvre, pour la simulation cinématique de robots manipulateurs à structure quelconque dans un environnement de type réalité virtuelle. Ce projet cherche à faciliter, pour les non-spécialistes, les aspects de simulation ou de programmation hors-ligne de robots. La mise en oeuvre rapide d'une nouvelle simulation s'effectue grâce à la description dans un fichier texte, des paramètres géométriques élémentaires de la structure du robot et les différentes contraintes que la structure comporte. Le fichier de description est analysé par le programme qui construit de manière automatique un modèle numérique de la cinématique du robot en vue de satisfaire toutes les contraintes. En fonction des consignes données en mode interactif par l'utilisateur, le "moteur" cinématique calcule les mouvements que le robot doit effectuer tout en respectant les contraintes externes. L'utilisation d'un environnement virtuel permet l'interaction entre l'utilisateur et le modèle contrôlé pour la définition de tâches et l'étude intuitive du comportement du robot.

Une architecture générale de contrôle de robots mobiles utilisant la réalité virtuelle a été décrit par Kheddar [Kheddar98]. Ce système défini un schéma de contrôle basé sur un principe appelé «le robot caché» [Kheddar97]. L'opérateur du système réalise les tâches dans une représentation fonctionnelle intermédiaire (RFI) de l'environnement virtuel, i.e. un environnement virtuel ou augmenté. Pour réaliser la tâche virtuelle, l'opérateur agit sur l'environnement virtuel de façon intuitive en utilisant des gestes naturels de la main. Un module se charge ensuite de percevoir et d'interpréter la tache virtuelle réalisée pour la transformer en commandes pour le robot. Avec un RFI, il n'est pas nécessaire d'afficher le robot esclave qui est donc visuellement caché pour l'opérateur. Dans ce système, l'opérateur réalise une tâche virtuelle au lieu de contrôler directement le robot qui exécute la tâche souhaitée.

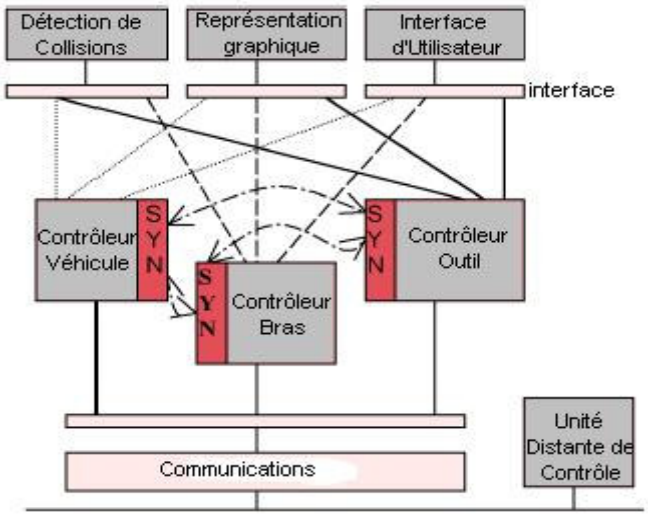


Figure 1.1.5 : Architecture de référence pour la téléopération

Alvarez et al [Alvarez00, Alvarez01a, Alvares01b] ont proposé une architecture de référence (figure 1.1.5) pour des systèmes de téléopération. Cette architecture est constituée de plusieurs composants : une représentation graphique pour l'affichage du robot et de l'environnement de travail, un composant de détection de collisions, l'interface utilisateur pour l'interaction avec l'opérateur, un composant de communications et le contrôleur, responsable de la gestion et de l'exécution des consignes. Cette architecture de référence a été utilisée dans divers projets, parmi lesquels, un système de contrôle pour un véhicule d'inspection sous-marine et un système robotique pour la maintenance de centrales nucléaires. Actuellement, l'architecture est adaptée pour le développement d'une unité de contrôle pour un robot autonome. Les différentes implantations de l'architecture ont été réalisées avec des bibliothèques commerciales comme Robcad et Paradise.

Conclusion

Dans ce chapitre nous avons présenté l'état de recherches dans le domaine de la téléopération. Nous avons vu comment l'utilisation d'Internet et des interfaces Web a réussi à rendre accessibles de nombreux systèmes robotiques, permettant ainsi la collaboration entre les équipes de recherche et l'ouverture de la robotique à un large public. Les interfaces de réalité virtuelle pour la téléopération ont aussi été présentées. Plusieurs projets ont montré que l'utilisation des environnements virtuels permet à l'utilisateur d'augmenter ses connaissances sur l'environnement de travail et sur la tâche à réaliser. Puisque la réalité virtuelle rend possible l'exploration et la manipulation intuitives, elle permet à l'utilisateur de comprendre et d'analyser l'environnement distant.

La recherche sur les architectures pour la téléopération a aussi été étudie. La leçon à retenir de cette étude est le manque d'un environnement de construction d'applications prenant en compte tous les aspects impliqués dans le développement d'un système de téléopération et non seulement au niveau du contrôle de dispositifs effecteurs. Plusieurs environnements de ce type existent dans le contexte de la réalité virtuelle, leur étude étant le sujet du chapitre suivant.

chapitre 2

Réalité Virtuelle

Pour développer une application de réalité virtuelle il est nécessaire de considérer plusieurs éléments comme la gestion de l'architecture matérielle et des différents dispositifs, la construction et l'affichage de la scène 3D ainsi que les calculs liés à la simulation. En outre, dans les systèmes multi-utilisateurs où les participants dans le monde virtuel interagissent entre eux, il est nécessaire de fournir une architecture de communication pour coordonner les activités et les échanges d'information. Les systèmes de développement des applications de réalité virtuelle prennent en charge la gestion de ces aspects basiques pour simplifier le processus de création des nouvelles applications. Dans ce chapitre, nous allons présenter les systèmes qui sont, à notre avis, les plus significatifs des travaux réalisés dans le domaine et que nous pouvons classer en deux grandes catégories : les systèmes boîte à outils (toolkits) et les architectures. Les systèmes boîte à outils fournissent des outils et des bibliothèques de programmation (API) pour le développement et la gestion de l'interaction d'un environnement virtuel. Dans cette catégorie se trouvent les systèmes AreVi, Minimal Reality Toolkit et World Toolkit[2]. Les architectures, elles, fournissent un système entier qui réalise les tâches basiques d'un environnement virtuel. Ainsi, les détails de programmation de ces tâches, de la synchronisation et le transport de données sont transparents pour le développeur d'une application. Pour créer une nouvelle application avec un système de ce type, l'utilisateur configure et modifie les composants fournis ou les remplace par de nouveaux

[2] Pour faciliter la lecture de ce chapitre, la liste des références concernant les systèmes cités est présentée dans la rubrique « Systèmes de Réalité Virtuelle » du chapitre Bibliographie.

modules. Dans cette catégorie nous trouvons : NPSNET, DIVE, Bamboo, Avango, VRJuggler, Blue-c et VIPER.

Caractéristiques à analyser

La création d'un système de réalité virtuelle comporte plusieurs aspects qui doivent être pris en compte par les systèmes de développement notamment la gestion du monde, les communications, la gestion de dispositifs et la généricité. Dans cette section nous décrivons brièvement ces critères qui nous permettront ensuite d'analyser les systèmes présentés.

Gestion du monde

Un environnement virtuel est la représentation d'un ensemble de données qui évoluent pendant la simulation. Il est possible d'envisager plusieurs possibilités pour gérer ces données et leurs changements. Par exemple, les données à partir desquelles le monde est construit peuvent être stockées dans chaque site, dans plusieurs sites ou dans un serveur central. De la même façon, les objets du monde peuvent être locaux à chaque site, partagés entre tous les utilisateurs ou disponibles seulement pour un utilisateur ou pour un ensemble de participants. Dans le cas où les modifications d'un objet devraient être envoyées à plusieurs utilisateurs, il est nécessaire de définir une façon efficace de communiquer l'information concernant l'objet entre les différents participants afin d'éviter la congestion du réseau. Une solution possible est de dupliquer les données de l'objet dans tous les sites et de communiquer seulement les modifications. Cependant les environnements virtuels peuvent contenir des objets en perpétuel changement donc si chaque modification est envoyée à tous les participants le réseau sera saturé. Une stratégie pour réduire la quantité d'information circulant entre les sites est la technique de *dead-reckoning*. Dans cette technique prédictive, les sites calculent la position probable des objets contrôlés par les autres sites en utilisant les données de position, vitesse et direction de l'objet. Le site qui contrôle l'objet envoie un message de mise à jour lorsque la différence entre la position estimée avec l'algorithme de dead-reckoning et la position réelle excède un certain seuil.

Communications

L'information entre les participants d'un environnement virtuel peut être échangée de diverses façons. Chaque système distribué choisit sa stratégie de communication en accord avec ses besoins de communication (vitesse, nombre de participants, fiabilité) en définissant trois éléments

principaux : protocole de communication, schéma de communication et topologie du réseau. Le standard le plus utilisé pour les transmissions de données est actuellement TCP/IP qui comprend deux protocoles de transport : le Transmission Control Protocol (TCP) et l'User Datagram Protocole (UDP). TCP est un protocole basé sur la connexion qui permet de transmettre l'information entre deux processus s'exécutant dans des machines différentes. TCP est garanti ce qui signifie que les données envoyées par un processus sont délivrées au destinataire dans l'ordre et sans perte. Cette propriété implique que le processus expéditeur peut être bloqué jusqu'à ce que le processus destinataire indique qu'il a reçu les données. UDP est un protocole sans connexion moins fiable qui n'attend pas de réponse. Ensuite, nous avons les schémas de communication : *unicast* (point à point), *broadcast* (diffusion) et *multicast* (diffusion restreinte). Dans le schéma unicast la communication se produit seulement entre deux sites, dans le broadcast un site envoie un message à tous les autres sites et dans le multicast un message est adressé simultanément à un ensemble particulier de sites. La diffusion multicast utilise des groupes de communications auxquels les applications peuvent s'inscrire. Par la suite tout message envoyé au groupe est reçu par toutes les applications inscrites. Ainsi, seules les applications qui sont intéressées reçoivent l'information ce qui représente un gros avantage par rapport à la diffusion broadcast.

Pour finaliser la configuration des communications, le développeur doit choisir la topologie de connexions des applications. Il existe deux catégories principales de topologies : égal à égal (*peer-to-peer*) et client-serveur. Dans la topologie égal à égal l'ensemble des participants communiquent entre eux et chaque site peut envoyer des messages directement à tout autre site sur le réseau. Dans la topologie client-serveur, les communications entre les participants sont gérées par un serveur auquel ils sont connectés via un lien unicast bidirectionnel. Un système distribué peut aussi choisir une combinaison de ces deux topologies.

Gestion de dispositifs

Plusieurs types de dispositifs sont d'utilisation courante dans les systèmes de réalité virtuelle. Les systèmes immersifs intègrent un dispositif de visualisation pour limiter la perception de l'utilisateur au seul contenu de l'écran tels les casques de visualisation ou Head-Mounted-Display (HMD). Pour créer l'illusion de profondeur dans un casque de visualisation, les images projetées dans l'écran de l'œil gauche sont légèrement différentes de celles de l'œil droit. L'effet d'immersion peut être augmenté en ajoutant au casque un capteur de localisation. Dans ce cas, le capteur déclenche le rafraîchissement des images projetées pour refléter les changements du point

de vue. Pour avoir des environnements de visualisation immersive plus larges, certains systèmes utilisent la projection vidéo. Le CAVE est un environnement d'audio et vidéo 3D de haute résolution. Les images stéréo du monde virtuel sont projetées sur les écrans qui forment les murs et le sol du CAVE. La visualisation et l'interaction se réalisent via des lunettes stéréoscopiques et des capteurs de position. Lorsque l'utilisateur se déplace, les images projetées sont mises à jour en accord avec les changements. Pour l'utilisateur d'un CAVE les écrans de projection deviennent transparents, le monde virtuel semble s'étendre à l'infini. D'autres systèmes utilisent un seul écran de visualisation pour permettre à plusieurs utilisateurs de « participer » au monde virtuel. Les systèmes à écran ont des interfaces plus limitées : le champ visuel occupé par le monde virtuel coïncide avec les dimensions de l'écran utilisé. La visualisation peut être améliorée par le port de lunettes stéréoscopiques permettant de créer un effet de profondeur.

Figure 1.2.1 : Interface de réalité virtuelle utilisant un casque de visualisation

Les systèmes de réalité virtuelle peuvent également comporter des dispositifs d'interaction spécifiques comme des gants de données (data-gloves) dotés de capteurs de position qui permettent de suivre la position des doigts et de leurs articulations pour se déplacer ou actionner les éléments de l'environnement virtuel. Il est possible aussi d'ajouter des dispositifs sonores ou de retour d'effort pour augmenter la « présence » de l'utilisateur dans le monde simulé. La recherche dans le domaine des interfaces utilisateur se poursuit de façon intensive il est donc très important de permettre aux développeurs d'intégrer et d'évaluer différents dispositifs dans leurs systèmes de façon simple.

Généricité

Les outils logiciels et matériels évoluent constamment, il est très important pour les utilisateurs de travailler avec des systèmes flexibles, et modifiables facilement pour permettre l'addition de nouvelles caractéristiques et fonctionnalités. Un système de développement doit donc détenir certaines qualités au niveau génie logiciel, que nous avons regroupé sous le concept de **généricité** : le système doit être *flexible* pour s'adapter tant aux différentes applications qu'aux différentes configurations d'une application donnée, *modulaire* pour permettre la modification ou l'addition des fonctionnalités d'une façon simple et enfin, *facile à utiliser*, les utilisateurs n'ayant pas besoin pour travailler de connaissances approfondies sur le système sous-jacent. Nous analyserons la généricité d'un système par rapport aux plate-formes logicielles et matérielles dans lesquelles il peut être exécuté et aussi par rapport aux changements réalisables sur une application (changements de configuration, de données ou de dispositifs).

Ensuite nous présentons chaque système et l'approche qu'il a suivie pour gérer les différents aspects impliqués dans le développement d'un système de réalité virtuelle.

AVANGO

Avango est un environnement de développement d'applications de réalité virtuelle distribuées, construit par le GMD (German National Research Center for Information Technology). Ce système fournit l'infrastructure nécessaire pour distribuer les applications d'une façon transparente pour le développeur.

Gestion du monde

Avango définit deux catégories d'objets : les nœuds (nodes) et les capteurs (sensors). Les nœuds permettent la création du graphe de scène pour la représentation et le rendu de la géométrie. Les capteurs sont utilisés pour importer les données des dispositifs dans une application. Avango utilise des « champs » pour maintenir les attributs d'état d'un objet. Les champs encapsulent les types de données basiques et fournissent une interface générique de « streaming » ce qui signifie qu'il existe pour chaque champ des méthodes qui permettent la sérialisation et la reconstruction de la valeur du champ dans un flux de données. Les champs des objets Avango peuvent être connectés entre eux pour créer des réseaux de données, i.e. si le champ A est connecté au champ B, le champ A recevra la valeur de B lorsqu'elle changera. Les connexions entre champs

permettent de définir des comportements très complexes d'une façon simple et de réaliser des associations additionnelles entre les nœuds qui ne peuvent pas être exprimées en termes de graphe de scène standard. Ce procédé de connexion entre champs est analogue à celui défini par le langage VRML97 [VRML] en utilisant le concept de « route ».

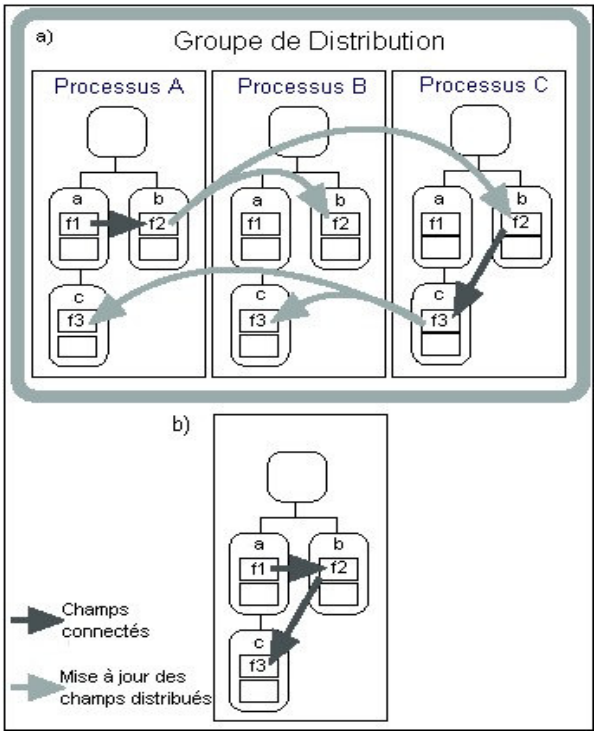


Figure 1.2.2 : Connexions entre les champs des nœuds Avango.
Le graphe d'un monde distribué (a) a le même comportement qui celui d'un monde simple avec les mêmes connexions (b)

Pour les applications distribuées, Avango utilise la réplication d'objets avec un modèle de mémoire partagée distribuée. La mémoire partagée dénote une région locale de mémoire qui peut être utilisé par plusieurs processus simultanément. Un processus lié à la mémoire partagée verra tous les changements que les autres processus réalisent sur elle. Le concept de mémoire distribuée partagée (MDP) étends ce concept aux processus distribués sur un réseau. Chaque processus maintient une copie locale de la MDP qui est synchronisée avec les changements réalisés par les autres processus. Pour implanter ce concept, Avango utilise des *groupes de distribution* auxquels les processus peuvent se joindre. Les objets peuvent être créés localement ou dans un des groupes de distribution. Un objet local existe uniquement pour le processus qui l'a crée alors qu'un objet créé dans un groupe de distribution est un objet distribué : il existe une copie de lui dans chaque processus appartenant au groupe. Les modifications réalisées sur un objet distribué sont envoyées à tous les processus du groupe. Les objets distribués peuvent migrer entre les groupes de distribution mais ils n'appartiennent qu'à un seul groupe à la fois.

Communications

Avango utilise le système Ensemble de l'Université de Cornell pour implanter un modèle de communication multicast fiable entre les processus distribués. Ensemble fournit aussi la bibliothèque Maestro, une collection de classes C++ qui offre au développeur des abstractions de haut-niveau pour les communications de groupe. Maestro permet l'envoi de messages ordonnés afin que chaque membre du groupe reçoive les messages exactement dans le même ordre. Logiquement, cette option introduit un temps de latence additionnel mais est une façon pratique de garantir la consistance des données dans chaque processus. Maestro gère les processus d'un groupe de communication comme une liste de membres, appelée vue. Lorsqu'un membre joint ou quitte le groupe, la vue est mise à jour et ses changements sont envoyés à tous les membres. De cette façon, chaque membre a toujours une liste actualisée de tous les autres membres du groupe. Au moment de rejoindre un groupe, les nouveaux membres ne possèdent pas de copie de l'état partagé de l'application. La communication de l'état est réalisée de manière atomique, toutes les autres communications dans le groupe sont suspendues. Après le transfert de données le nouveau membre aura exactement la même information d'état que les autres membres et les opérations pourront être reprises.

Gestion de dispositifs

Les capteurs d'Avango encapsulent le code nécessaire pour accéder aux dispositifs d'entrée de différents types. Les données produites par un dispositif correspondent aux champs du capteur, donc lorsqu'un dispositif génère de nouvelles données les champs du capteur sont modifiés. Pour incorporer les données dans l'application, il est nécessaire de réaliser des connexions de champs entre les champs du capteur et les champs des nœuds correspondants dans le graphe de scène. Avango fournit les classes pour différents dispositifs comme la souris, le clavier, des capteurs de position et des gants de données.

Généricité

Avango est écrit en C++ et basé sur Iris Performer donc le système est exécutable seulement sur les plate-formes SGI. Il utilise comme langage de script Scheme ce qui permet le prototypage rapide d'applications. Une application Avango est un ensemble de scripts Scheme qui créent des objets Avango, appèlent leurs méthodes, modifient leurs valeurs et gèrent les relations entre eux. Pour obtenir des fonctionnalités plus complexes, il est possible de dériver et étendre les classes C++ qui définissent les objets Avango.

Discussion

Le système Avango est disponible seulement pour des machines SGI donc les applications développées avec ce système ne sont pas portables. Par rapport aux applications distribuées, il est clair que la réalisation des applications est amplement facilitée grâce aux mécanismes fournis par le système. Par contre, quelques problèmes de latence peuvent apparaître étant donné que la communication de données pour les nouveaux participants de la simulation se réalise de façon atomique. Une caractéristique à remarquer est l'utilisation de scripts car elle permet à l'utilisateur de changer de façon dynamique le contenu des mondes, les comportements des objets et les caractéristiques de la visualisation sans modifier et recompiler l'application.

BAMBOO

Bamboo est un outil créé par plusieurs membres du groupe NPSNET pour la recherche et le développement d'environnements virtuels distribués ouverts. Ouvert, dans le contexte de Bamboo, signifie que les systèmes construits pourront se configurer de façon dynamique pendant l'exécution pour acquérir de nouvelles fonctionnalités. Les fonctionnalités sont chargées par l'application en utilisant la philosophie de plug-ins utilisée par des logiciels commerciaux (comme Netscape et PhotoShop).

Gestion du monde

Bamboo consiste en un petit noyau flexible appelé « kernel » qui permet la gestion dynamique du code d'un système, c'est-à-dire le chargement et le déchargement des segments du code exécutable. Pour que cette gestion soit possible, la fonctionnalité du système global doit être divisée en composants ou modules. L'organisation du code en modules dépend entièrement de la conception du système original. Les modules permettent aux différents sites de partager leurs résultats avec les autres sites. Ces résultats représentent typiquement la géométrie, la texture, le son ou le comportement d'un objet dans l'environnement virtuel.

Gestion de dispositifs

Pour la gestion de dispositifs, Bamboo définit une classe qui contient une abstraction des dispositifs qui peuvent être utilisés par le système – y compris les dispositifs standards comme le clavier, la souris et l'écran. Chaque dispositif est contrôlé de manière spécifique par un « thread » pour optimiser son utilisation et augmenter la performance du système. La classe « base » doit

fournir des méthodes de transformation communes à plusieurs dispositifs : un ordre d'avancer doit être exécuté en réponse à un événement spécifique du joystick ou de la souris. Ces méthodes sont conçues pour faciliter l'introduction de nouveaux dispositifs dans des applications existantes.

Communications

Un module peut aussi définir son propre protocole de communication au lieu d'utiliser le protocole standard. Dans ce cas, la seule manière de représenter un utilisateur dans l'environnement est de télécharger d'abord son module client. Ce module doit contenir l'information du protocole et doit prendre en charge l'initialisation et l'interprétation du trafic réseau avec le nouvel utilisateur. Cette approche permettra l'ajout de nouveaux utilisateurs dans des simulations existantes et l'optimisation des communications entre les sites.

L'entrée dans un nouvel environnement se réalise selon le protocole suivant : le système qui arrive annonce son existence avec un message adressé au groupe multicast. Chaque système présent dans l'environnement répond avec un message unicast. Les messages contiennent l'URL de téléchargement du module client et les données de configuration nécessaires à l'initialisation du module. A ce moment-là, tous les systèmes dans l'environnement sont synchronisés.

Généricité

Bamboo est un système gratuit. Il a été développé en utilisant plusieurs API standard (C++, Java, STL, OpenGL) et ACE (Adaptative Communication Environment) une bibliothèque de composants qui facilitent le développement d'applications distribuées portables. Bamboo utilise le Netscape Portable Runtime NSPR pour être portable. Le NSPR fournit la portabilité via une API C++ qui gère les services liés au système d'exploitation comme la gestion de threads, la synchronisation, la gestion du temps et le réseau, entre autres.

Discussion

Les idées de configuration dynamique et de programmation par modules présentes dans la philosophie de Bamboo sont très intéressantes parce qu'elles permettent une communication facile entre n'importe quel ensemble d'applications indépendantes. Cependant, pour arriver à une telle flexibilité Bamboo utilise un grand ensemble d'outils hétérogènes (en niveaux et langages), construits hors Bamboo ce qui augmente la complexité d'utilisation pour le développeur. Le système manque aussi d'une API graphique de haut niveau pour faciliter le développement d'environnements virtuels.

BLUE-C

Blue-c est un projet de recherche de plusieurs instituts appartenant à l'ETH Zurich dont l'objectif est la construction d'un environnement virtuel immersif et collaboratif qui intégrera des images vidéo des utilisateurs. Une application comportera plusieurs portails Blue-c, connectés via un réseau à haute vitesse, qui permettront aux utilisateurs distants géographiquement de se rencontrer, communiquer et collaborer dans un espace virtuel partagé. Le projet Blue-c est un travail en cours dont la première phase sera terminée au printemps 2003.

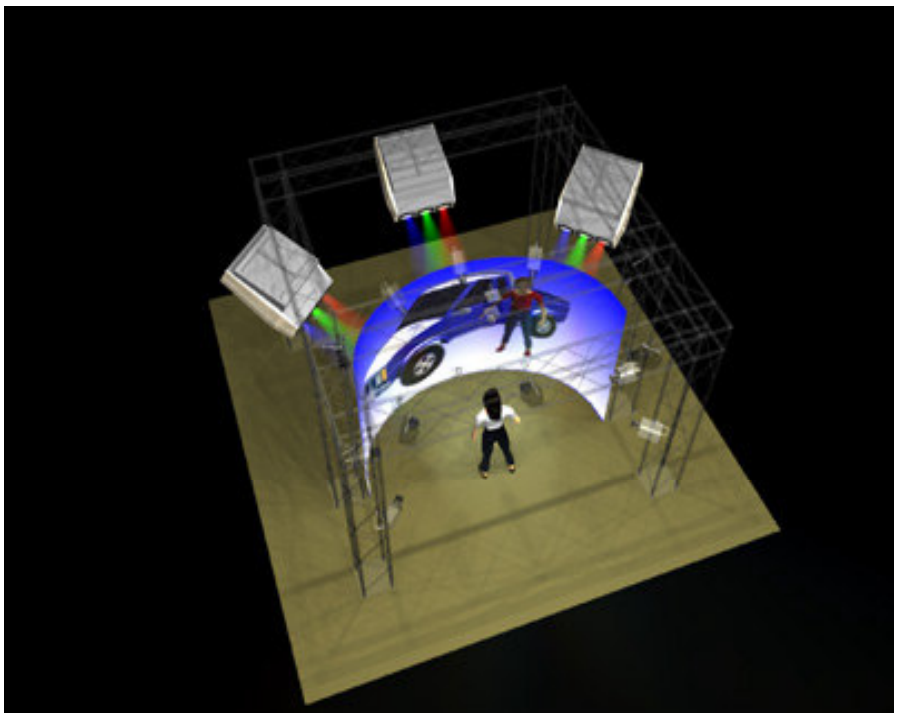


Figure 1.2.3 : Configuration envisagée d'un portail Blue-c

Gestion du monde

Le système JAPE [Staadt01] a été créé pour tester les algorithmes et techniques à utiliser dans le système Blue-c. Pour la gestion de monde, JAPE utilise un graphe de scène, basé sur celui de Iris Performer, pour stocker et afficher l'environnement virtuel. Dans le système JAPE chaque utilisateur possède sa propre vue de la scène et ses dispositifs, par contre le code de l'application s'exécute dans un seul processus qui maintient une copie du graphe de scène.

Un ensemble de caméras situées dans le portail Blue-c capture les images vidéo des utilisateurs à partir desquelles une représentation graphique 3D est construite. La combinaison géométrie-

vidéo est transmise par le réseau aux autres portails Blue-c pour être intégrée dans l'environnement virtuel distribué.

Communications

La complexité des communications du système Blue-c requiert un modèle de distribution avec des caractéristiques de portabilité et de temps-réel pour la transmission efficace de l'information de contrôle et des données multimédia. Pour l'implantation de ce modèle CORBA a été choisi car il permet l'invocation de méthodes distantes et fournit les services de flux de données audio/vidéo et de notification d'événements. Actuellement le module de communication de Blue-c est construit en utilisant le système TAO/ACE, de l'Université de Washington, qui permet l'utilisation de services CORBA pour la localisation de ressources, la synchronisation et la distribution d'événements.

Gestion de dispositifs

A l'heure actuelle, le modèle de gestion et le type de dispositifs qui seront intégrés dans le système Blue-c ne sont pas connus.

Généricité

Le système Blue-c est une plate-forme hétérogène qui utilise un serveur graphique SGI Irix, un cluster de PC Linux pour l'acquisition et le traitement des images de vidéo, et un PC Windows pour le traitement du son.

Discussion

Le système Blue-c est un système en développement qui utilise la technologie des environnements virtuels pour créer un environnement de collaboration entre plusieurs utilisateurs. Etant donné l'importance du matériel requis pour implanter un portail Blue-c, il est possible que l'utilisation du système soit un peu limitée.

DIVE

Le Distributed Interactive Virtual Environment (DIVE), développé par le Swedish Institute of Computer Science (SICS), est une plate-forme pour le développement d'environnements virtuels, d'interfaces utilisateur et d'applications utilisant des environnements virtuels partagés. DIVE est

un système pour la manipulation et l'affichage de mondes virtuels fourni comme un ensemble de bibliothèques de programmation pour le développement d'applications de réalité virtuelle.

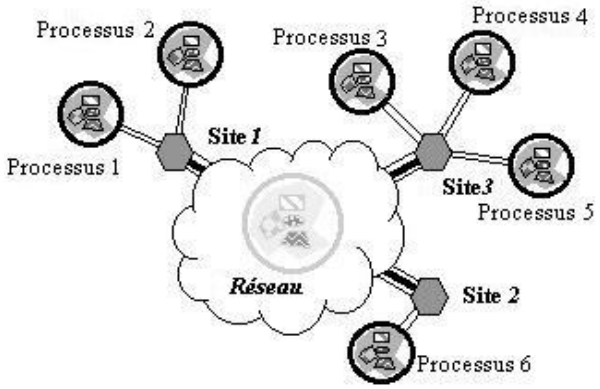


Figure 1.2.4 : Architecture dynamique de DIVE
La figure montre six processus interagissant via un monde unique. Chaque processus maintient une copie des objets relevants.

Gestion du monde

Une entité DIVE est un objet qui maintient des données graphiques, des données de l'utilisateur et des descriptions de comportement autonome. En effet, il est possible de définir le comportement dynamique des objets de l'environnement virtuel par des scripts DIVE/Tcl qui seront déclenchés par des événements du système comme l'interaction avec l'utilisateur, une horloge ou une collision. Dans DIVE tout est un objet, y compris le monde. L'objet monde contient, à la différence des objets ordinaires, un ensemble de paramètres spéciaux comme la couleur du fond du monde et le point où les utilisateurs y entrent. Un monde DIVE est une base de données hiérarchique d'entités dont le nœud racine contient la description complète du monde. Chaque nœud a un parent (sauf la racine) et peut avoir un nombre illimité de nœuds fils.

L'architecture de DIVE est basée sur la réplication active de la base de données dont chaque processus possède une copie. Les modifications de la base de données (ajout, modification et élimination d'objets) sont réalisées localement par chaque site et distribuées ensuite aux autres sites participants pour mettre à jour toutes les copies du monde. Lorsqu'un changement se produit dans le monde, son nouvel état est distribué à toutes les applications DIVE qui l'utilisent. Les mondes DIVE ne sont pas persistants, un monde est « vivant » seulement s'il existe au moins une application DIVE qui l'utilise. Pour ne pas perdre l'état courant du monde quand toutes les applications sont terminées, il est possible de le sauvegarder dans un fichier de définition d'objets qui pourra être utilisé plus tard. Les fichiers de définition d'objets décrivent les mondes DIVE. Ils contiennent la définition des objets du monde (position, géométrie, couleur, texture…) et des

scripts Tcl décrivant leur comportement. Lors de la lecture du fichier, les scripts Tcl sont interprétés. Dans le cas multi-utilisateurs, les fichiers Tcl associés aux objets sont interprétés dans chaque application DIVE. Comme les scripts sont exécutés localement dans la copie de chaque application, le contenu des bases de données locales ne sera pas le même à un instant donné. DIVE tolère cette inconsistance et offre des mécanismes comme le dead-reckoning et les mises à jour dynamiques pour assurer la cohérence des données dans tous les sites. Pour diminuer la charge du réseau associé aux messages d'actualisation, DIVE permet de diviser le monde en régions qui possèdent leur propre groupe de communication et qui seront répliquées et utilisées seulement par les applications intéressées. Ceci permet aux processus qui ne sont pas intéressés par une région d'ignorer les messages du groupe associé et de réduire ainsi le trafic du réseau.

Communications

DIVE utilise une approche égal à égal avec des groupes de communications. La racine de la hiérarchie du monde est associée à un groupe de communication utilisé par défaut pour tous les messages. Lorsqu'un message concernant une entité doit être envoyé, la hiérarchie est parcourue en remontant à partir de l'entité. Si un groupe de communication est trouvé pendant le parcours, il sera utilisé comme moyen de communication à la place du groupe par défaut. Les communications de DIVE sont gérées par Sid2, un module de communication qui offre un service de groupes de communications multicast fiables. Ce module fournit un service fiable en utilisant des acquittements négatifs : lorsqu'un site s'aperçoit qu'un paquet qui lui était destiné est perdu, il le redemande en envoyant un acquittement négatif.

Le *diveserver* maintient une liste des mondes DIVE et des groupes de communication, il est le point central avec lequel les applications DIVE ont besoin de communiquer pour pouvoir accéder à un monde. Le diveserver est un serveur de noms qui traduit le nom d'un monde en un groupe de communication associé à la racine de la hiérarchie. De plus, si un réseau ne possède pas de services multicast, il peut utiliser les services du *proxyserver* pour accéder aux mondes DIVE. Le proxyserver de DIVE agit comme un messager entre les réseaux multicast et les autres réseaux. Il se connecte comme n'importe quelle application au diveserver et renvoie les messages qu'il reçoit des clients dans le réseau non-multicast.

Gestion de dispositifs

Un utilisateur voit un monde à travers une application de rendu appelé « visualizer ». Le visualizer affiche une scène en utilisant le point de vue de l'avatar et il est possible de le configurer pour utiliser différents dispositifs d'interaction comme des casques de visualisation et des gants de

données. Le visualizer lit les données du dispositif d'entrée et fait correspondre les actions physiques réalisées par l'utilisateur aux actions logiques du système DIVE : navigation 3d, sélection et saisie d'objets, etc.

Généricité

DIVE est gratuit pour une utilisation non commerciale. Le système a été développé en C et peut être exécuté sur plusieurs plates-formes : SGI, Solaris, WindowsNT et Linux. Pour développer de nouvelles applications DIVE, il est possible de construire une interface graphique utilisateur en Tcl/Tk pour le navigateur de mondes ***DIVA*** fourni avec le système ou d'écrire une application C qui utilise les fonctionnalités de DIVE à travers l'interface Dive C. Il est possible aussi d'écrire une application qui communique avec DIVE en utilisant les sockets.

Discussion

DIVE permet le développement d'applications portables et générales car il sépare les fonctions basiques de l'environnement virtuel de celles de l'application. Par contre, pour créer une application DIVE complexe, une connaissance très approfondie des mécanismes et du fonctionnement du système ainsi que du développement en Tcl/Tk est nécessaire.

MRTOOLKIT

Le Minimal Reality Toolkit (MR Toolkit), conçu par le groupe de recherche en informatique de l'Université d'Alberta, est un ensemble de logiciels utilitaires pour le développement d'applications de réalité virtuelle. Ce système consiste en une collection de bibliothèques qui gèrent plusieurs aspects d'une application de réalité virtuelle (pilotes de dispositifs, communications, partage de données) et un langage pour décrire la géométrie et le comportement des objets d'un environnement virtuel.

Gestion du monde

Le langage de modélisation d'objets OML (Object Modeling Langage) permet de définir la géométrie et le comportement des objets à utiliser dans un monde virtuel. Un objet en OML contient la géométrie d'un objet, l'information nécessaire à son affichage (couleur, texture, …) et les méthodes qui définissent son comportement en réponse aux événements du système. Le processeur de géométrie de l'interpréteur OML réalise la détection de collisions entre les objets

choisis par le concepteur du monde. Le gestionnaire d'environnement EM (Environment Manager) est une application MR Toolkit écrite en C qui permet au développeur de créer des environnements virtuels à partir d'un ensemble de données OML et d'un fichier de description. Le EM lit le fichier de description qui contient la configuration du monde et collecte l'information sur la configuration de dispositifs, les fichiers d'objets, les instances et les comportements. L'EM peut aussi gérer des mondes multi-utilisateurs et dans ce cas, il est responsable de la configuration du réseau et de la gestion de l'architecture répliquée et des données partagées.
L'EM gère son propre ensemble d'objets et d'instances dont celles partagées avec d'autres mondes dans le réseau. Les versions des autres EM des objets locaux sont appelées fantômes. Pour réduire la communication entre les applications, l'EM transmet uniquement les variables partagées dont l'état a changé et utilise une technique de dead-reckoning pour permettre la simulation locale du comportement des instances partagées. Pour permettre une interaction appropriée avec l'utilisateur, chaque EM possède sa propre copie de la simulation et de l'information partagée.

Communications

Il existe différents types de processus dans une application MR Toolkit. Il existe un processus « maître » qui contrôle les processus « esclaves » et les processus de « calcul ». Les processus esclaves sont créés généralement pour réaliser un rendu additionnel, par exemple l'image gauche d'un casque de visualisation peut être réalisée par le processus maître tandis que l'image droite est réalisée par le processus esclave. Les processus de « calcul » sont crées pour réaliser la simulation et les autres tâches de calcul intensives. Une application MR établie aussi des connexions avec un ou plusieurs processus « serveurs » responsables de la gestion des dispositifs. Le processus maître, démarré en premier, se charge d'initialiser les processus, d'établir les communications avec eux et de réaliser le rendu.

MR Toolkit permet d'une façon limitée la distribution des processus : les processus esclaves, serveurs et de calcul communiquent avec le processus maître mais ils ne peuvent pas communiquer directement entre eux. Dans ce cas, les sockets sont utilisés pour la communication et la synchronisation entre le processus maître et les autres processus.

Le MR Toolkit Peer Package permet la connexion entre deux ou plusieurs applications MR distantes en utilisant des communications UDP. Les processus maîtres des applications peuvent

échanger des données des dispositifs et aussi de l'information spécifique définie par le développeur. Dans les applications multi-utilisateurs développés avec l'EM, la communication est égal à égal. Toutes les demandes de connexion se réalisent de façon asynchrone pour permettre à l'application d'initier la connexion à n'importe quel moment. Le Peer Package et l'EM maintiennent un graphe complet de la topologie de connexion donc chaque site est connecté aux autres explicitement. Chaque site a une liste de tous les autres sites actifs auxquels il distribue un message lorsqu'une nouvelle connexion arrive. Quand un EM souhaite terminer les communications, il envoi un message qui permet aux autres mondes de l'effacer de la liste de sites et d'éliminer les instances et les objets partagés qui lui appartiennent.

Pour résoudre les conflits entre plusieurs participants l'EM fournit deux schémas de contrôle de concurrence. Le premier schéma utilise un jeton, la consistance des données dans le monde virtuel distribuée étant garantie en limitant la manipulation de façon à ce qu'un seul site puisse modifier l'état du monde virtuel à un instant donné. Le processus modifie une entité puis distribue les changements tout en gardant le jeton. Si un site souhaite modifier une entité, il doit demander le jeton avant de réaliser l'opération. Le second schéma utilise la propriété et les droits d'accès, la propriété indiquant qu'un seul site a le contrôle sur une variable ou une instance. Cette caractéristique peut être fixe ou peut changer entre EM si l'application le demande, par exemple, dans un jeu de handball la propriété de la balle peut être transférée entre les joueurs. Ce schéma permet de contrôler les entités partagées : une instance peut être visible des autres EM mais seul son propriétaire peut la modifier. Il est aussi possible d'assigner des droits d'accès à chaque entité partagée. La permission d'écriture permet à tous les sites de modifier une entité partagée et la permission de lecture de lire son contenu.

Gestion de dispositifs

Le MR Toolkit gère les dispositifs des applications en utilisant un modèle client-serveur. Pour chaque dispositif il existe un serveur responsable de l'interaction : les serveurs d'entrée récupèrent les données des dispositifs d'entrée et les serveurs de sortie mettent à jour les valeurs dans les dispositifs de sortie. Le serveur transforme la sortie du dispositif dans un format utilisable par les autres modules de l'application MR. Le côté client du dispositif est une bibliothèque de méthodes utilisée par l'application. Pour utiliser un dispositif, le client établit une connexion au serveur associé et définit les communications et les services additionnels dont il a besoin (par exemple, le filtrage). Lorsqu'un client réalise une connexion, le serveur commence à collecter ou à envoyer des données au dispositif.

Un aspect important dans MR est la gestion des dispositifs disponibles. Pour avoir un système portable qui puisse être facilement reconfiguré pour s'adapter aux différents dispositifs, MR utilise le « servertab » un fichier qui codifie toutes les dépendances d'un dispositif. Dans le servertab se trouvent tous les serveurs MR qui sont en train de s'exécuter dans un site. Lorsqu'une application MR démarre, elle utilise le servertab pour déterminer comment se connecter aux serveurs et accéder ainsi à tous les dispositifs disponibles.

Le MR Toolkit prend en charge plusieurs dispositifs d'interaction parmi lesquels différents systèmes de repérage (Bird, Flock of Birds, souris Logitech 6D), de casques de visualisation (VPL EyePhone 1, Virtual Research Flight Helmet, Virtual I/O I.Glasses), de gants de données (VPL DataGlove, Virtual Technologies CyberGlove) et de dispositifs avec retour d'effort. Le système permet aussi le développement de modules de contrôle de nouveaux dispositifs.

Généricité

Le MR Toolkit est gratuit pour des institutions éducatives et de recherche. Le système est disponible pour systèmes Unix principalement mais il existe des composants disponibles pour Windows. Une application MR peut être écrite en C, C++ et FORTRAN 77 et utiliser les systèmes graphiques PHIGS et OpenGL. L'application appelle les méthodes des bibliothèques MR pour réaliser la configuration initiale, démarrer les différents processus et initialiser les dispositifs.

Discussion

Le MR Toolkit, le gestionnaire d'environnement EM et le langage OML fournissent un ensemble d'outils de haut-niveau pour la création d'applications de réalité virtuelle. Dans le fichier de description utilisé pour créer une application, il est possible de spécifier les dispositifs qui vont être utilisés ce qui permet de changer de configuration assez facilement. Le principal désavantage de MR se trouve au niveau des communications car l'implantation de l'architecture égal à égal est réalisée en utilisant des messages point à point. Ceci peut diminuer l'efficacité du système si l'on tient compte des communications déjà instaurées pour la communication entre processus et entre les dispositifs et le système.

NPSNET-IV

En 1984, le D-ARPA (Defense Advanced Research Projects Agency) a créé le projet de simulation distribué SIMNET (SIMulator NETworking). Ce projet - conçu pour l'entraînement de militaires - utilisait plusieurs dizaines de simulateurs de véhicules connectés en réseau, proposait des adversaires intelligents simulés et réalisait l'analyse des résultats des simulations. Les expériences accumulés du développement de SIMNET ont été utilisés par le NPSNET Research Group (NRG) pour développer le système NPSNET, un projet qui cherche à étendre les capacités de la simulation et de l'interaction dans des environnements virtuels pour gèrer un grand nombre d'utilisateurs (plus de 1000). Dans le contexte militaire, un environnement virtuel de cette taille permet le partage d'un même champ de bataille entre participants géographiquement dispersés. Il existe plusieurs générations du système NPSNET : une démonstration de NPSNET-1 a été réalisée pendant la conférence ACM SIGGRAPH 91. NPSNET-2 et 3 ont été conçus pour explorer des techniques plus avancées pour le graphisme et pour augmenter la taille des bases de données de terrain. NPS-Stealth, dérivé de NPSNET-1, avait comme but le développement d'un système capable d'interpréter les données de terrain et les protocoles de réseau de SIMNET. NPSNET-IV a été construit en 1993 pour utiliser le système graphique Performer de SGI et le standard DIS (Distributed Interactive Simulation). Le protocole DIS est constitué d'un ensemble d'unités de données (PDU) qui transportent des informations d'état et des événements de la simulation. En utilisant ce protocole, des simulateurs avec des plates-formes logicielles et matérielles différentes peuvent partager un même environnement virtuel.

Gestion du monde

Le modèle du monde de NPSNET est une base de données répliquée des objets de la simulation. Dans le monde réel les entités ont une zone d'intérêt limitée et spécifique. Par exemple dans le cas d'un champ de bataille, un tank peut voir et affecter les entités qui se trouvent dans un rayon de 4 kilomètres alors que dans un jeu de football, un joueur de football a une zone d'intérêt d'une dizaine de mètres. La zone d'intérêt (Area of Interest, AOI) qui entoure une entité de NPSNET est définie par un ensemble de six cellules hexagonales. Chaque cellule est associée à un groupe multicast donc une entité est membre de sept groupes de communications. Le site propriétaire de l'entité écoute les sept groupes mais n'émet de PDUs que dans le groupe associé à la cellule dans laquelle l'entité est située.

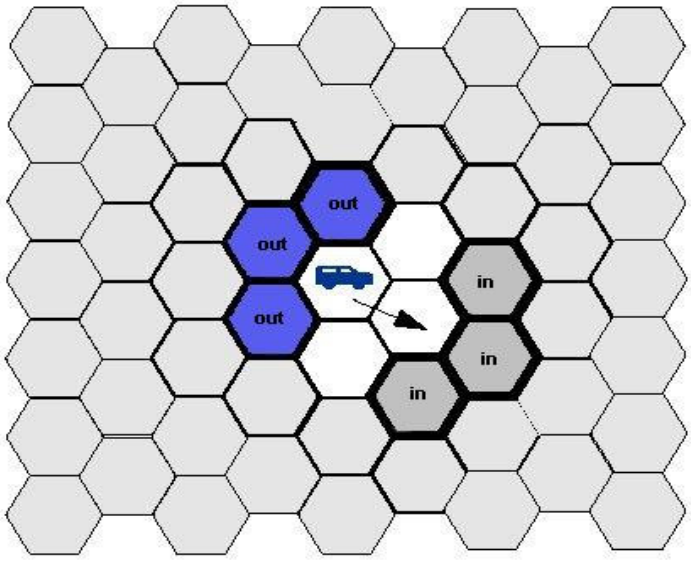


Figure 1.2.5 : Un véhicule NPSNET et sa zone d'intérêt (AOI). Le véhicule doit rentrer et sortir des trois groupes associés aux cellules entourant son AOI. Les cellules 'out' sont enlevées et les cellules 'in' sont ajoutées lorsque l'entité se déplace dans une nouvelle cellule.

Communications

NPSNET-IV utilise le paradigme de simulation distribuée connu sous le nom de « joueurs et fantômes » (players and ghost). Dans cette technique, chaque entité est contrôlée à partir de son site hôte par un objet logiciel appelé Joueur. Sur les autres sites participants à la simulation, une version du Joueur est modélisée via un objet logiciel appelé Fantôme. Les objets Fantômes modifient leur état en utilisant un algorithme d'estimation (dead-reckoning). Le Joueur utilise ce même algorithme pour pouvoir comparer son état estimé et son état réel. Lorsque la différence entre les deux états excède un certain seuil, un message de mise à jour est diffusé à tous les autres sites. Les sites utilisent cette information pour mettre à jour les Fantômes et reprendre les estimations à partir des nouvelles valeurs de position et d'orientation. Une entité dans un groupe de communication peut être un membre passif ou actif. Les membres actifs, localisés dans la cellule associée au groupe, envoient et reçoivent des PDUs et peuvent devenir le responsable de la cellule. Les membres passifs d'un groupe sont les entités qui ne sont pas dans la cellule mais dont l'AOI la contient. Ils n'envoient de PDUs que lorsqu'ils entrent ou sortent du groupe. Quand une entité entre dans un nouveau groupe, elle enregistre le temps d'entrée et envoie une requête (Join Request PDU) au groupe. Le responsable du groupe (le membre le plus ancien) lui envoie un message de réponse (Pointer PDU) et puis, il envoie à tous les membres du groupe un PDU contenant un pointeur vers lui-même ou vers une autre entité active. Le nouveau membre envoie une requête de données (Data Request PDU) à l'entité référencée qui lui répond avec

l'information sur l'ensemble des entités actives. Les sorties du groupe sont annoncées avec une requête de sortie (Leave Request PDU). Le premier membre actif d'un groupe doit envoyer plusieurs requêtes d'entrée avant de conclure qu'il est le seul membre du groupe et par conséquent le plus ancien.

Les participants d'une simulation NPSNET-IV communiquent avec les autres environnements virtuels en utilisant IP Multicast et l'Internet Multicast Backbone, MBone. Le MBone est un réseau virtuel qui simule l'IP Multicast sur Internet, il est virtuel parce qu'il utilise le même médium physique que l'Internet. MBone fournit un outil appelé le Session Directory (SD), qui permet aux applications comme NPSNET de trouver de façon automatique une adresse multicast libre pour une nouvelle séance d'exécution.

Gestion de Dispositifs

NPSNET-IV peut être configuré pour simuler un véhicule (aérien, terrestre, nautique ou virtuel) ou un humain. L'utilisateur contrôle le véhicule avec le dispositif d'interaction qu'il a choisi parmi les interfaces offertes par le système. Le système modélise le mouvement du véhicule en accord avec ses caractéristiques, par exemple, le mouvement sur la surface de l'eau si le véhicule simulé est un bateau. Les autres véhicules dans la simulation sont contrôlés par d'autres utilisateurs ou par des entités autonomes. NPSNET-IV prend en charge plusieurs dispositifs d'entrée parmi lesquels les périphériques de simulation de vol Thrust Master Flight Control System et Kinney Aerospace Flight Set.

Généricité

Le système NPSNET-IV utilise une plate-forme matérielle et des logiciels spécifiques aux machines SGI, ce qui empêche son exécution dans des environnements plus répandus comme les PC. Il a été développé en C++ et utilise les bibliothèques de développement Performer pour la visualisation.

NPSNET V

Le système NPSNET-IV est un système versatile qui a servi de base à plusieurs projets dans la Naval Postgraduate School. Tous ces projets ont ajouté de nouveaux modules au code existant donc, pour y ajouter des améliorations, il est nécessaire de se familiariser pendant plusieurs mois avec la structure monolithique du système. L'objectif de la nouvelle version de NPSNET est de construire une plate-forme de développement pratique en utilisant une architecture par

composants et le langage de programmation Java [Capps00]. NPSNET-V permet l'addition de composants pendant l'exécution d'une application pour lui donner de nouvelles fonctionnalités. Ces composants peuvent être chargés à partir d'un disque local ou du réseau. Cette capacité permet aussi aux applications de découvrir de nouveaux types d'entités. La première fois qu'une application rencontre une entité, elle peut télécharger sa description et en créer une instance. De cette manière, les applications participantes augmentent leur fonctionnalité de façon dynamique pour gérer les nouvelles entités et les nouveaux environnements dès qu'ils sont disponibles. Grâce aux caractéristiques de portabilité de Java, les développeurs peuvent, en suivant une interface basique, écrire des entités et des comportements de façon indépendante pour les combiner ensuite dans un même environnement virtuel.

Discussion

NPSNET-IV offre un environnement efficace pour la construction d'environnements virtuels distribués à grande échelle. Les environnements NPSNET-IV peuvent intégrer différents media (audio, vidéo, images), des agents autonomes, de la simulation physique temps-réel, et des protocoles de réseau comme IP Multicast. Cependant il manque de généralité car il est construit pour des simulations de bataille et de portabilité parce qu'il est seulement disponible pour l'environnement SGI avec des dispositifs prédéfinis. Par ailleurs, le système est monolithique et peu flexible ce qui rend très long et difficile le processus de création des nouvelles applications. Le Naval Postgraduate School a donc décidé de développer NPSNET-V pour fournir un système portable et extensible tout en suivant une architecture par composants. Ce système est en plein développement et veut être un outil pour la recherche en environnements virtuels distribués plus pratique et accessible que ses prédécesseurs.

oRis-ARéVi

Les systèmes oRis et ARéVi ont été développés par le Laboratoire d'Informatique Industrielle (LI2) de l'Ecole Nationale d'Ingénieurs de Brest (ENIB). Le système oRis est un environnement de simulation interactive : c'est un langage de programmation par objets et un environnement d'exécution. Ces caractéristiques en font une plate-forme généraliste pour l'implémentation de systèmes multi-agents, plus particulièrement dédiée à la simulation. ORis permet l'intervention en cours de simulation pour observer le système multi-agents, interagir avec les agents ou sur l'environnement et les modifier en ligne. Le système ARéVi (Atelier de Réalité Virtuelle), est une

boîte à outils pour créer des applications de réalité virtuelle distribuée. Le noyau d'ARéVi est le système oRis, étendu par du code C++ offrant des fonctionnalités propres à la réalité virtuelle.

Gestion du monde

Dans ARéVi, les entités graphiques peuvent être réparties sur différentes machines. Lorsqu'une entité est créée sur une machine, des copies ayant un comportement cinématique dégradé sont créées sur les autres machines. La mise à jour des caractéristiques géométriques et cinématiques des copies est réalisée lorsqu'une divergence trop importante est estimée entre leur situation et la situation réelle (dead-reckoning).

Communications

La communication entre entités distantes et le mécanisme de dead-reckoning présents dans ARéVi ont été mises en place en utilisant les fonctionnalités en matière de communication par réseau du système oRis. Des connexions TCP ou UDP peuvent donc être établies, en utilisant les classes TCPListener, TCPConnection et UDPTransmitter. Cette dernière classe propose les services d'envoi et réception de messages textuels ou binaires en point à point ou par diffusion sur le réseau local.

Gestion de dispositifs

ARéVi gère des périphériques variés tels un gant de données, une manette de commande, un volant, des capteurs de localisation, un bras à retour d'effort et un casque de vision stéréoscopique. Pour ce qui est du domaine sonore, ARéVi propose une sonorisation spatiale et des fonctionnalités de synthèse et de reconnaissance vocale.

Généricité

Les premières versions d'ARéVi ont été réalisées sous IRIX sur machines Silicon Graphics et utilisaient la bibliothèque graphique Open Inventor. Les dernières versions d'ARéVi utilisent, en revanche, des machines de type PC avec des cartes graphiques car elles permettent d'obtenir de bonnes performances graphiques à un coût d'achat et de maintenance raisonnable. Le système oRis, réalisé sous UNIX, a été adapté pour pouvoir être intégré dans la version d'ARéVi s'exécutant dans les environnements Windows à l'aide de la bibliothèque OpenGL Optimizer. Le système ARéVI a été utilisé dans le cadre du prototypage interactif d'une cellule d'emboutissage et du développement d'une plate-forme de formation pour la sécurité civile.

Conclusion

ARéVi a subi plusieurs évolutions (ARéVi 3.0, ARéVi 4.0). A ce jour, des travaux concernant une refonte totale du système pour favoriser le portage sur différents systèmes ont été entrepris. Une fois opérationnelle, la plate-forme sera intègre au système oRis.

VIPER

VIPER est une plate-forme générique orientée-objet, développée à l'IRIT, qui permet la gestion d'environnements virtuels multi-utilisateurs et, plus généralement, le développement d'applications de réalité virtuelle distribuées. Le but de ce système est de proposer un modèle de calcul réparti générique permettant de gérer n'importe quelle application pouvant être modélisée en terme d'échanges (symbolisés par des **stimuli**) entre des **entités**, dans un **univers virtuel**.

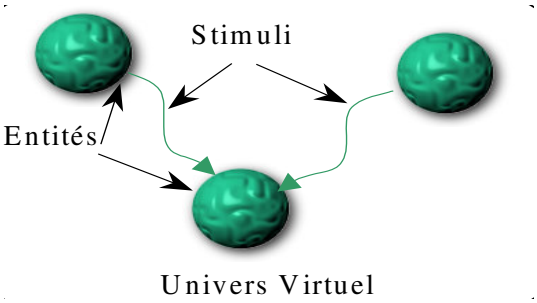


Figure 1.2.6 : Structure d'un environnement virtuel VIPER.
Les stimuli sont les véhicules des interactions entre les entités. L'univers virtuel représente le milieu tridimensionnel dans lequel les entités coexistent et interagissent.

Gestion du monde

Les entités de VIPER sont utilisées pour gérer de façon uniforme les décors des mondes virtuels, les objets virtuels et les **clones**. Un clone est un objet associé à un utilisateur, à une application ou à un robot, qui leur permet de participer à la simulation comme s'ils étaient réellement présents dans l'univers virtuel; le clone d'un utilisateur est généralement appelé **avatar.** Les entités sont autonomes et possèdent un ensemble de propriétés et de comportements. Elles sont conceptuellement regroupées en familles. Une famille est une population d'entités qui possèdent les mêmes propriétés et les mêmes comportements.

Les interactions entre les entités transitent par un média qui permet des communications entre plusieurs entités simultanément. Une entité reçoit les modifications de son environnement grâce à des capteurs et peut agir sur cet environnement grâce à des effecteurs. Pour chaque type d'interaction VIPER propose une modélisation par quatre types d'éléments : capteur, effecteur, stimulus et espace de stimuli.

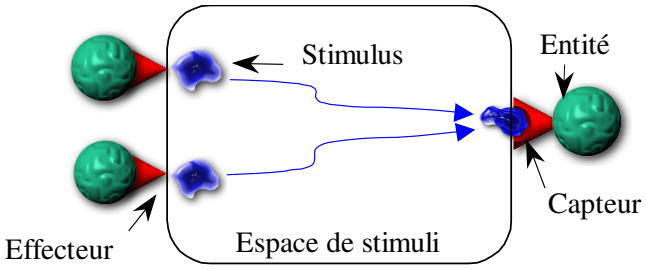


Figure 1.2.7 : Mécanisme de communications inter-entités en VIPER

Un stimulus est un phénomène ou un événement perceptible par une entité (par exemple une forme graphique, un son ou une collision). Un espace de stimuli est un milieu dans lequel s'échangent les stimuli d'un certain type. Il peut être défini comme une projection de l'univers virtuel selon un axe propre à un stimulus (espaces des formes graphiques, espaces des sons...). Un capteur reçoit les stimuli que l'entité est capable de percevoir (formes visibles par l'entité, sons proches...). Par l'exécution d'une action, un effecteur produit un stimulus que d'autres entités percevront. Une entité possède un ensemble de capteurs et d'effecteurs qui lui permettent d'interagir avec son environnement. Une entité est, de plus, dotée d'un certain nombre de comportements qui modifient son état interne et commandent des actions à ses effecteurs.

Pour distribuer l'environnement virtuel VIPER définit des univers virtuels distribués (UVD) sur lesquels les entités sont reparties. Chaque type d'UVD est un agrégat d'entités. Il définit une fonction de nommage d'entité qui attribue à chaque entité un identificateur unique sur le réseau. Un UVD définit aussi la fonction de répartition qui permet, à partir d'un identificateur d'entité, de trouver le site qui gère l'entité à un instant donné. D'autre part, certains UVD offrent des méthodes permettant d'émettre et de recevoir des entités via le réseau. Il existe plusieurs types d'UVD définis par VIPER :

- Les univers virtuels distribués passifs. Ceci est la classe la plus simple d'UVD qui contient des entités qui sont liées à leur site de création et ne peuvent pas migrer. Il permet la

création dynamique d'entités sur les différentes machines gérées à un instant donné par VIPER. Ce type d'UVD permet de gérer les avatars car ils sont obligés de rester sur la station de travail de l'utilisateur qu'ils représentent.

- Les univers virtuels distribués actifs. Ce type d'UVD permet aux entités de migrer d'un site à l'autre. Ainsi, la migration peut être utilisée pour « rapprocher » deux entités qui interagissent. Par exemple, si un utilisateur manipule un objet virtuel, l'UVD va faire migrer l'entité représentant l'objet sur le site de l'avatar de l'utilisateur. De cette façon l'interactivité du point de vue de l'utilisateur sera grandement augmentée.
- Les univers virtuels dupliqués. Dans les univers virtuels dupliqués toutes les entités sont dupliquées sur chaque site. Les comportements sont exécutés sur chaque site et VIPER s'occupe de la synchronisation entre les états des entités dupliquées.

L'espace de stimuli dans le cadre distribué est un objet distribué qui gère de façon autonome les interactions entre les entités. Il traite uniformément les interactions locales à un site et les interactions impliquant des entités qui sont gérées par des sites distants. Lorsque plusieurs avatars évoluent ensemble dans un même monde virtuel, chaque utilisateur doit avoir conscience de la présence des autres utilisateurs : il doit pouvoir voir, entendre (et éventuellement toucher) les avatars représentant les autres utilisateurs, et doit percevoir les modifications du monde virtuel réalisées par eux. Pour gérer ces interactions, VIPER utilise un type particulier d'espace de stimuli distribué, appelé espace dupliqué. Ceci est un objet distribué contenant un ensemble de stimuli et qui lors de la modification de l'un de ces stimuli maintient la cohérence entre la version originale et les copies figurant sur chaque site. Cette cohérence est assurée par l'envoi de messages de mise à jour de façon transparente. L'envoi de messages est typiquement réalisé via un groupe de communication privé à l'espace de stimuli dupliqué.

Pour gérer des environnements virtuels complexes comportant de nombreuses entités, VIPER utilise des techniques permettant de limiter le nombre de messages de mise à jour émis sur le réseau ainsi que le nombre de sites participants. Ces techniques utilisent le fait que deux entités ne peuvent se voir que si elles sont assez proches. L'exploitation de cette propriété de localité de la perception peut résoudre certains des problèmes induits par la présence d'un grand nombre d'entités dans un même monde virtuel. VIPER utilise un découpage spatial proche de celui de NPSNET IV, il associe à chaque région du monde un groupe de communication. Par la suite, les modifications de chaque forme sont envoyées sur un seul groupe de communication à un moment donné (celui correspondant à la région dans laquelle l'entité est localisée) et chaque

entité qui veut recevoir des stimuli s'abonne aux groupes de communication correspondant à la zone qui l'intéresse (c'est-à-dire qu'elle peut observer).

Pour déterminer la zone d'intérêt (ensemble de régions) d'une entité, VIPER utilise un algorithme de découpage de l'environnement en cellules reliées par des portails. Chaque cellule est représentée par un volume polyédrique convexe (qui doit être disjoint des polyèdres représentant les autres cellules et qui doit permettre de déterminer si un point est à l'intérieur du volume) et chaque portail est représenté par un polygone transparent quelconque (qui peut être concave). Une cellule ne peut être « vue » depuis une autre cellule qu'au travers de portails qui correspondent aux diverses portes et fenêtres qui les relient (directement ou indirectement). Le traitement rajouté pour gérer la limitation des échanges entre les différents sites, consiste à déterminer entre deux étapes quelles sont les cellules qui sont désormais visibles et celles qui sont désormais invisibles. Puis, pour toutes les cellules visibles, le site gérant l'entité considérée s'abonne aux groupes de communication des cellules. Symétriquement, le site se retirera des groupes de communication des cellules désormais invisibles. Etant donné que les messages de mise à jour de la forme d'une entité sont diffusés sur le groupe de la cellule qui la contient, seules les mises à jour des formes visibles par les entités locales seront reçues par un site donné. De plus, lorsque le site s'abonne à un groupe, il va demander (en envoyant une requête au groupe) l'état actuel de toutes les formes de la cellule. En effet, même si le site était précédemment connecté à ce groupe, depuis la dernière connexion les formes ont pu évoluer. Les réponses à cette demande sont envoyées par les sites qui gèrent les entités possédant les formes concernées. Actuellement, ces réponses sont envoyées sur le groupe correspondant à la cellule et permettent ainsi une remise à jour de tous les sites connectés (ceci peut être utile si un précédent message de mise à jour a été perdu).

Communications

L'architecture de communication adoptée par VIPER est essentiellement de type égal à égal. La base de données représentant l'environnement virtuel est distribuée : l'ensemble des entités est partagé entre les différents sites et les formes et les icônes sonores sont dupliquées. Le modèle de communication utilise aussi bien des diffusions de groupe que des communications point à point. VIPER utilise la communication par groupe (multicasting IP) partout où elle est disponible. Néanmoins, l'architecture peut utiliser un serveur, principalement lors de la connexion pour obtenir des informations sur les groupes de communication utilisés par un monde virtuel. D'autre part, ce serveur peut permettre d'obtenir des identificateurs uniques pour les types de stimuli qui

peuvent être échangés dans les mondes virtuels. Ce serveur peut aussi servir à gérer une procédure d'authentification pour protéger l'accès à un environnement virtuel. Enfin, le serveur peut gérer la connexion de clients qui n'ont pas accès aux services de multicasting.

Gestion de dispositifs

Dans VIPER, la classe capteur sert à la communication entre le monde réel et l'environnement virtuel. VIPER définit les capteurs qui encapsulent les périphériques de réalité virtuelle gérés par WorldToolKit. Il existe, par exemple, des capteurs qui gèrent les périphériques servant à calculer la position d'une partie du corps d'un utilisateur (comme le Fastrak de Polhemus) ainsi que des capteurs gérant les divers gants de données existants. Etant donné que WTK gère lui-même ces dispositifs, les capteurs définis par VIPER sont très simples et ne font que faire appel aux fonctions de WTK. Néanmoins, ces différents capteurs permettent au système de rester indépendant de la bibliothèque utilisée.

Généricité

VIPER suit un développement orienté-objet, il existe des versions de VIPER écrites en Java et en C++ dont l'interopérabilité est actuellement à l'étude. Pour répondre aux spécificités des diverses applications, de nombreux choix peuvent être faits parmi les optimisations proposées par VIPER. Le système offre donc aux développeurs deux niveaux de programmation. Le premier niveau masque totalement l'aspect réparti de l'application : le programmeur définit de nouvelles entités (en définissant leurs comportements et leurs attributs), de nouveaux stimuli, de nouveaux capteurs et de nouveaux effecteurs comme s'il travaillait avec une hiérarchie normale de classes utilisée pour réaliser un programme non distribué. La distribution des données et des calculs est cachée dans les espaces de stimuli et les univers virtuels utilisés. Le second niveau permet de redéfinir les mécanismes de répartition. Ce niveau est notamment utilisé pour définir des filtres pour limiter l'utilisation de la bande passante réseau où pour mettre en place des algorithmes de prédiction (dead-reckoning). A ce niveau, le développeur peut créer de nouveaux espaces de stimuli et de nouveaux univers virtuels qui permettent de gérer la répartition de stimuli et d'entités. Le programmeur est ainsi libre d'optimiser l'application à sa convenance.

Pour le rendu graphique et sonore, VIPER utilise principalement WorldToolKit , cependant, les aspects graphiques sont suffisamment découplés des aspects répartis pour que le changement de bibliothèque soit aisé. Les mondes virtuels de VIPER, les avatars et les autres objets que peut apporter un site lorsqu'il rejoint le monde virtuel sont définis dans des fichiers VRML. VIPER est

générique par rapport aux plates-formes matérielles et par rapport aux applications qu'il peut gérer. Il peut gérer plusieurs types de mondes possédant diverses lois physiques.

Pour créer des comportements, VIPER propose plusieurs techniques : des comportements compilés dans le système (via la définition de nouvelles classes d'entités), des comportements interprétés (en utilisant le langage ObjectTcl) et des comportements liés dynamiquement (les nouvelles classes d'entités sont placées dans des plugins : DLL ou DSO). Enfin, VIPER offre la possibilité de changer dynamiquement les comportements pendant l'exécution (comportements interprétés et liés dynamiquement).

CAVALCADE

CAVALCADE est un outil de prototypage virtuel collaboratif qui a été développé par CS SI, l'IRIT et le CRS4. CAVALCADE est employé par les utilisateurs finaux pour plusieurs activités de conception de voiture, d'engin spatial, de train, de bâtiments. Le système repose sur les concepts de VIPER pour la modélisation de l'univers virtuel et pour le modèle de communication. Les deux composants principaux pour définir un environnement virtuel sont les entités qui encapsulent les objets graphiques et leurs comportements et le stimulus qui est la base de l'interaction et de l'échange de l'information entre les entités. Les entités de type avatar ont été introduites pour agir en tant qu'interfaces entre l'environnement et les utilisateurs virtuels dans le but de simplifier la définition des modèles de distribution. Les entités autonomes mènent à une encapsulation parfaite du comportement et de l'état d'une entité, et facilitent donc leur distribution : une telle entité peut exécuter son comportement sur n'importe quel site en communiquant avec d'autres entités par des stimuli bien définis.

Dans sa version de base, CAVALCADE définit les entités suivantes : le constructeur, l'avatar (qui gère entre autres l'interface homme-machine), l'expert et plusieurs prototypes. Ces entités reprennent les schémas définis dans VIPER. Le constructeur a pour but de fournir à l'utilisateur des commandes de haut niveau. Il contrôle le processus d'interaction (dialogue multimodal) et les fonctions spécifiques de l'application. L'expert vise à examiner le prototype pour en assurer la cohérence : poids, puissance énergétique, intégrité de construction. Cette entité doit contenir la connaissance d'un expert et la spécificité industrielle du domaine. Les prototypes sont également conformes au modèle d'entité. Ainsi, une entité 3D inclut plusieurs modules comportementaux tels que la dynamique pour simuler des forces et les couples appliqués aux entités et permet à l'utilisateur d'accéder à de la documentation en ligne associée grâce à un lien stocké dans le

prototype, dirigeant vers une page HTML. Les composants comportementaux (modules d'entrée) du constructeur analysent et contrôlent les stimuli en entrée tels que les gestes ou le son afin de les convertir en événements normalisés. Ces événements sont envoyés au gestionnaire d'événement qui les fusionne dans une commande simple qui est alors envoyée au modeleur. Grâce à son gestionnaire d'événements multimodal, CAVALCADE permet à l'utilisateur de combiner plusieurs dispositifs d'entrée (gant, système de reconnaissance de la parole, Spaceball, souris 3D, souris...) pour établir une seule commande. Le gestionnaire d'événements multimodal permet aussi de combiner des informations venant de plusieurs utilisateurs, ce qui donne la dimension coopérative.

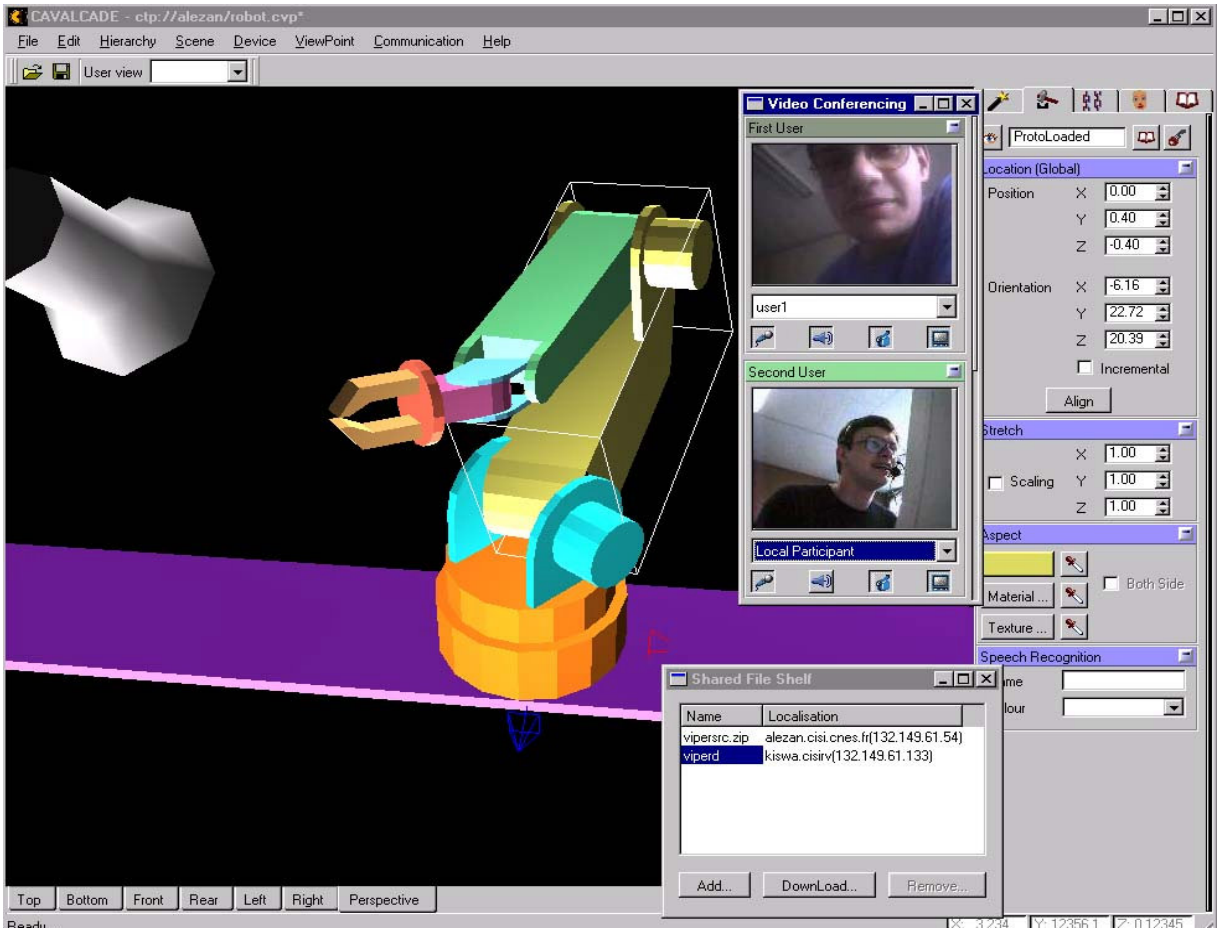


Figure 1.2.8 : Interface de CAVALCADE

L'interface homme-machine est un simple cadre où les autres entités ajoutent ou enlèvent leurs éléments graphiques spécifiques. Ensuite, une entité reçoit des stimuli par un capteur spécifique, chaque fois que ses éléments graphiques sont activés. Comme le montre la figure 1.2.9, deux types d'objets sont présents dans l'interface : des éléments graphiques 2D et 3D. L'interface 2D est un ensemble de menus, de boites de dialogue et de boutons qui entourent la partie de

l'interface 3D qui est composée de la fenêtre de rendu et d'un ensemble de manipulateurs 3D permettant d'agir sur les prototypes.

VIPER est utilisé dans CAVALCADE pour fournir l'architecture distribuée à l'application collaborative. Les entités de CAVALCADE sont de deux types : les entités simples qui sont contrôlées par le site où elles ont été créées et les entités dupliquées qui existent sur un ensemble de sites. Les entités dupliquées peuvent définir des attributs dupliqués qui sont synchronisés par VIPER sur un ensemble de sites : quand un tel attribut est modifié sur un site, toutes les copies de cet attribut sur d'autres sites sont mises à jour. Dans CAVALCADE les ordres sont envoyés aux sites intéressés en utilisant des communications point à point en passant par un serveur. Le serveur permet l'utilisation de moins de connexions entre les sites que dans le cas d'une connexion directe site à site et permet également l'envoi d'ordres à un ensemble de sites. CAVALCADE n'utilise pas de communication par groupe pour des impératifs de sécurité, de fiabilité et pour que l'application puisse fonctionner au-dessus d'un réseau n'appartenant pas au MBone.

Discussion

L'approche génie logiciel n'est qu'effleurée par la plupart des systèmes de réalité virtuelle distribuée. VIPER propose deux niveaux de programmation qui permettent soit de développer le contenu d'environnements virtuels (entités et modèles d'interaction) soit de développer des modèles de répartition. VIPER incorpore aussi bien des mécanismes d'extrapolation du mouvement des formes des entités (dead-reckoning), des découpages de l'environnement en cellules et portails qu'un niveau de fiabilité permettant d'adresser des applications coopératives (utilisation de TCP, de PVM et/ou de RMP). De plus, les entités peuvent changer de site pour permettre une interactivité accrue et un équilibre des charges de calcul sur la plate-forme répartie. Le système est extensible via la définition de nouveaux modèles de répartition, de nouvelles entités et de nouveaux stimuli en définissant de nouvelles classes qui sont ajoutées au système. De plus, le système peut être étendu par des modules externes (librairies dynamiques ou fichiers interprétés) définissant des comportements et des attributs d'entités ainsi que des stimuli, ce qui permet une extensibilité sans recompilation totale du système. Par contre, VIPER ne gère actuellement que les dispositifs pris en charge par WorldToolkit et tous les modules et fonctionnalités du système peuvent compliquer et alourdir le développement d'un prototype ou d'une application d'évaluation.

VRJUGGLER

VR Juggler, créé par le centre d'applications de réalité virtuelle de l'Université d'Iowa, a comme but d'offrir un environnement standard pour le développement, l'exécution et l'évaluation des applications de réalité virtuelle. Le système est conçu pour créer des applications avec n'importe quelle combinaison de dispositifs, de systèmes graphiques et de plates-formes matérielles. VR Juggler offre une plate-forme virtuelle pour le développement d'applications en fournissant un ensemble de primitives qui « cachent » le système d'exploitation sous-jacent. Le concept de plate-forme virtuelle s'étend sur toute la conception de VR Juggler comprenant des domaines comme le graphisme et la gestion de dispositifs d'interaction.

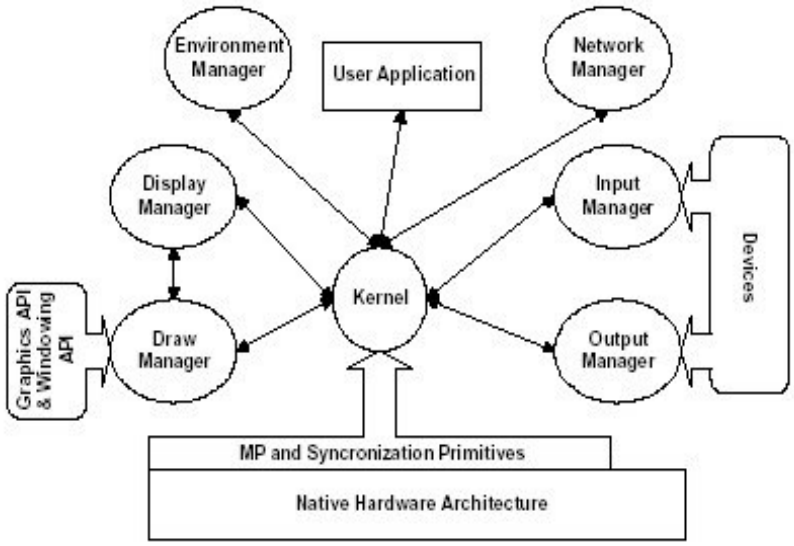


Figure 1.2.9 : Architecture de VRJuggler.
Dans ce système la fonctionnalité a été partagée entre plusieurs composants (managers). Chaque manager encapsule et cache un ensemble spécifique des détails du système. Par exemple, il existe un manager pour gérer les dispositifs et un autre pour la gestion de systèmes graphiques.

Gestion du monde

Pour VR Juggler, les applications sont des objets. Le système utilise l'objet application pour créer l'environnement virtuel dans lequel l'utilisateur interagit. L'objet application implante les interfaces nécessaires pour que la plate-forme virtuelle crée l'environnement virtuel. Le développeur de l'application doit écrire les routines OpenGL pour afficher l'environnement virtuel dans les endroits prévus à cet effet dans le squelette d'application fourni par le système.

Communications

VR Juggler ne permet pas encore la construction d'applications distribuées. A long terme, les processus qui forment une application pourront être distribués entre plusieurs machines pour utiliser les avantages de puissance de calcul, de dispositifs d'interaction et des systèmes graphiques.

Gestion de dispositifs

VR Juggler a été conçu pour permettre l'utilisation d'une grande variété de materiels de réalité virtuelle comme le système de projection CAVE, des tables de projection, de HMD et aussi des écrans simples. Le système prend aussi en charge différents dispositifs d 'entrée comme la souris Logitech3D et plusieurs types de gants de données. Une application VR Juggler est indépendante des dispositifs spécifiques, la même application peut être exécutée dans un CAVE ou avec un casque de visualisation sans être modifiée. Les dispositifs d'entrée ont des interfaces communes, les données de deux dispositifs similaires seront les mêmes pour l'application sans considération des différences de matériel.

VR Juggler définit des objets appelés « position proxies » qui représentent une entrée de position pour l'application, par exemple, un système de repérage ou le clavier. Le développeur et l'application communiquent seulement avec le proxy, le dispositif physique est inaccessible. De la même façon, l'application ne traite pas les complexités des dispositifs d'affichage : elle doit seulement décider l'API graphique à utiliser (OpenGL et Performer sont disponibles) et fournir l'ensemble des routines pour le rendu. Le 'Input Manager' possède et contrôle l'accès à tous les dispositifs d'interaction. Il est responsable de l'initialisation et de l'exécution de tous les pilotes des dispositifs. Il offre à tous les autres composants du système l'accès aux proxies. Quand un composant demande l'accès à un certain dispositif, il reçoit un pointeur ver le proxy approprié.

Généricité

VR Juggler est gratuit et disponible pour plusieurs plate-formes : IRIX, Linux et Windows. Il a été développe en C++ et permet l'utilisation des systèmes graphiques OpenGL et Iris Performer. Pour créer une application VR Juggler, il est nécessaire d'écrire le code spécifique de l'application dans les classes de base du système. Pendant son exécution, une application VR Juggler peut être contrôlée et reconfigurée via une interface graphique Java fournie avec le système.

Discussion

Dans VR Juggler tous les accès aux capacités concernant le système d'exploitation (threads, mémoire partagée, synchronisation) sont réalisés par des classes abstraites définies par le système. Ceci permet d'avoir des applications portables par rapport au VR Juggler, c'est à dire, une application peut être exécutée partout où VR Juggler est installé. Une caractéristique importante de ce système est la configuration dynamique qui permet de modifier certaines caractéristiques d'une application pendant son exécution. Cependant, pour ajouter de nouvelles fonctionnalités, il est nécessaire d'écrire les modules appropriés (par exemple le pilote d'un dispositif) et de recompiler l'application. D'autre part, VR Juggler ne permet pas encore le développement d'applications distribuées ce qui limite son utilisation à la construction d'applications mono-utilisateur.

WORLD TOOLKIT

WorldToolkit (WTK) est un système commercial, développé par Sense8, qui fournit une API de programmation pour le développement d'applications 3D immersives distribuées. WTK offre des services de lecture des capteurs, le rendu de la géométrie de la scène, le chargement de bases de données et la détection de collisions. Le développeur d'une application doit manipuler la simulation et modifier le graphe de scène de WTK.

Gestion du monde

La géométrie de WTK est basée sur un graphe de scène, qui spécifie comment effectuer le rendu de l'application. Les développeurs peuvent créer la géométrie en utilisant des sommets et des polygones ou en utilisant les primitives que WTK fournit (sphères, cônes, cylindres et texte 3D). La base des simulations WTK est l'Univers qui contient tous les objets de la simulation. Il est possible d'avoir plusieurs graphes de scène dans une application mais il doit exister un seul Univers. L'architecture de WTK est basée sur trois concepts : l'objet, la propriété et l'événement. Tous les objets ont des propriétés, accessibles via une interface commune, qui peuvent être partagées entre plusieurs sites et dont les changements génèrent des événements.

Communications

Pour créer des applications multi-utilisateurs, WTK fournit le système World2World. World2World utilise une architecture client/serveur pour distribuer et synchroniser les données

de la simulation. Cette architecture comprend deux types de serveur, pour séparer la fonctionnalité administrative de la simulation. Un client qui veut se connecter à une simulation fait une requête au Serveur « Manager ». Après autorisation accordée par celui-ci, les communications avec le client sont gérées par le Serveur de Simulation. Le Serveur de Simulation gère l'accès et la distribution des données partagées de la simulation aux clients qui ont indiqué leur intérêt pour ces valeurs. Un client reçoit uniquement les mises à jour des propriétés qui l'intéressent ce qui permet de réduire d'une certaine manière les communications (réalisées avec des messages UDP).

Gestion de dispositifs

WTK prend en charge plusieurs dispositifs de réalité virtuelle comme des joystick, capteurs, gants de données et casques de visualisation. Pour gérer les dispositifs WTK utilise des objets capteurs qui retournent la position et l'orientation dans le monde réel et qui peuvent contrôler le mouvement des objets dans la simulation. Il existe deux catégories de capteurs : relatifs et absolus. Un capteur relatif indique seulement les changements de position et d'orientation. Un capteur absolu rapporte les valeurs correspondant à une position et une orientation spécifiques. La boucle de simulation de WTK s'occupe de mettre à jour les données des capteurs et de gérer les données en accord avec leur catégorie.

Les interfaces que WTK offre pour contrôler les dispositifs ont été conçues pour des dispositifs spécifiques. Par exemple, les méthodes utilisées pour communiquer avec un casque de visualisation sont différentes de celles utilisées pour un CAVE donc l'application doit être modifiée et recompilée lorsque les dispositifs changent.

Généricité

WorldToolKit est un système commercial, développé en C. Il est disponible pour plusieurs plate-formes : SGI, Sun, HP, DEC, Intel et Linux. Les applications WTK sont des applications écrites en C/C++ qui utilisent les bibliothèques de programmation fournies par le système.

Discussion

Le système WorldToolKit est un système commercial donc de disponibilité limitée. Il fournit des pilotes pour un grand nombre de dispositifs mais ne permet pas le changement de ceux-ci de façon dynamique. En effet, les applications doivent être changées et recompilées lorsque les dispositifs changent donc elles ne sont pas completement indépendantes du matériel utilisé.

Conclusion

Dans ce chapitre nous avons présenté plusieurs systèmes de développement d'applications de réalité virtuelle. Un système de développement fournit aux développeurs des outils (architectures, bibliothèques) pour la construction et l'exécution de leurs applications. Typiquement, il définit une architecture prenant en charge les différents aspects d'un système de réalité virtuelle : gestion de dispositifs, rendu, communications, gestion de données et synchronisation. Ceci permet donc de créer une application sans avoir des connaissances approfondies sur chaque détail du système. Les systèmes de développement cherchent à simplifier la création d'applications de réalité virtuelle. Cependant, dans la plupart des cas, ils ont une telle complexité qu'il est très difficile de les utiliser pour le prototypage et la construction rapide de programmes de test. Par ailleurs, même si le développement d'applications de téléopération avec ces systèmes peut être envisagé, ils ne sont pas tout à fait adaptés à ce propos. Néanmoins, leur étude nous permet de définir les caractéristiques souhaitées d'un environnement de développement pour la téléopération que nous présentons dans le chapitre suivant.

chapitre 3

Vers un Outil de Développement pour la Téléopération

La conception du système ASSET a commencé à l'Université de Los Andes en Colombie, avec l'objectif d'utiliser les environnements virtuels comme interface utilisateur des systèmes de téléopération. Dans les premières applications liées à ce projet, le langage VRML [VRML] a été utilisé pour la description des scènes tridimensionnelles. Un applet Java (figure 1.3.1) permettait à l'utilisateur de diriger le robot virtuel. La connexion entre l'environnement virtuel et l'applet Java se réalisait en utilisant la technologie EAI (External Authoring Interface) [EAI].

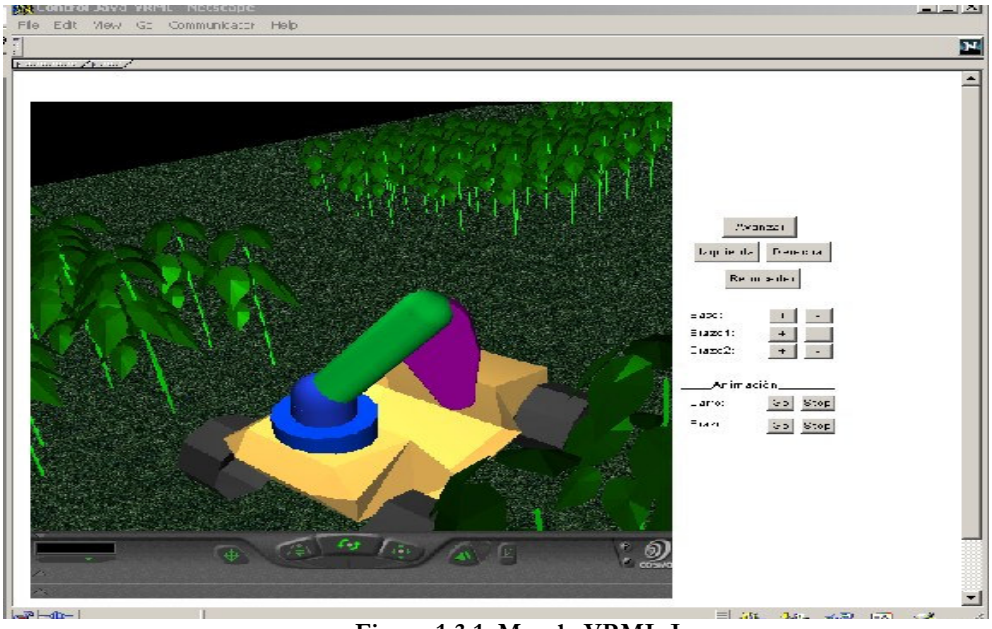


Figure 1.3.1 Monde VRML-Java

En combinant Java, VRML et l'EAI il est possible de construire des mondes virtuels complexes. Le désavantage de cette approche est que les applications ainsi développées sont liées et limitées par les capacités du navigateur utilisé, notamment pour l'intégration de dispositifs d'interaction et l'accès aux services du système d'exploitation. En outre, comme dans la plupart des projets en robotique, le résultat est un programme pour contrôler un robot spécifique qui interagit d'une façon particulière avec l'utilisateur et l'environnement. En conséquence, pour réaliser une modification au niveau des dispositifs utilisés ou de la fonctionnalité implantée, il est nécessaire de reconstruire toute l'application. Pour cette raison, il est important de mettre en œuvre des systèmes de développement flexibles permettant la construction rapide de prototypes et l'exploration des différentes alternatives de configuration d'un système robotique. Cependant, les architectures pour la robotique existantes ne permettent pas la construction de solutions complètes car elles se concentrent sur un seul aspect de la problématique (la conception du système, le contrôle de dispositifs effecteurs, l'interface utilisateur).

Dans le chapitre précédent, plusieurs systèmes de développement d'applications de réalité virtuelle ont été étudiés. Ces systèmes facilitent la création de nouvelles applications en prenant en charge les différents aspects concernant leur développement : la gestion du monde virtuel, le rendu graphique, l'intégration des dispositifs, etc. Nous avons donc utilisé ces expériences pour définir les caractéristiques qu'un outil de développement d'applications de téléopération doit avoir. Plusieurs leçons sont à retenir de cette étude. D'abord, le suivi d'une approche orientée objet dans la conception et le développement permet, grâce à ses caractéristiques (encapsulation, héritage), la construction de systèmes modulaires et extensibles. Une autre idée à conserver est l'utilisation de scripts de configuration afin de permettre la réalisation de modifications et donc le prototypage rapide. En revanche, plusieurs des systèmes de développements vus sont très complexes et leur utilisation pour la construction rapide et facile d'applications reste très limitée.

Toutes ces idées nous permettent de clarifier la problématique adressée par le système ASSET (Architecture pour des Systèmes de Simulation et d'Entraînement en Téléopération). L'objectif de notre travail est donc de construire une plate-forme de développement qui conserve les avantages des interfaces web de téléopération (facile à utiliser, infrastructure simple et à bas coût) tout en étant générale et très configurable. Notre système aborde la problématique de la téléopération en utilisant un environnement virtuel distribué, ce qui permet de mettre en œuvre

les techniques de communications, d'interaction, d'optimisation et de gestion de l'environnement propres a la réalité virtuelle et à la simulation distribuée.

Comme nous l'avons dit précédemment, un environnement de développement doit permettre aux utilisateurs de se concentrer sur la construction de leurs applications. Pour ce faire, plusieurs aspects doivent être pris en compte : les aspects liés au contexte des applications à développer, dans ce cas, à la téléopération (communications, contrôle de dispositifs et interface utilisateur) et la généricité du système.

- **Généricité**

Un système de développement doit savoir s'adapter aux différentes applications et aux différentes configurations d'une même application. De cette façon, il n'est pas nécessaire de réaliser des modifications importantes sur l'application lorsqu'un nouveau dispositif est utilisé. En outre, la modification et l'intégration de nouvelles fonctionnalités doivent pouvoir se réaliser d'une façon simple.

- **Communications**

Un système de téléopération est un système distribué. Ceci implique la définition et la gestion des communications entre les sites participants. Il est donc nécessaire de définir la configuration des sites, la partition des données et les mécanismes de synchronisation. Le système doit aussi prendre en compte les délais de communications en permettant l'intégration de mécanismes comme la simulation prédictive, la visualisation 3D et les dispositifs autonomes. Enfin, et pour accroître sa généralité, le système doit être indépendant d'un protocole de communications particulier.

- **Contrôle de dispositifs**

Dans certaines applications, les délais de communication rendent presque impossible le contrôle direct des dispositifs. Plusieurs schémas de contrôle ont été proposés pour donner de l'autonomie aux dispositifs effecteurs et permettre ainsi la réalisation de diverses tâches sans intervention continue de l'opérateur. Le système de développement doit donc faciliter l'intégration et la modification des différents schémas de contrôle disponibles.

- **Interface utilisateur**

L'interface d'utilisateur doit permettre à l'opérateur de se concentrer sur sa tâche plutôt que sur l'utilisation du système. Elle doit aussi présenter d'une façon simple toute l'information du site de travail nécessaire à l'accomplissement de la tâche téléoperée. Cette information peut être sous la forme d'une image vidéo (si la qualité des communications le permet) avec des images de synthèse ou une visualisation 3D. Si le système utilise une interface de réalité virtuelle, l'aspect immersion doit être considéré. Une immersion partielle utilise des dispositifs d'interaction plus simples que pour l'immersion totale, typiquement l'affichage a lieu sur un écran d'ordinateur auquel peuvent s'ajouter des lunettes stéréoscopiques.

Conclusion

Dans ce chapitre, les exigences d'un environnement de développement pour la téléopération sont décrites. Ces exigences sont identifiées grâce à l'étude des systèmes de développement de réalité virtuelle existants.

Ce chapitre clôt la première partie de ce mémoire, dédié à la présentation de l'état des recherches dans les domaines de la téléopération et la réalité virtuelle. La deuxième partie, consacrée à la description du système ASSET, commence par la description de l'architecture générale pour la téléopération définie par notre système.

PARTIE 2

le système ASSET

Description du système ASSET : architecture, mécanismes de collaboration, mise en oeuvre et expériences réalisées.

chapitre 1

Description de l'Architecture

Le système ASSET a pour but de faciliter la construction d'applications dans les différents domaines liés à la téléopération (dispositifs, autonomie, techniques d'interaction). Le premier pas vers la réalisation de ce système a donc été la définition d'une architecture générique réutilisable. Ce chapitre présente les considérations de conception de cette architecture, les différents modules qui la composent et ses principales caractéristiques.

Conception de l'architecture

Dans les chapitres précédents nous avons vu les trois éléments principaux qui forment un système de téléopération : le système d'interaction avec l'opérateur (système local), le système qui réalise la tâche souhaitée sur le site de travail (système distant) et le schéma de communication qui relie les deux. La figure 2.1.1 illustre les relations entre ces éléments.

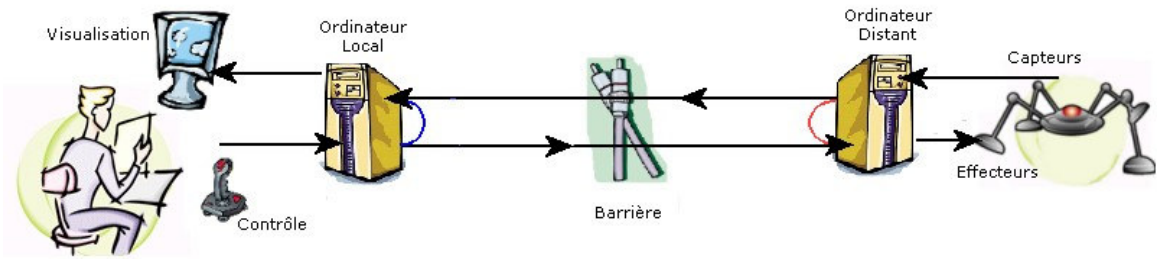


Figure 2.1.1 : Eléments de base d'un système de téléopération. Les ordinateurs local et distant sont séparés par une barrière de temps.

Le système d'interaction avec l'opérateur est l'ensemble des outils qui lui permettent de contrôler le système et de connaître l'état du site de travail. L'utilisateur génère des instructions et obtient le retour d'informations via l'interface du système local. La génération de commandes peut se réaliser par divers moyens : dispositifs d'interaction, interfaces de contrôle graphique, consoles textuelles ou par reconnaissance de la parole. Le retour d'information peut aussi être d'une grande diversité : images vidéo, images virtuelles, retour d'effort, etc. Le système local reçoit l'information provenant du système distant et réalise les traitements adéquats pour la présenter d'une manière compréhensible par l'utilisateur. De la même façon, le système local doit traiter les commandes de l'utilisateur pour les transformer en ordres que le système distant puisse interpréter. Le système distant, quant à lui, reçoit les instructions et les exécute via les dispositifs effecteurs. Les dispositifs effecteurs permettent au système de modifier le monde réel tandis que les capteurs permettent d'obtenir des informations sur le site du travail qui seront envoyées à l'opérateur. La barrière de temps représente les retards de transmission et d'opération inhérents au contrôle distant. Le retard de transmission est le temps que prennent les commandes pour atteindre le système réel et le retard d'opération est le temps nécessaire pour les exécuter.

La boucle locale de la figure 2.1.1 indique les informations de retour que le système local peut présenter à l'utilisateur sans attendre le résultat du système réel. Cette boucle locale cherche à diminuer le temps de réponse perçu par l'utilisateur en lui donnant, par exemple, des informations sur l'état de son ordre [Shneiderman92]. Si les retards de communication sont importants, une simulation prédictive peut s'avérer très utile. En utilisant la simulation pour estimer l'état du système réel après l'exécution des instructions, le système local obtient le résultat sans retards de transmission. La visualisation du nouvel état peut se réaliser avec des images de synthèse superposées aux images vidéo du site de travail (réalité augmentée) ou avec une scène complètement virtuelle (réalité virtuelle). L'avantage d'un environnement purement virtuel, l'option que nous avons choisie, est une visualisation et une interaction beaucoup plus fluides car moins affectées par les délais de communication. La figure 2.1.1 présente également une boucle distante. Cette boucle indique les commandes qui peuvent être générés directement par le système distant à partir de l'information récupérée par les capteurs. La boucle permet au système distant d'être autonome pour réagir promptement à certains événements (par exemple une collision) ou pour prendre en charge des tâches pour lesquelles l'intervention de l'utilisateur n'est pas indispensable.

Deux modules sont présents dans notre architecture : le module chargé de gérer l'interaction avec l'utilisateur que nous avons appelé **Gestionnaire Utilisateur** et le module responsable du

contrôle du système distant appelé **Gestionnaire Système Réel**. La table 2.1.1 synthétise les fonctions de chacun de ces modules.

Composant	Fonctions
Gestionnaire Utilisateur	• Gestion de l'interface utilisateur • Présentation de l'information • Production d'informations de retour • Envoi des instructions
Gestionnaire Système Réel	• Exécution des instructions • Récupération de l'information des capteurs • Envoi de l'état du système

Table 2.1.1 : Fonctionnalités des composants d'un système de téléopération

Dans les modules Gestionnaires, chaque fonction est encapsulée dans un composant spécifique. De plus, l'ensemble des services offerts par un composant est accessible uniquement à travers une interface déterminée. Les composants que nous avons définis sont :

- **Communications** : ce composant est présent dans les deux modules. Il permet l'envoi et la réception des instructions et de l'état du système.
- **Visualisation** : ce composant est le canal de communication entre le système et l'utilisateur. Il lui permet de connaître l'état du système et de générer des requêtes. Il comprend l'interface graphique utilisateur et les différentes sources d'information : images vidéo, images de synthèse, son, etc.
- **Simulation** : grâce à ce composant, il est possible de reproduire l'environnement de travail pour prédire les résultats des instructions données. Il permet d'avoir de l'information de retour sans attendre la réponse du système réel.
- **Dispositifs** : Dans le Gestionnaire Utilisateur, les dispositifs permettent à l'utilisateur de générer les instructions (dispositifs d'interaction) et parfois de recevoir de l'information, sous la forme de retour d'effort notamment. Dans le Gestionnaire Système Réel, les dispositifs sont utilisés pour rassembler l'information (capteurs) et pour exécuter les instructions (effecteurs).

Notre architecture, en plus d'être modulaire, cherche à être flexible. Il est donc aussi important de déterminer les éléments particuliers à chaque application. Il est clair que la sémantique de l'information n'a pas la même importance pour tous les composants du système : un message pour un dispositif doit avoir une signification permettant de l'exécuter alors qu'un message pour

le composant de communications est simplement un ensemble de données à transporter. Ce qui différencie une application d'une autre est justement la sémantique de ses données et le traitement qu'elle en fait. Cependant, tous les composants du système doivent échanger de l'information, nous avons donc défini et implanté tous les services leur permettant d'interagir. Par contre, dans notre architecture, le traitement des données spécifiques de l'application sera restreint aux objets de simulation et aux dispositifs. De cette façon, nous avons un système générique qui peut être mis en marche rapidement pour obtenir la fonctionnalité souhaitée. Le programmateur doit fournir uniquement le comportement des objets de simulation et des dispositifs : toute la collaboration entre composants pour l'exécution de l'application est déjà en place.

Le Gestionnaire Utilisateur

La figure 2.1.2 illustre la composition du Gestionnaire Utilisateur. Dans ce module nous trouvons les composants énumérés dans la section précédente, à savoir : communications, visualisation, simulation et dispositifs. Comme nous l'avons vu, ces deux derniers composants requièrent une partie complémentaire externe à l'architecture qui doit être fournie par le développeur d'applications. Les modèles géométriques et les comportements sont utilisés par la simulation pour récréer l'environnement virtuel. Les dispositifs virtuels quant à eux définissent l'interface que les dispositifs physiques doivent implanter (les détails de l'interaction entre les éléments internes et externes de l'application sont présentés dans le chapitre suivant).

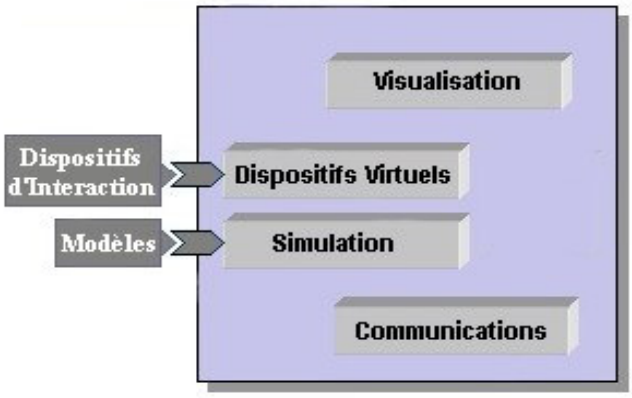


Figure 2.1.2 : Composants du Gestionnaire Utilisateur/Gestionnaire Système Réel

Le Gestionnaire Utilisateur contrôle l'exécution de la partie utilisateur du système de téléopération. Ce module implante ses services en appelant les méthodes définies dans les composants, qui n'interagissent jamais directement. En effet, chacun d'eux offre une fonction qui ne requiert pas de collaboration avec les autres. Et puisqu'il n'existe pas de dépendances entre les composants, il est possible de les modifier ou de les échanger facilement. Le Gestionnaire Utilisateur se charge de l'initialisation et de la configuration de tous les composants. Il démarre les communications avec le système distant et met en place l'environnement de travail de l'opérateur en utilisant le fichier de configuration (voir chapitre suivant - Généricité). Le Gestionnaire Utilisateur se charge aussi de répondre aux requêtes de l'utilisateur pour démarrer, arrêter et mettre à jour la simulation. Il traite également les messages provenant du système distant pour les présenter à l'utilisateur.

La figure 2.1.3 montre les actions réalisées par le Gestionnaire Utilisateur à chaque signal de l'horloge une fois la simulation démarrée. Les instructions générées par l'utilisateur sont d'abord récupérées, puis elles sont communiquées au simulateur pour modifier l'état du système et pour produire l'information de retour. Si l'état résultant est valide, les instructions seront envoyées au système réel (un état est valide si aucune collision ne se produit pendant l'exécution des instructions).

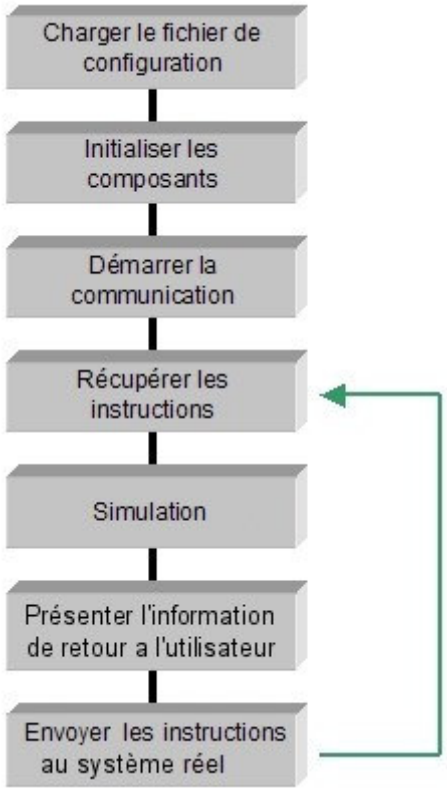


Figure 2.1.3 : Boucle principale du Gestionnaire Utilisateur

Le Gestionnaire Système Réel

Le Gestionnaire Système Réel est constitué de la même façon que le Gestionnaire Utilisateur. Il possède un composant pour la gestion des communications, un composant pour le contrôle des capteurs et des effecteurs, un composant de visualisation, et un composant de simulation (Figure 2.1.2).
L'inclusion du composant de simulation dans le système distant est une des innovations de notre approche. En effet, dans les systèmes de téléopération la simulation est utilisée exclusivement pour produire l'information de retour à l'utilisateur. Par contre, dans notre architecture la simulation est utilisée aussi comme un outil pour augmenter la performance du système. Comme nous avons vu dans la partie précédente, les systèmes de réalité virtuelle distribuée utilisent diverses stratégies pour gérer l'information, parmi lesquelles la duplication des données et le dead-reckoning. En dupliquant les données dans les sites participants, la quantité d'information échangée diminue car sont communiqués uniquement les modifications. Nous avons donc dans chaque module une copie de la géométrie et des comportements des objets de la simulation.

Un autre aspect important de la transmission de l'information est sa fréquence. Pendant l'exécution d'une application distribuée, les commandes de l'utilisateur sont transmises au système distant pour ensuite récupérer l'état et le présenter à la fin de chaque intervalle de simulation. Nous utilisons la simulation pour montrer un résultat approximatif avant de présenter la réponse du système distant. Mais nous pouvons nous passer de cette dernière étape si la simulation est assez fiable. En effet, si le résultat approximatif est très proche du résultat réel nous n'avons pas besoin de réaliser la visualisation de ce dernier à chaque intervalle de temps. Les mises à jour étant donc plus éloignées, le nombre de transmissions diminue et l'interaction avec l'utilisateur est plus fluide.

Le mécanisme de mise à jour doit être configuré pour chaque application. Le programmeur définit dans le fichier de configuration l'ensemble des variables indicateurs de l'état du système (UpdateSet) et pour chacune de ces variables, le seuil d'erreur. La simulation est mise à jour si une ou plusieurs variables atteignent leur valeur d'erreur maximum. L'UpdateSet et les seuils d'erreur étant définis dans le fichier de configuration, il est possible de les modifier facilement pour évaluer la performance du système par rapport à la fréquence des mises à jour.

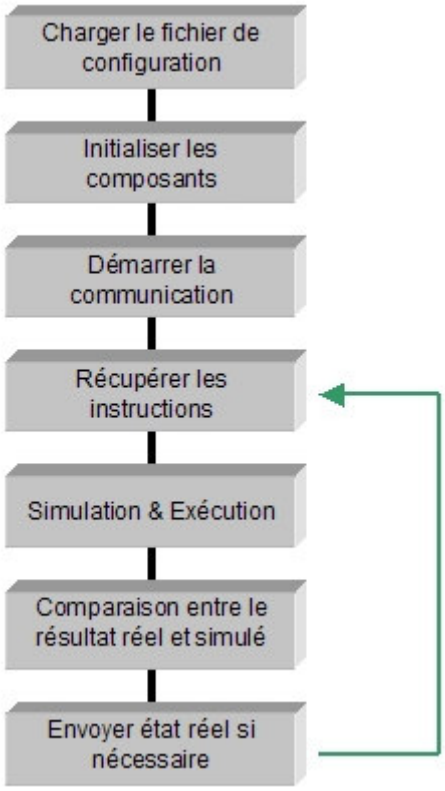


Figure 2.1.4 : Boucle principale du Gestionnaire Système Réel

La figure 2.1.4 illustre les actions réalisées par le Gestionnaire Système Réel pendant l'exécution de l'application. Les composants sont d'abord configurés puis les communications sont démarrées. Ensuite, à chaque signal de l'horloge, les instructions provenant du Gestionnaire Utilisateur sont récupérées et communiquées au simulateur et aux dispositifs. Après l'exécution des instructions, l'état provenant de la simulation et l'état réel du système sont comparés pour démarrer si nécessaire la mise à jour de la simulation du Gestionnaire Utilisateur.

L'Administrateur

L'architecture que nous avons définie jusqu'à présent est une architecture mono-utilisateur qui permet la réalisation d'une mission de téléopération. Mais la collaboration entre plusieurs utilisateurs est un élément important à considérer pour des applications d'entraînement ou d'apprentissage notamment. Nous avons donc ajouté un troisième module à notre architecture pour donner la possibilité de créer des applications multi-utilisateurs. Ce module, appelé Administrateur, comprend deux composants : le Coordinateur d'Utilisateurs et les

Communications. Le Coordinateur d'Utilisateurs est chargé de résoudre les conflits produits par les ordres de différents utilisateurs tandis que le composant de communication transmet les instructions à exécuter et l'information du système réel aux utilisateurs.

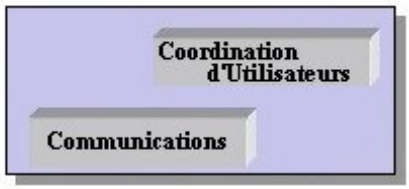


Figure 2.1.5 : Composants du module Administrateur

L'Administrateur est un médiateur entre les deux Gestionnaires. Dans notre architecture, les Gestionnaires ne communiquent pas directement : toutes les interactions entre eux sont gérées par l'Administrateur. Cependant, ce module est indépendant des autres et il est donc possible aux utilisateurs d'établir une connexion en attendant que le Gestionnaire Système Réel soit disponible. De cette façon, les connexions entre les utilisateurs et le Gestionnaire Système Réel peuvent être facilement initiées ou rétablies. De plus, la présence de l'Administrateur poursuit l'idée de séparation fonctionnelle, caractéristique de notre architecture : la gestion des clients sera réalisée par un module spécifique et non par le Gestionnaire Système Réel comme c'est le cas dans la plupart des systèmes de téléopération actuels. De même, l'indépendance de l'Administrateur permet d'envisager la gestion de plusieurs Gestionnaires Systèmes Réels. Ainsi, un ou plusieurs utilisateurs, peuvent contrôler des robots situés sur différents sites.

Communication entre composants

Dans les paragraphes précédents nous avons présenté les éléments composant notre architecture. Nous allons maintenant décrire comment ils interagissent entre eux. Toutes les interactions entre les modules de notre système s'effectuent suivant un modèle d'événements. Ce modèle, étendu du modèle Java pour la gestion des interfaces utilisateurs graphiques [JavaEventModel97], est basé sur trois concepts : l'événement, le générateur d'événements (la source) et l'observateur d'événements (le listener). Un événement indique que quelque chose s'est produit : l'utilisateur a déplacé la souris, un message du réseau est arrivé, etc. La source propage les événements et permet aux observateurs de s'enregistrer pour recevoir les événements qui les intéressent. L'événement se propage de la source aux observateurs en appelant une méthode spécifique dans ceux-ci. Un observateur doit donc implanter une interface particulière pour pouvoir répondre aux événements envoyés par la source.

Ce modèle d'événements est puissant et flexible. Les observateurs peuvent recevoir tout type d'événements provenant des différentes sources. Pour avoir accès aux événements, un composant doit simplement disposer de l'interface requise et après s'être enregistré auprès de la source, il est notifié des occurrences de l'événement. Ce modèle nous permet donc de garder les propriétés voulues (modularité et flexibilité) pour notre architecture car les composants continuent à communiquer uniquement à travers des interfaces.

L'événement contient l'information sur la source et l'information spécifique qui permet à l'observateur de réagir. En définissant les événements et les observateurs, nous relions les composants de l'architecture pour mettre en œuvre le comportement global de l'application. Pour implanter la gestion d'événements, nous avons donc introduit dans chaque module un composant additionnel appelé l'espace de données (figure 2.1.6).

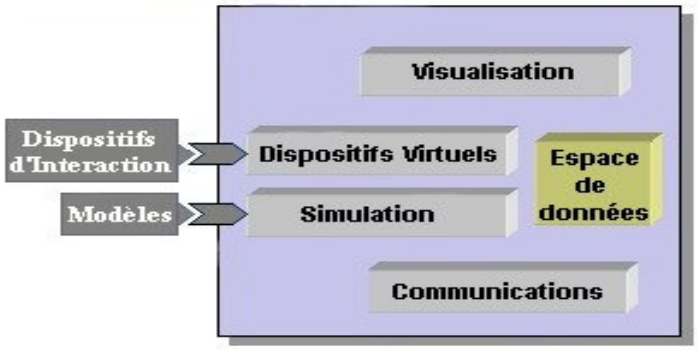


Figure 2.1.6 : Composition finale du Gestionnaire Utilisateur/Gestionnaire Système Réel

L'espace de données permet aux composants de stocker et de lire des messages. Des événements sont générés lorsqu'un composant ajoute un message permettant aux observateurs intéressés de le récupérer. Les composants observateurs n'ont pas besoin de vérifier fréquemment si les actions qui les intéressent se sont produites. Ils peuvent ainsi réaliser d'autres tâches et leur comportement s'en trouve d'autant simplifié. L'espace de données maintient l'information des observateurs et les tableaux de messages pour les objets de simulation, les dispositifs et le composant de communications. Il permet l'échange des informations entre les différents composants d'un module et facilite la tâche de celui-ci en réalisant un pré-traitement de l'information. La table 2.1.2 présente les observateurs et les événements définis dans notre système.

Evénement	Description
MsgDispAListener	Répond aux événements de type MsgDispAEvent, générés lorsqu'une instruction pour un dispositif effecteur est enregistrée dans l'espace de données.

MsgDispIListener	Répond aux événements de type MsgDispIEvent, générés lorsqu'une instruction pour un dispositif d'interaction est enregistrée dans l'espace de données.
MsgErrorListener	Répond aux événements de type MsgErrorEvent, générés lorsqu'une erreur se produit dans un dispositif.
MsgObjListener	Répond aux événements de type MsgObjEvent, générés lorsqu'une instruction pour un objet de simulation est enregistrée dans l'espace de données.
MsgOutListener	Répond aux événements de type MsgOutEvent, générés lorsqu'un message destiné à un module est enregistré dans l'espace de données.
NetMsgListener	Cette classe définit les méthodes qui doivent être définies par les composants souhaitant recevoir les messages provenant des autres modules.
UIWindowListener	Répond aux événements générés lorsque l'utilisateur interagit avec l'interface utilisateur graphique.
DeviceListener	Cette classe définit les méthodes qui doivent être définies par les composants souhaitant recevoir les messages provenant des dispositifs.

Table 2.1.2 : Fonctions des composants d'un système de téléopération

Les interactions entre la simulation, la communication réseau et les dispositifs se réalise à travers l'espace de données et les événements que celui-ci génère. Par exemple, un dispositif à retour d'effort implémente l'interface MsgDispIListener pour connaître les instructions provenant de l'objet de simulation auquel il est lié. De cette façon, les deux composants interagissent sans connaître leurs détails d'implantation.

Conclusion

Dans ce chapitre nous avons présenté l'architecture générique pour la téléopération définie dans le système ASSET. Cette architecture est constituée de trois modules : le Gestionnaire Utilisateur, représentant le système local, le Gestionnaire Système Réel représentant le système distant et l'Administrateur responsable des communications entre les deux autres modules. Plusieurs composants sont présents dans les Gestionnaires : communications, visualisation, simulation et dispositifs. L'architecture spécifie aussi les interactions entre les différents composants. En effet, toute l'interaction entre les composants est définie par le passage de messages et synchronisée à travers des événements et d'un espace de données. Les composants de notre architecture sont

indépendants d'une application particulière. Ils définissent et gèrent le transport des données sans effectuer un quiconque traitement. Le traitement de données, laissé au programmeur, est réalisé uniquement dans les classes de contrôle des objets de simulation et des dispositifs. Ceci, et l'approche orientée-objet que nous avons suivie pour la conception de notre architecture, permet la réutilisation des composants et facilite l'étape d'intégration puisque l'interface fonctionnelle de chaque composant reste fixée.

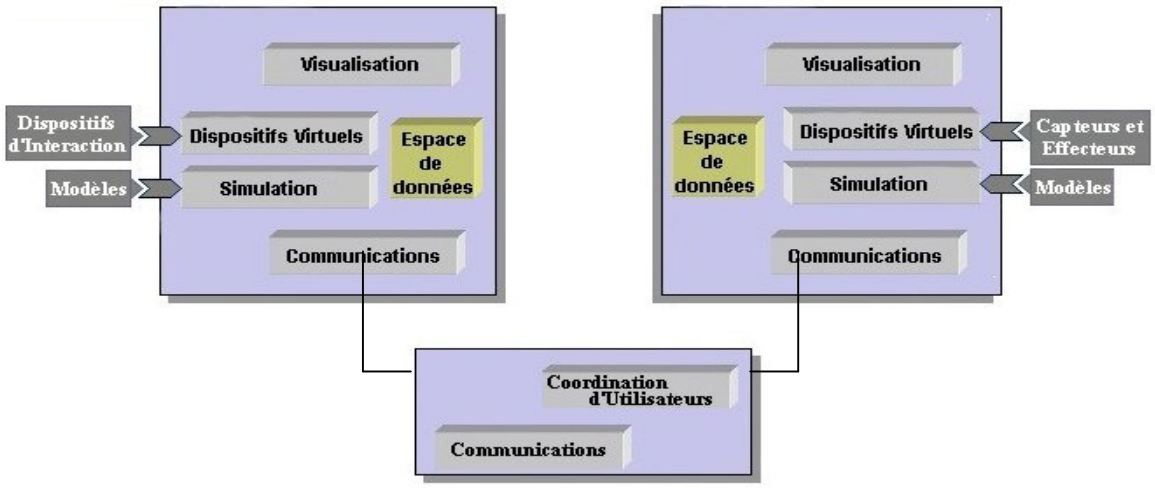


Figure 2.1.7 : Architecture du système ASSET

En plus de l'architecture, le système ASSET définit la gestion de différents aspects liés à une application de téléopération. Les détails de cette gestion sont présentés dans le chapitre suivant.

chapitre 2

Le Système Asset

Dans le chapitre précédent nous avons décrit l'architecture de téléopération et les mécanismes de communication interne du système ASSET. Maintenant, nous allons décrire comment les divers aspects à gérer, dans le développement d'une application de téléopération, sont implantés dans notre système.

Simulation

Le composant de simulation du Gestionnaire Utilisateur a deux buts : générer l'information de retour sans attendre la réponse du système distant et valider les instructions avant leur exécution. L'utilisateur interagit avec l'environnement virtuel - géré par la simulation - qui recrée le site de travail. Le modèle du monde, fourni par le développeur de l'application, est utilisé par le simulateur pour générer l'état de l'environnement à un instant donné. Ce modèle contient les informations (géométrie, caractéristiques physiques, comportement) des objets de l'environnement virtuel.

Pendant l'initialisation du système, le modèle de l'environnement est extrait du fichier de configuration pour créer les différents objets. Le composant de simulation (classe Simulator) maintient la liste de tous les objets du monde virtuel. Les objets de simulation (classe SimObject), quant à eux, contiennent l'information concernant la géométrie, le comportement et l'état (classe StateInfo).

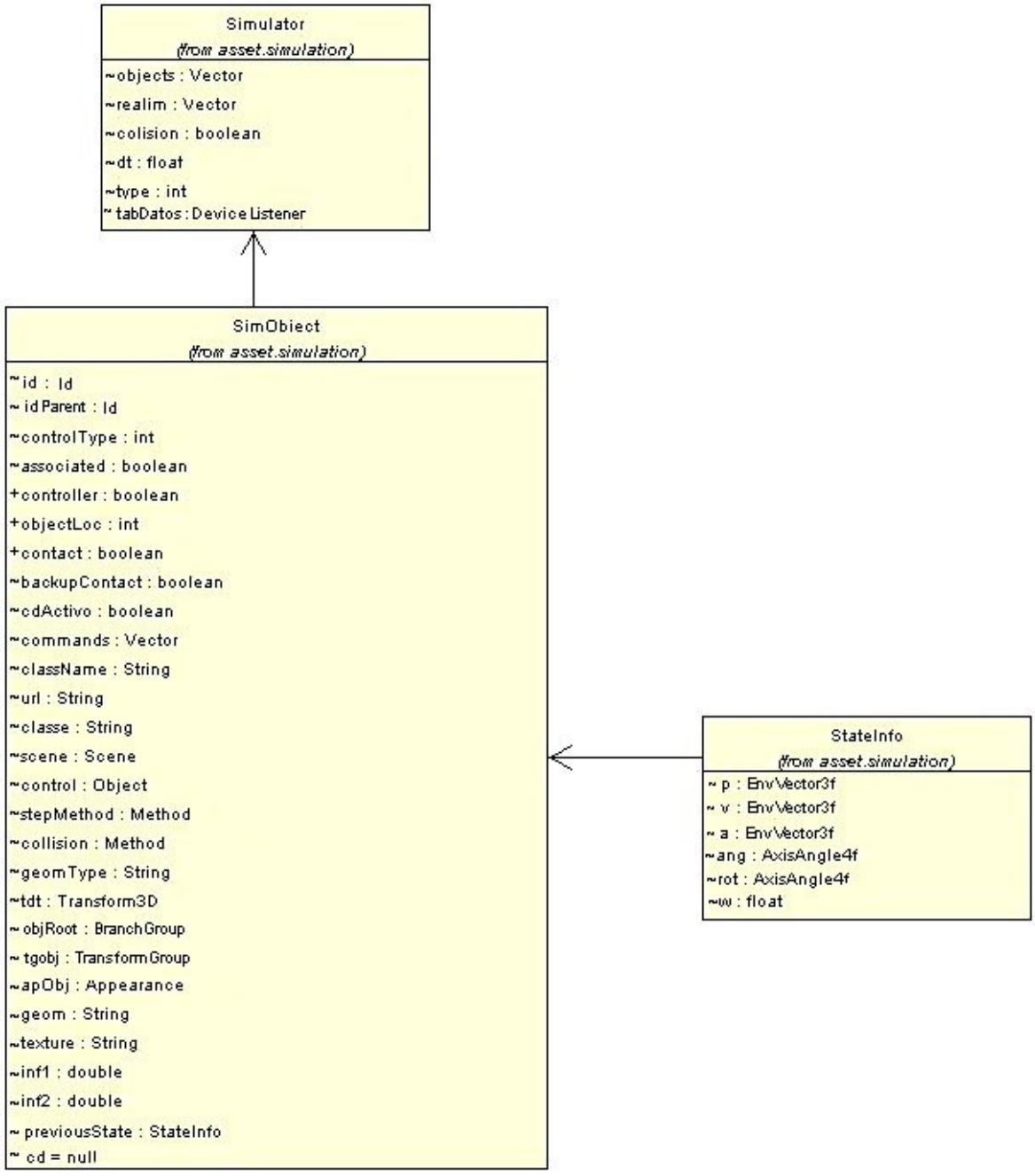


Figure 2.2.1 : Classes Simulator, SimObject et StateInfo

La géométrie d'un objet peut être définie en utilisant un fichier VRML97, un fichier Java3D ou une primitive Java3D. Le comportement de l'objet, par contre, doit être défini dans une classe Java qui implémente une interface prédéfinie. Ceci est nécessaire car les classes de comportement sont chargées de façon dynamique et car il n'existe pas de restrictions sur les méthodes à implanter. L'ensemble de méthodes de l'interface permet donc à la simulation de communiquer avec tous les objets, sans avoir besoin de connaître les détails d'implantation de chacune des différentes classes de comportement.

La simulation effectue diverses étapes à chaque signal de l'horloge. La première étape consiste à faire évoluer dans le temps les objets de la simulation. Chaque objet utilise son état actuel et les consignes reçues (provenant des dispositifs) pour invoquer les méthodes de sa classe de comportement et générer ainsi son nouvel état. Ensuite, l'ensemble des objets de la simulation est parcouru pour repérer les collisions qui ont pu se produire. Une fois les collisions identifiées, des méthodes dans les objets concernés sont invoquées pour les résoudre. Si une collision ne peut pas être résolue par la simulation, le Gestionnaire Utilisateur demande l'intervention de l'utilisateur. La collision est traitée alors par l'utilisateur en choisissant entre deux options : annuler les instructions qui ont amené à la collision pour revenir au dernier état valide ou mettre à jour la simulation pour avoir l'état réel du système. La collision ne peut pas être ignorée car les instructions à envoyer risquent de produire des erreurs dans le système distant[3]. Cependant, il est possible qu'une collision existant dans l'environnement virtuel ne puisse pas se produire dans le système réel. Ceci est dû à la présence des objets purement virtuels, ajoutés pour donner des repères ou pour aider l'utilisateur dans la réalisation de sa tâche (figure 2.2.2). Dans ce cas, pour éviter l'arrêt systématique du système, le développeur peut désactiver dans le fichier de configuration l'option de détection de collisions pour un objet particulier.

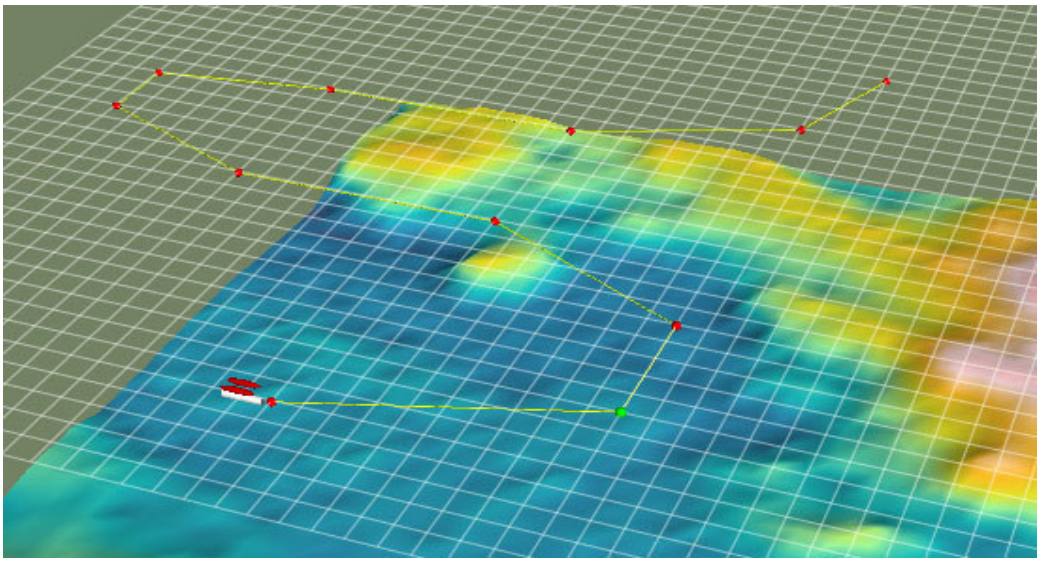


Figure 2.2.2 : Guides facilitant la navigation dans l'environnement virtuel [Komerska02]

Dans ASSET, il est possible de définir le comportement pour chaque objet de simulation. En conséquence, des entités avec différents degrés d'autonomie peuvent partager le même environnement. Les objets *autonomes* évoluent dans le monde virtuel sans intervention de

[3] Cependant, si les instructions sont envoyées, la simulation dans le Gestionnaire Système Réel détectera la collision et génèrera automatiquement la mise à jour du Gestionnaire Utilisateur.

l'utilisateur et les objets *contrôlés* évoluent en accord avec les consignes générées en utilisant un dispositif d'interaction. D'autre part, les objets de la simulation distante peuvent être *contrôleurs* de dispositifs réels. Enfin, les objets de la simulation locale et de la simulation distante peuvent être liés entre eux (*associés*). Ces attributs nous permettent de « trier » les objets pour améliorer la mise à jour du système. Par exemple, les objets caractérisés uniquement par l'attribut *autonome* ne requièrent pas de données de mise à jour provenant du système distant. Ceci est le cas des guides virtuels de la figure 2.2.2 : puisqu'ils ne sont pas associés à des objets réels, leur mise à jour n'est pas nécessaire. Cependant, si le développeur souhaite mettre à jour l'objet autonome local avec l'état de l'objet autonome distant il peut ajouter l'attribut *associé* à la définition de l'objet pour établir la connexion entre les deux entités.

Gestion du monde

Dans notre système, le composant de simulation est aussi présent dans le système distant. Grâce à ce composant, le Gestionnaire Système Réel n'est pas obligé de transmettre son état à la fin de chaque boucle de simulation. Dans ASSET, la mise à jour de la simulation locale se produit uniquement lorsque la différence entre l'état de la simulation et l'état réel dépasse un certain seuil, afin de diminuer la fréquence des synchronisations. Pour améliorer encore la communication entre les deux Gestionnaires, nous avons aussi cherché à diminuer la quantité d'information à transmettre. En effet, chaque simulation maintient une copie de la géométrie et du comportement des objets, ce qui permet aux sites de communiquer uniquement les instructions de modification. La duplication de comportement permet aussi de limiter le nombre d'objets qui doivent être mis à jour.

Pour clarifier les détails du mécanisme de synchronisation nous allons utiliser le scénario illustré sur la figure 2.2.3. Dans cette scène nous trouvons plusieurs objets : un chemin, un robot mobile, un petit avion de surveillance, un feu de signalisation et un point de repère.

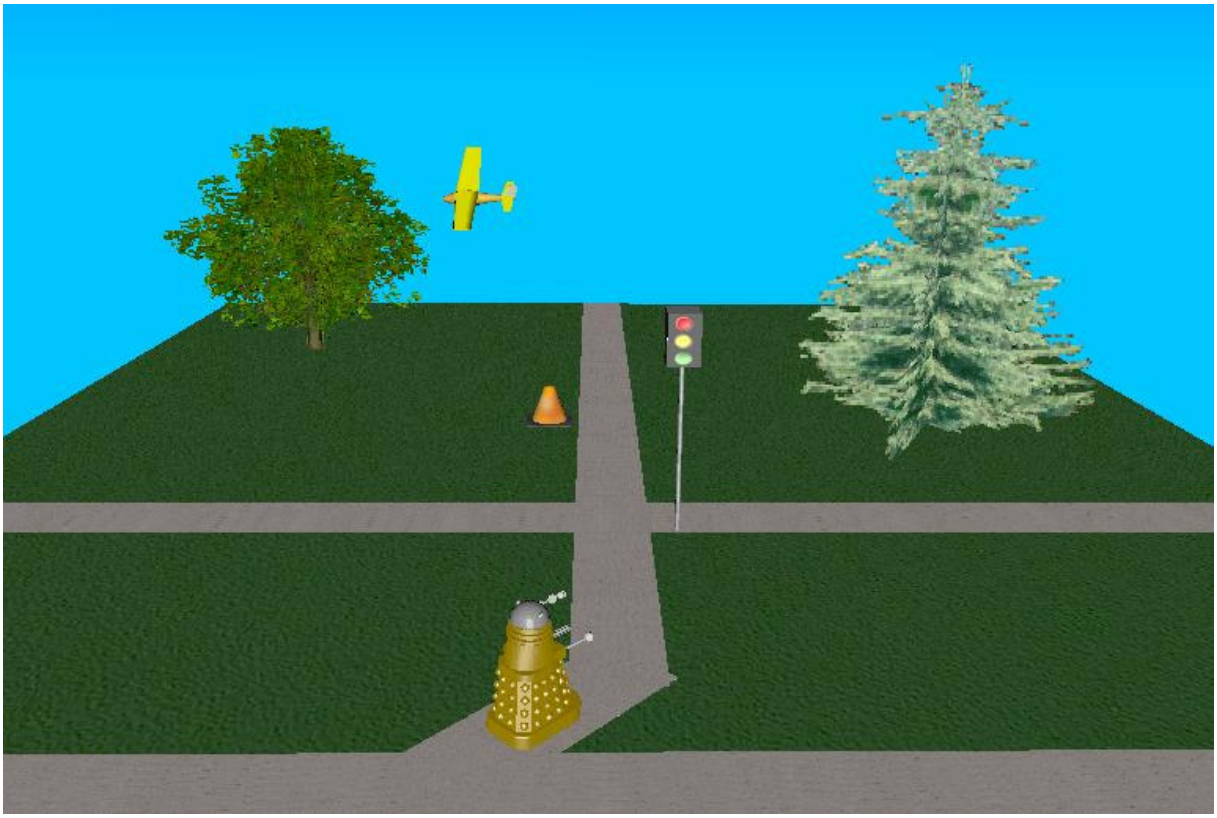


Figure 2.2.3 : Scénario illustrant les différents types d'objets présents dans la simulation

Dans notre exemple, le robot mobile est contrôlé par un dispositif d'interaction et l'avion est autonome. Nous supposons en plus que ces objets sont les représentations d'objets réels. Leur état sera donc inclus dans l'information de mise à jour et leurs modifications seront échangées par les deux sites. Le chemin et le point de repère sont des objets autonomes non modifiables (dans notre scénario) donc les deux Gestionnaires n'échangent pas d'information sur eux. Le feu, lui, est un objet autonome qui change dans le temps. Il évolue dans la simulation locale et dans la simulation distante de la même façon, son comportement étant exécuté sur les deux sites. Il n'est donc pas indispensable de transmettre de l'information le concernant. Cependant, si le développeur souhaite connaître son état au moment de la mise à jour sans avoir à le recalculer, il peut l'associer avec le feu de la simulation distant. Il existe encore un dernier type d'objet qui n'est pas présent dans notre scène : un objet virtuel contrôlé par un dispositif d'interaction. Un exemple de ce type d'objet peut être un point de repère modifiable par l'utilisateur. En principe, si un objet n'existe pas dans le monde réel, il est pas nécessaire d'envoyer ses modifications. Cependant, la transmission d'information permet, dans ce cas, de maintenir la cohérence entre les données locales et les données distantes. La table 2.2.1 résume les synchronisations réalisées pour les différents types d'objet.

Objet	Autonome	Contrôlé	Associé	Contrôleur	Participe à la Mise à jour
Robot		x	x	x	Oui
Avion	x		x	x	Oui
Chemin	x				Non
Feu	x		x		Oui
Point de repère	x				Non
Point de repère modifiable		x	x		Oui

Table 2.2.1 : Types d'objet et la synchronisation

Nous décrivons maintenant le mécanisme de déclenchement de la mise à jour. La synchronisation se produit lorsque la différence entre l'état de la simulation et l'état réel dépasse un certain seuil. Et puisque les concepts d'état, de différence et de seuil sont spécifiques à chaque application, le développeur doit intervenir dans leur configuration. Le développeur définit dans le fichier d'initialisation d'ASSET l'ensemble de variables qui constituent l'état de son application. Cet ensemble est géré dans le système ASSET par la classe UpdateSet (figure 2.2.4). Pour chacune de ces variables (classe UpdateVariable), l'utilisateur définit aussi un seuil d'erreur et une méthode permettant de calculer la différence entre deux valeurs de la variable. A chaque intervalle de simulation, l'état simulé et l'état réel sont comparés utilisant l'UpdateSet. La mise à jour est déclenchée lorsqu'une ou plusieurs variables atteignent leur seuil d'erreur.

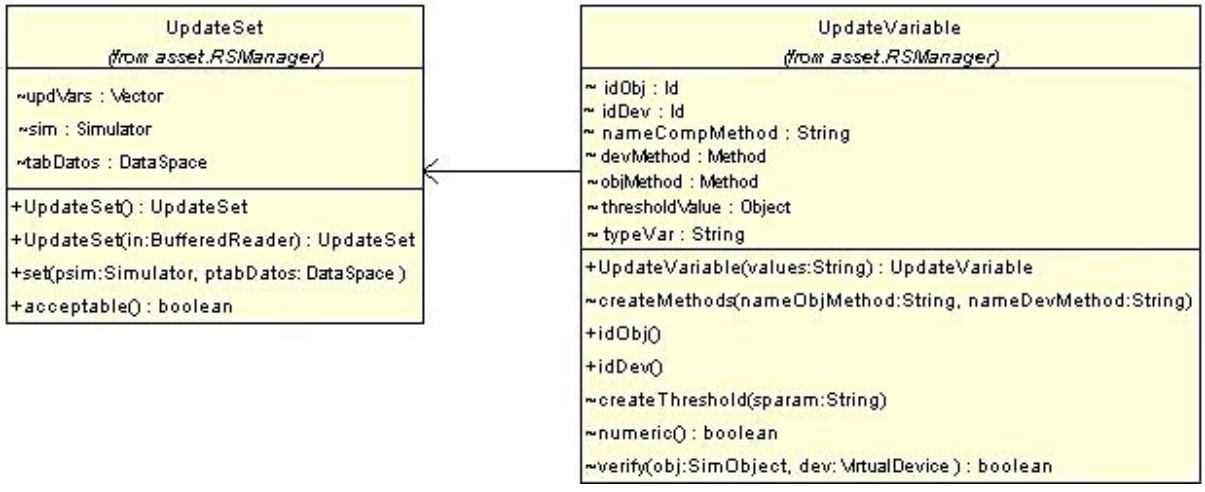
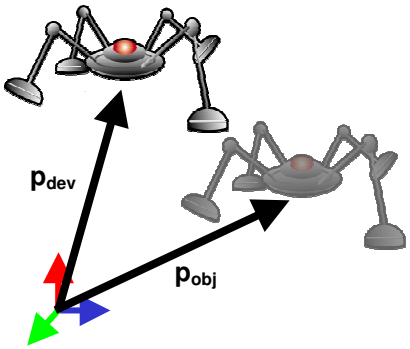


Figure 2.2.4 : Classes UpdateSet et UpdateVariable

Dans notre scénario la tâche à accomplir consiste à diriger le robot pour parcourir le chemin. Dans ce cas, trois variables sont choisies pour former l'état du système : la position du robot (p), son orientation (α) et une variable indiquant la proximité des obstacles (collision). Ces indicateurs déclencheront la mise à jour de la simulation locale (figure 2.2.5) pour corriger, si besoin est, la trajectoire du robot.



if (
($|\alpha_{obj} - \alpha_{dev}| > \varepsilon_\alpha$) ||
($|p_{obj} - p_{dev}| > \varepsilon_p$) ||
($collision_{obj}$!= $collision_{dev}$))
⇒ update()

*obj : objet de simulation
dev : dispositif

Figure 2.2.5 : Conditions de déclenchement de la mise à jour

La possibilité de définir l'état du système et la différence acceptable entre les deux états (réel et simulé), permet au développeur de synchroniser les modules en suivant les demandes de son application. De plus, il lui est possible de calibrer le système à la fréquence de mise à jour souhaitée pour atteindre une certaine performance.

Communications

Les trois modules de notre architecture (Gestionnaire Utilisateur, Gestionnaire Système Réel et Administrateur) reçoivent et envoient des requêtes. Le Gestionnaire Utilisateur par exemple transmet les instructions de l'utilisateur et reçoit les mises à jour et les messages d'erreur provenant du système distant. Il n'existe donc pas dans notre architecture un module ayant exclusivement un comportement de client ou de serveur.

Pour initier les communications, l'Administrateur (class Administrator) doit être démarré en premier pour permettre l'interaction entre les deux Gestionnaires. Il attend ensuite les messages de « présence » provenant du Gestionnaire Système Réel (classe RSManager) ou d'un Gestionnaire Utilisateur (classe UIManager). Dans notre système, le Gestionnaire Utilisateur peut s'enregistrer avant le Gestionnaire Système Réel. Dans ce cas, il établit la connexion avec

l'Administrateur pour être informé dès que le système réel est disponible (Figure 2.2.6). Cette caractéristique permet au Gestionnaire Système Réel de rétablir facilement les connexions avec ses clients après une reconfiguration.

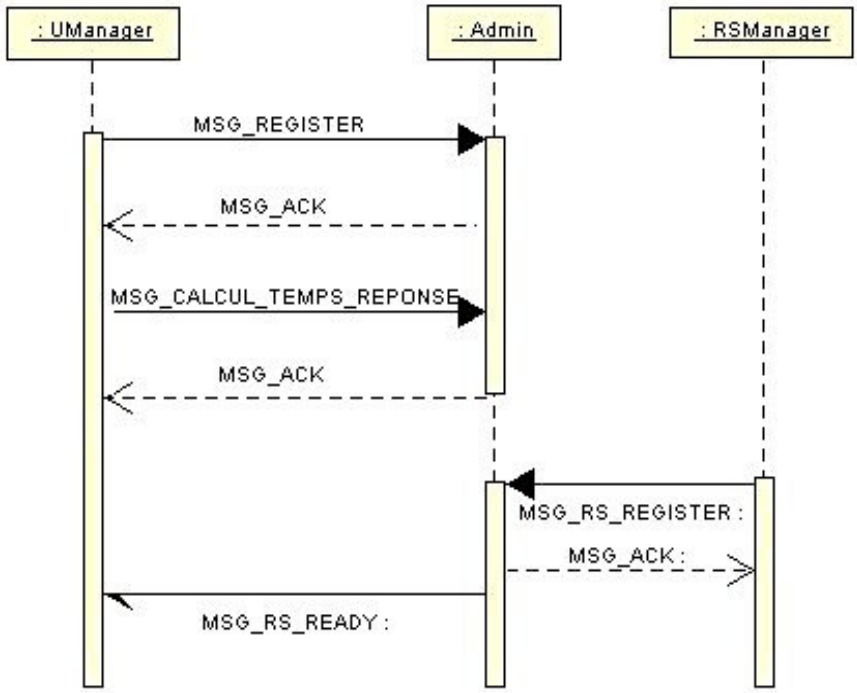


Figure 2.2.6 : Etablissement de communications entre les modules

La figure 2.2.7 montre les messages échangés par les modules pour les requêtes provenant du Gestionnaire Utilisateur : démarrer, arrêter ou reprendre la simulation (MSG_START/MSG_STOP/MSG_RESTART) et mettre à jour le système à la demande de l'utilisateur (MSG_UPDATE_GU). Ces messages sont adressés à l'Administrateur qui les renvoie (sans l'identificateur du client) au Gestionnaire Système Réel. L'identificateur permet à l'Administrateur de stocker et visualiser les messages provenant d'un client donné, il n'a pas de signification pour le Gestionnaire Système Réel.

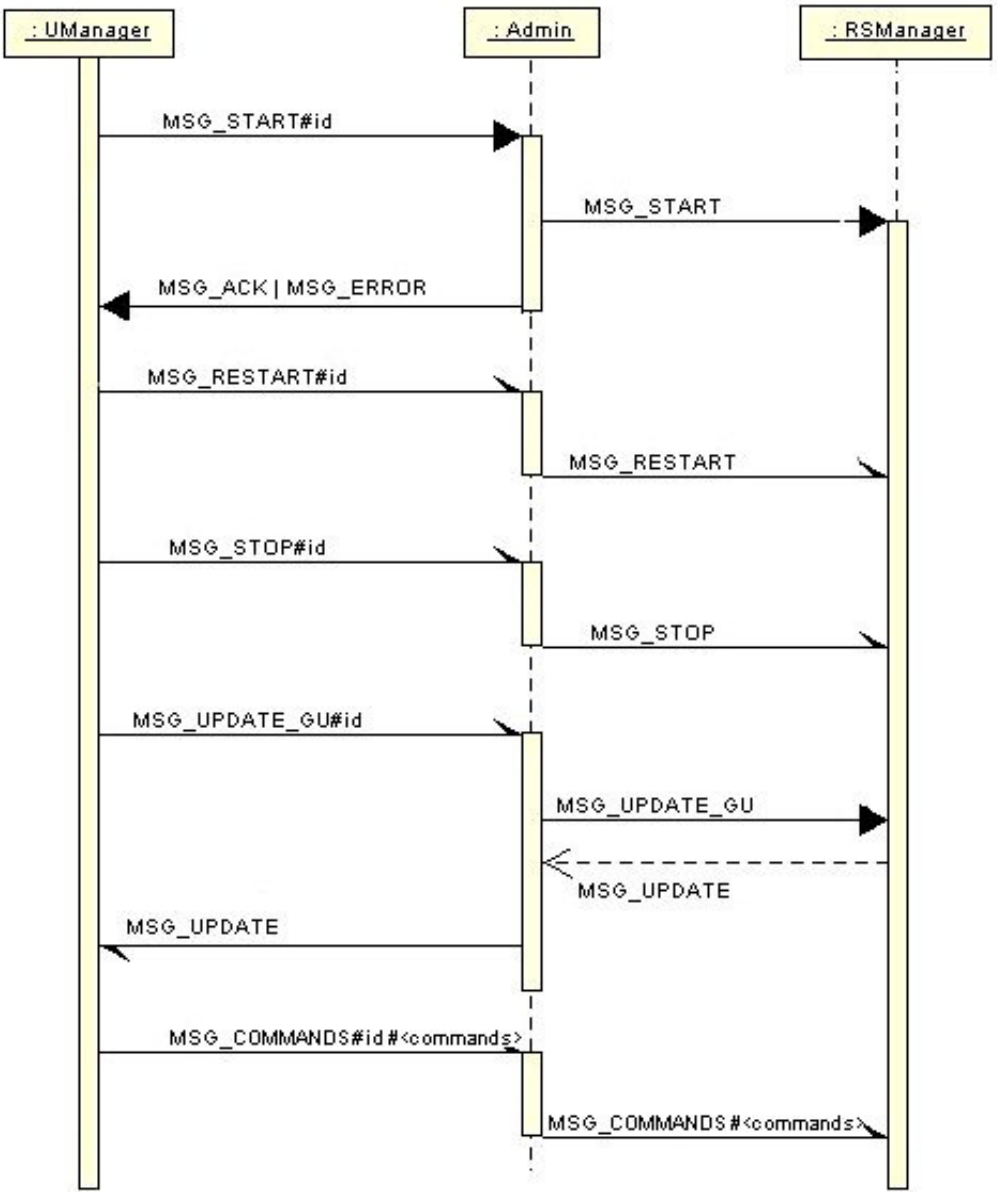


Figure 2.2.7 : Messages initiés par le Gestionnaire Utilisateur

La figure 2.2.8 montre les échangées réalisés pour répondre aux requêtes du Gestionnaire Système Réel : mise à jour des simulations locales (MSG_UPDATE) et communication des erreurs (MSG_ERROR). A la différence du message MSG_UPDATE_GU, valable pour un seul utilisateur, le message MSG_UPDATE synchronise les simulations de tous les Gestionnaires Utilisateurs participants.

Pour permettre le contrôle à distance des dispositifs, notre système a besoin d'un schéma de communications fiable. En effet, il est nécessaire de garantir la réception par l'utilisateur de tous les messages expédiés par le système distant. De la même manière, les instructions doivent être

reçues par les dispositifs en ordre et sans perte : la séquence des instructions à exécuter doit correspondre exactement à la séquence générée par l'opérateur.

Pour satisfaire ses besoins de communications, notre système utilise les sockets et le protocole de transport TCP. L'utilisation d'UDP dans les composants de communications peut être envisagée, elle demande toutefois le traitement par l'application des informations provenant du réseau. Il faut noter aussi que le développeur peut modifier le protocole de communications utilisé par ASSET en fournissant les classes Java qui implémentent les services et en les référençant ensuite dans le fichier de configuration.

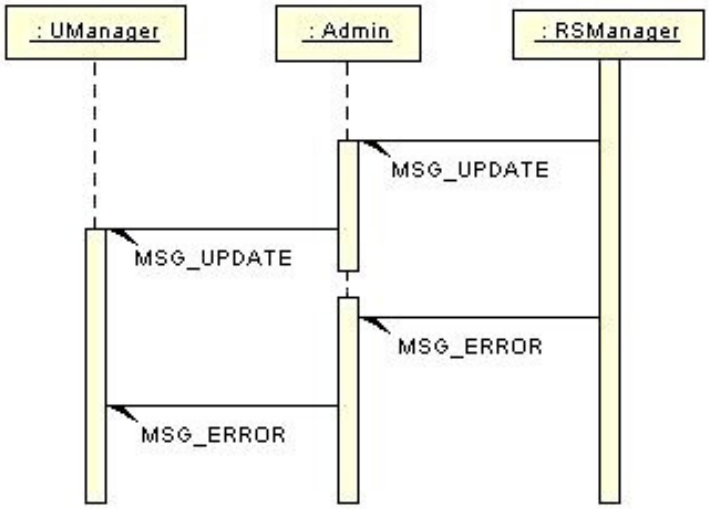


Figure 2.2.8 : Messages initiés par le Gestionnaire Système Réel

Gestion de dispositifs

La configuration matérielle d'une application de téléopération n'est pas statique. Les avancées de la recherche sur les périphériques d'interaction et les composants robotiques amènent des changements dans les systèmes. Il est donc très important de permettre aux développeurs d'intégrer et de tester les différents dispositifs à leur disposition d'une façon simple. Pour ce faire, le système ASSET réalise un traitement générique des dispositifs physiques associés à l'application, afin de séparer les services fournis de la manière dont ils sont implantés. L'application n'a pas de connaissance sur les dispositifs physiques, elle communique avec eux en utilisant de médiateurs : les « dispositifs virtuels ». Ces dispositifs permettent la collaboration entre les dispositifs réels et les autres composants d'un module afin que les applications soient

indépendantes d'une configuration matérielle particulière et donc très peu affectées pour ses changements.

L'information des dispositifs de l'application, pendant l'exécution, se trouve dans l'espace de données (classe DataSpace). Les dispositifs d'interaction qui fournissent un retour à l'utilisateur doivent implanter les méthodes de l'interface MsgDispIListener pour recevoir les messages provenant des objets de simulation. De leur coté, les dispositifs effecteurs doivent implanter les méthodes de l'interface MsgDispAListener pour recevoir les instructions à exécuter sur le site de travail. L'espace de données, lui, implante les méthodes de l'interface DeviceListener pour récupérer les messages provenant des dispositifs.

Figure 2.2.9 : Classe VirtualDevice

Les classes de contrôle des dispositifs, comme les autres éléments configurables de notre système, sont chargées à l'initialisation de l'application. L'application ne requiert donc pas d'être recompilée en sa totalité lorsque le dispositif ou la classe de contrôle change. De plus, puisque l'information des périphériques est contenue dans le fichier de configuration, l'intégration de nouveaux dispositifs ou la configuration de ceux déjà existants est facilitée. Une classe de contrôle se charge de lire et exécuter les commandes selon les capacités et caractéristiques du dispositif contrôlé. Elle contient les méthodes pour récupérer les commandes (lecture à la fréquence requise d'un dispositif du type « polling » ou traitement de l'événement associé au changement d'état du dispositif) et pour les traiter afin qu'elles puissent être comprises par les classes de comportement des objets de simulation. Le dispositif virtuel permets la communication entre le système et les dispositifs sans imposer une structure quiconque aux classes de contrôle. Ainsi, elles peuvent exploiter toutes les fonctionnalités offertes par un dispositif particulier.

Généricité

Pour offrir un maximum de flexibilité dans la configuration des différents éléments d'une application, le système ASSET utilise un fichier d'initialisation. Ce fichier permet la configuration des objets de simulation, des dispositifs, de l'état, des conditions de synchronisation et des différentes variables du système (pas de simulation, éclairage, points de vue). Cette approche permet l'évaluation des diverses configurations possibles d'un système sans modifier le code source et sans re-compilation.

Nous avons défini un fichier de configuration dont le format est très simple, facile à comprendre et à modifier. Pour décrire en détail ce fichier, nous utiliserons comme exemple une application minimale qui utilise un dispositif d'interaction pour contrôler un robot mobile. Le fichier de configuration comprend quatre parties : la configuration de la simulation, la configuration de l'espace de données, la configuration de la visualisation et de l'information relative aux objets. Dans le fichier de configuration distant nous trouvons une partie additionnelle dédiée à la configuration de l'état et à la mise à jour de l'application. La simulation est concernée par la dernière partie (dédiée aux objets de l'environnement virtuel) et par la première car celle-ci contient le pas de simulation qui sera utilisé pour configurer l'horloge :

```
ASSET Configuration File
[*******deltat (millis)]
500
```

Figure 2.2.10 Configuration de la simulation

Dans la deuxième partie du fichier, nous trouvons l'information nécessaire à la configuration des dispositifs (figure 2.2.11). Le fichier de configuration local contient l'information des dispositifs d'interaction et le fichier distant contient l'information des dispositifs capteurs et effecteurs. L'information des dispositifs distants peut exister dans le fichier local et dans ce cas, elle sera affichée dans l'interface graphique utilisateur.

a)

```
[*******dataSpace configuration]
1
1.joystick|TScale:0.001#accum:false#Gain:3000#ifrFile:effects.ifr"|1.robot
khepera|asset.devices.FFDevice|0.001#false#3000#effects.ifr"
0
```

b)

```
[*******dataSpace configuration]
0
1
1.robotKhepera|TScale:2#RScale:1|1.robot
khepera|asset.devices.KhepDeviceFile|2#1
```

Figure 2.2.11 : Configuration des dispositifs a) d'interaction b) effecteurs

La troisième partie du fichier contient les données de configuration de la visualisation (figure 2.2.12). Dans cette partie nous trouvons les différents points de vue définis dans la scène et le point de vue à utiliser pour la première visualisation de l'environnement virtuel. Ensuite nous avons la description de l'éclairage de la scène tridimensionnelle. La dernière partie du fichier contient l'information de configuration des objets de simulation (figure 2.2.13). Pour chaque objet nous avons les différents attributs qui caractérisent ses relations avec les dispositifs et les autres objets, sa classe de comportement et une variable pour activer la détection de collisions avec cet objet. Nous trouvons ensuite l'information de l'état de l'objet et de sa géométrie qui peut être définie dans un fichier VRML, Java3d ou dans le fichier de configuration si la géométrie est une primitive Java3d simple (boîte, cône, cylindre ou sphère). Dans certains cas, la géométrie d'un objet peut faire partie d'un autre objet (une roue fait, par exemple, partie de la géométrie du robot), c'est-à-dire que deux objets de simulation existent dans un seul fichier géométrique. Pour permettre ce genre de dépendances, le type de géométrie peut avoir la valeur « inParent » pour indiquer la relation entre la géométrie de deux objets.

```
[*******object viewpoint id]
null
[*******viewer: position \n orientation]
1
pointVue1
4.30,27.74,159.044
0.006,-0.988,-0.154
0.18
[*******background color]
null
0.4,0.4,1
[*******lights in the scene]
3
[*******light description: ambient/point/directional, color, direction]
ambient
0.8,1,0.6
directional
0.8,0.8,0.8
0,0,-1
point
1,1,1
0,500,0
0,0,0
```

Figure 2.2.12 : Configuration de la visualisation

```
[*******objects in the scene]
4
local
[*******object description:
idParent|idObject|controlType|controller|associated|behaviorClass|collisionActive]
null|1.robot khepera|autonome|false|true|asset.simulation.DefaultBehavior|true
-13.2967, 0.243212, 20.1478
1.5,0,0
0,0,0
0,-1,0
1.5708
0,1,0
0
0
Vrml File
.\classes\asset\resources\models\khepera.wrl
[*******object description:
idParent|idObject|controlType|controller|associated|behaviorClass|collisionActive]
1.robot khepera|2.Wheels|autonome|false|true|asset.simulation.DefaultBehavior|true
.025, 0, .007
0,0,0
0,0,0
0,0,1
1.5708
0,1,0
0
0
inParent

…
```

Figure 2.2.13 : Configuration des objets de simulation

L'état et les conditions de mise à jour se trouvent dans le fichier de configuration du système distant (figure 2.2.14). L'utilisateur définit, pour chaque variable de l'état, le seuil d'erreur et les méthodes permettant la récupération de la valeur et la comparaison. Chaque variable possède aussi un type et une entité (objet ou dispositif) à laquelle elle est associée. Cette information permet d'utiliser les méthodes de façon dynamique, c'est-à-dire sans connaître, au moment de la compilation, les classes et les noms des méthodes qui vont être utilisés.

```
[interesting variables]
3
1.robotKhepera|1.KhepDevice|java.lang.Boolean|collision|collision|true
1.robotKhepera|1.KhepDevice|asset.math.EnvVector3f|position|position|distance
|0.1,0.1,0.1
1.robotKhepera|1.KhepDevice|java.lang.Float|angle|angle|null|0.1
```

Figure 2.2.14 : Configuration de la mise à jour

La configuration des communications des modules se réalise avec des fichiers séparés, les fichiers de propriétés. Ces fichiers (figure 2.2.15) contiennent l'information nécessaire à l'établissement des communications entre les trois modules : la classe qui implémente le protocole de communications à utiliser et les paramètres qui permettent sa configuration.

```
imgDir=classes/asset/resources/images/
communicationClass=asset.comm.commTCP
IP=123.123.123.1*1089
flagAdmin=false
Admin=0|0:0:0:0|0:0:0:0|0|123.123.123.1*1083|asset.comm.commTCP
RealSys=0|0:0:0:0|0:0:0:0|0|123.123.123.1*1088|asset.comm.commTCP
```

Figure 2.2.15 : Exemple de fichier de propriétés pour la configuration des modules

Conclusion

Dans ce chapitre nous avons présenté les services, indispensables dans une application de téléopération, fournis par le système ASSET : simulation, gestion de dispositifs, gestion de données et communications. Nous avons aussi introduit les dispositifs virtuels, qui éliminent les liens directs entre l'application et la configuration matérielle utilisée. Ceci permet donc de modifier les dispositifs sans affecter l'application en sa globalité. Nous avons décrit également le mécanisme de synchronisation et les différents types d'objets présents dans notre système.

Pour construire une application spécifique, les services d'ASSET s'intègrent avec les ressources particuliers à l'application. Par exemple, pour évaluer un comportement, le développeur doit fournir uniquement le modèle géométrique et la classe de comportement de son entité. L'intégration de ces ressources est réalisée en utilisant le fichier de configuration et le chargement dynamique des classes développées par l'utilisateur. Cette approche permet le prototypage rapide et la réalisation de diverses modifications sans re-compilation totale de l'application.

Le chapitre suivant détaille l'implantation des services décrits et les choix techniques réalisés pour la mise en œuvre du système ASSET.

chapitre 3

Mise en OEuvre

Dans les chapitres précédents nous avons présenté la structure et les mécanismes définis dans le système ASSET pour le développement de systèmes de télérobotique. Ce chapitre décrit la plate-forme logicielle et matérielle ainsi que les outils utilisés dans l'implémentation de notre projet.

Pour la mise en œuvre du système ASSET, nous avons utilisé Java [Java]. Ce langage a été choisi car son approche objet permet de réaliser aisément le découpage des fonctionnalités définies par notre architecture. De même, l'implantation des notions de la conception orientée objet comme l'abstraction de données, l'héritage, l'encapsulation et la liaison dynamique est facilitée. Logiquement, ces concepts peuvent être implantés facilement avec d'autres langages à objets comme C++ mais nous avons retenu Java parce qu'il est neutre par rapport aux plates-formes matérielles. La portabilité de Java permet la création d'applications capables de fonctionner sur de multiples plates-formes sans être modifiées ce qui est un avantage de notre système.

Java fourni un ensemble d'APIs de base qui couvrent un large éventail de fonctionnalités telles que la gestion des entrées/sorties, de threads, du réseau et pour la construction d'interfaces graphiques utilisateur. Nous avons utilisé dans le développement de ASSET plusieurs paquetages optionnels de Java dont Java3D [Java3D], une bibliothèque pour la création et l'affichage de scènes tridimensionnelles.

Gestion de l'environnement en Java3d

La réalité virtuelle permet aux utilisateurs de percevoir, comprendre et contrôler de façon intuitive les systèmes complexes. Etant donné que le point de vue peut être changé par l'utilisateur à tout moment (à la différence des visualisations réelles utilisant des images vidéo), celui-ci peut analyser les situations plus efficacement. Nous avons donc utilisé des environnements virtuels comme interface du système ASSET. La mise en œuvre de cette interface est réalisée avec la bibliothèque graphique Java3D qui fournit des méthodes de haut niveau pour créer et afficher la géométrie tridimensionnelle des objets dans un monde virtuel.

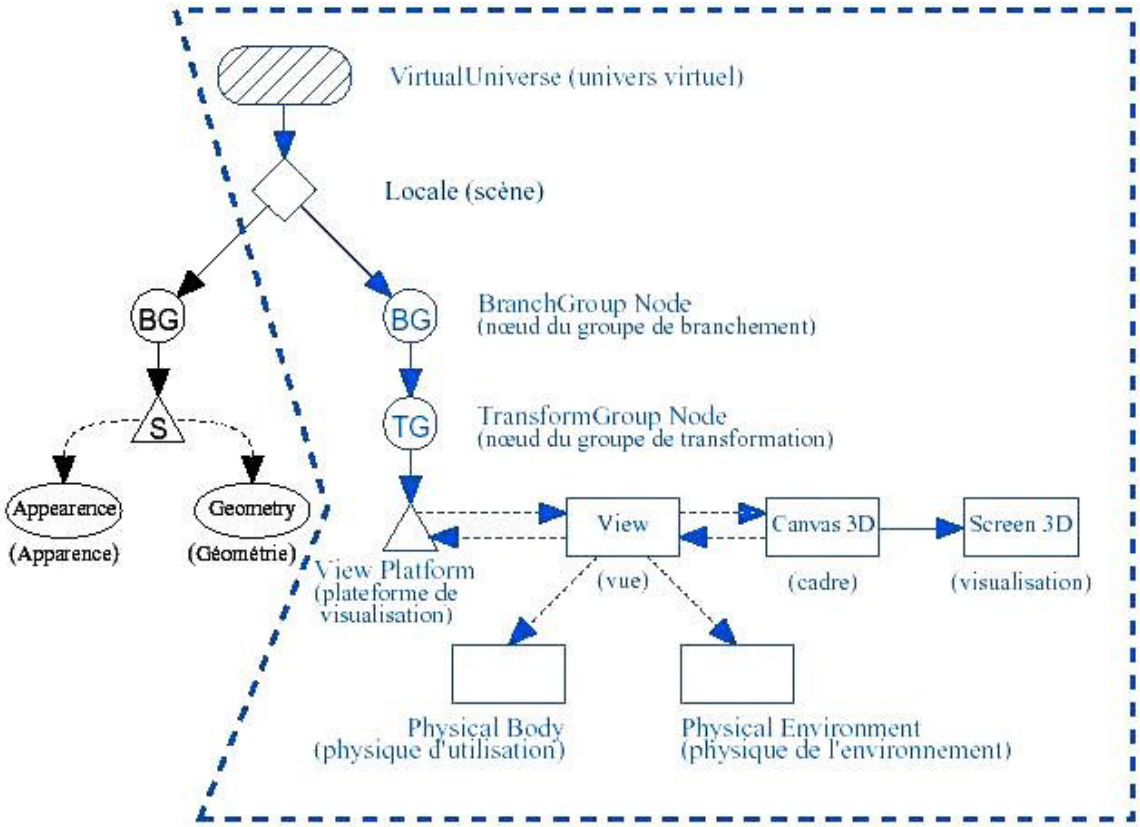


Figure 2.3.1 : Graphe de scène de Java3D [Java3DTutorial01]

Un programme Java3D crée des instances d'objets Java3D et les place dans un graphe de scène qui spécifie le contenu du monde virtuel et la façon dont il doit être rendu. Le graphe de scène est composé d'objets qui définissent, entre autres, la géométrie, l'apparence, la position et l'orientation des objets graphiques. Chaque graphe possède un *VirtualUniverse* (univers virtuel) auquel est attachée une certaine quantité d'objets appelés *Locale*. Un objet Locale sert de point de référence pour déterminer la position des objets 3D dans l'univers virtuel. La plupart des programmes Java3D utilisent un seul Locale, donc un seul système de coordonnées pour l'univers

virtuel. Dans le graphe de scène nous trouvons aussi les *BranchGroup*, les racines des branches qui contiennent les objets destnés à être visualisés, les lumières à appliquer, les comportements à exécuter, les sons à jouer, etc. Les branches définissant le contenu de l'univers virtuel sont appelées **branches de volume** et celles qui contiennent un objet *ViewPlatform* (objet spécifiant la position et l'orientation de l'utilisateur) **branches de visualisation**. La branche de visualisation contient une *image plate*, une surface rectangulaire sur laquelle est projeté le volume pour créer l'image de rendu. L'objet *Canvas3*, qui produit une image dans une fenêtre sur l'écran de l'ordinateur, affiche l'image plate. La classe SimpleUniverse de l'API Java3D simplifie la construction du graphe de scène. Elle construit le VirtualUniverse, crée un objet Locale et une branche de visualisation et les attache à l'objet VirtualUniverse. Ceci est illustré dans la figure 2.3.1.

Une classe indispensable pour la création du contenu de l'univers virtuel est la classe *TransformGroup* (groupe de transformation), une sous-classe de la classe Group qui maintient des transformations géométriques. Les objets *Transform3D,* représentant des transformations comme la translation et la rotation, sont utilisés dans la création des objets TransformGroup. Lorsque plusieurs transformations différentes sont définies pour un même objet visuel, elles peuvent être combinées dans une seule matrice de transformation et portées par un seul objet TransformGroup. L'exemple de la figure 2.3.2 montre une combinaison de transformations pour réaliser deux rotations simultanées sur les axes x et y. Deux objets Transform3D sont crées et leurs attributs sont spécifiés pour définir les transformations de rotation. Puis les rotations sont combinées par la multiplication des objets Transform3D. La combinaison des deux transformations est ensuite chargée dans l'objet TransformGroup.

```
Transform3D rotate =new Trasnform3D()
Transform3D tempRotate = new Transform3D()
rotate.rotX(Math.PI/4.0d) ;
tempRotate.rotY (Math.PI/5.0d)
rotate.mul(tempRotate)
TransformGroup objRotate = new TransformGroup(rotate)
```

Figure 2.3.2 : Définition des transformations avec Java3D

Après sa création, la branche de volume est insérée dans l'univers virtuel. L'insertion la rend *vivant* et par conséquent tous les objets qui partent de cette branche deviennent vivants. Les objets *vivants* ont deux caractéristiques importantes : ils sont susceptibles d'être visualisés et leurs paramètres ne peuvent plus être modifiés (à moins que cette aptitude leur ai été attribuée de

façon spécifique avant que l'objet devienne vivant). La branche de volume peut aussi être compilée. La compilation d'une branche convertit tous ses objets et composants dans une forme plus optimale pour le rendu. Lorsqu'une branche est rendue vivante ou compilée, le système de rendu Java3D la convertit en une représentation interne plus efficace en vue d'améliorer les performances du rendu. Le rendu Java3D commence à s'exécuter dans une boucle infinie dès qu'une branche contenant une instance de l'objet *View* devient vivante dans l'univers virtuel. Une fois démarré, le rendu Java3D exécute de façon cachée les opérations décrites dans la figure 2.3.3.

```
While (true) {
   Traitement des entrées
   If (demande de finalisation) break
      Réaliser les comportements
      Traverser le graphe de scène et
      Afficher les objets visuels
   }
Nettoyage et finalisation
```

Figure 2.3.3 : Le processus du rendu en Java3D

Les comportements des applications Java3D sont spécifiés avec des objets *Behavior*. Un objet Behavior dans un graphe de scène permet la modification du graphe et des objets visuels contenus, en réponse à un stimulus (ou à une combinaison de stimulus) comme le mouvement de la souris, une collision, etc.

La création du graphe de scène dans le système ASSET est réalisée par la classe VisualControl. Cette classe crée et gère les objets Canvas3D et SimpleUniverse ainsi que les BranchGroup pour les différents objets visuels. Lorsque la création de la liste des objets de la simulation est finie, cette structure est parcourue pour créer la branche de volume du graphe de scène. A chaque intervalle de simulation, les objets de simulation utilisent l'information de position et d'orientation pour créer les transformations de translation et rotation (Transform3D). Ces transformations sont ensuite combinées et utilisées pour mettre à jour les groupes de transformation (TransformGroup). Pour incorporer le contenu VRML dans l'application Java3D nous utilisons la bibliothèque de chargement fournie par Xj3D, un projet dédié a l'implémentation du code Java pour l'affichage et l'intégration de contenu VRML dans toute sorte d'applications[Xj3D]. Cette bibliothèque charge la scène décrite dans un fichier VRML et fournit un BranchGroup pour sa manipulation. Comme résultat, nous pouvons traiter tous les objets visuels de l'environnement de manière identique tout en permettant au développeur d'utiliser des modèles géométriques VRML.

Interface Utilisateur

L'interface utilisateur du système ASSET comprend deux parties : une partie est dédiée à l'affichage de l'environnement 3D et l'autre à l'affichage de l'information sur les objets de simulation, les dispositifs et l'environnement distant à travers une interface graphique utilisateur classique (figure 2.3.4). L'interface graphique 2D de ASSET est réalisée avec Swing, une bibliothèque fournie par Java pour la création d'interfaces graphiques utilisateur. Les composants Swing sont intégralement écrits en Java ce qui assure leur portabilité et facilite, par ailleurs, la programmation [JavaTutorial02]. Swing est indépendant du système d'exploitation mais les composants peuvent être configurés pour présenter un aspect conforme à celui de l'interface du système utilisé.

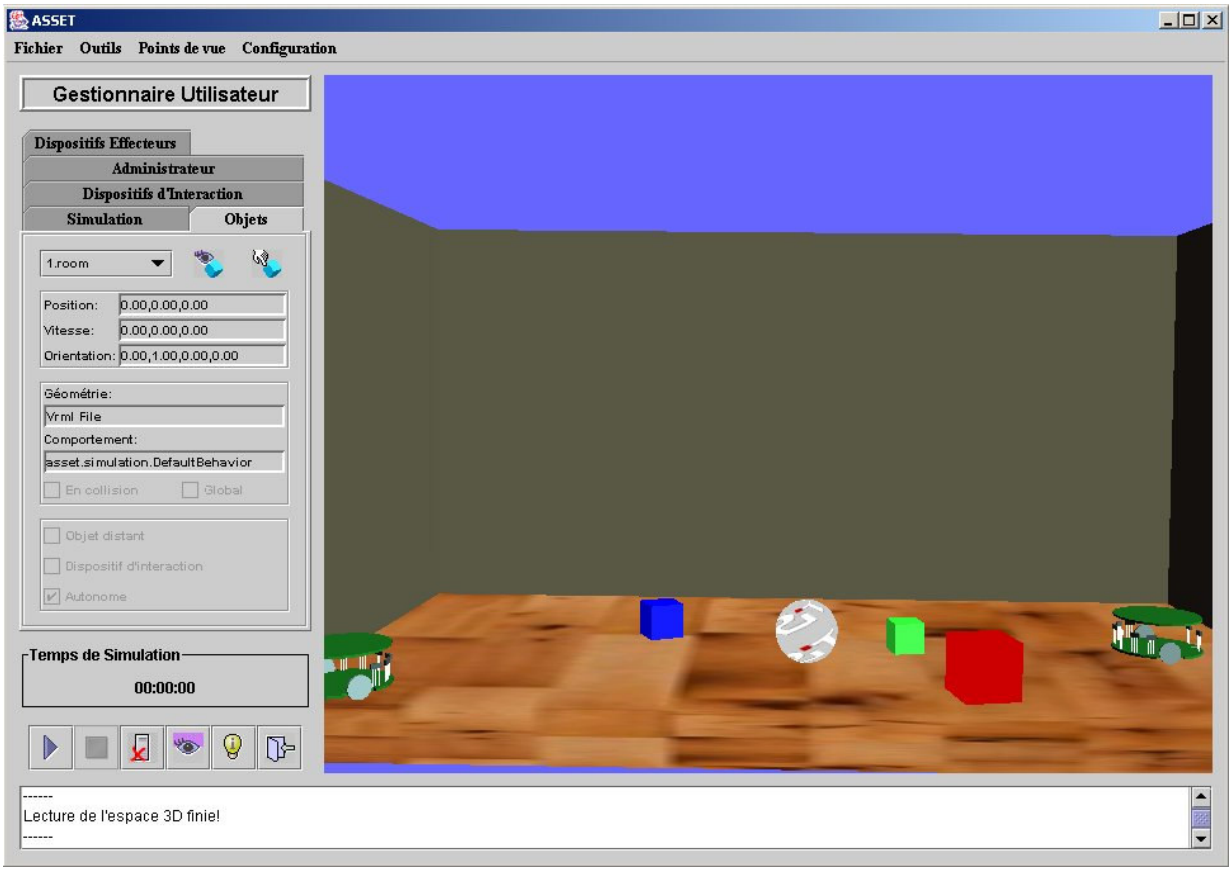


Figure 2.3.4 : Interface graphique utilisateur de ASSET

Grâce à l'interface graphique, l'utilisateur peut contrôler la simulation et connaître, entre autre, l'état de chaque objet et dispositif. La barre de boutons et le menu principal permettent principalement à l'utilisateur de démarrer ou arrêter la simulation, de modifier les points de vue et l'éclairage, et d'établir ou finaliser les communications avec l'Administrateur (figure 2.3.5).

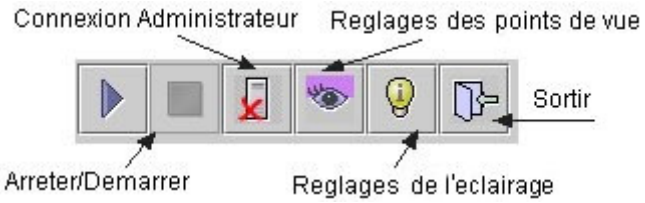


Figure 2.3.5 : Barre de boutons

Le panneau d'affichage de l'information possède plusieurs sections : Objets, Simulation, Dispositifs d'interaction, Administrateur et Dispositifs effecteurs. Les sections consacrées à l'information des dispositifs affichent l'état de chaque dispositif et les objets auxquels ils sont associés. L'information des associations entre les dispositifs et les objets est aussi présentée dans la section Simulation ainsi que le pas de simulation et le fichier de configuration de l'application. La section Administrateur permet de connaître le temps de simulation (approximatif) du site distant et de générer une mise à jour. Enfin, la section Objets présente à l'utilisateur l'état et les attributs des objets de la simulation. L'utilisateur peut choisir un objet spécifique et lui associer un point de vue en cliquant sur les boutons situés à droite du cadre de texte qui contient l'identificateur de l'objet (figure 2.3.6).

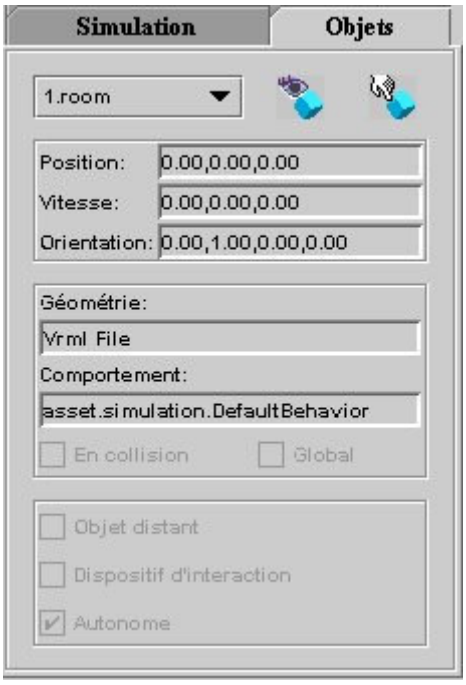


Figure 2.3.6 : Panneau d'information de l'interface du système ASSET

Dynamicité et adaptabilité du système

Les options de configuration fournies par ASSET permettent au développeur d'expérimenter avec les différentes possibilités de la mise en œuvre de son application. Toute l'information et les données spécifiques à l'application résident dans un fichier, chargé à l'initialisation (voir chapitre précédant). Ce fichier de configuration contient des informations sur les objets de simulation, les dispositifs et le réseau ce qui permet de réaliser une grande variété de modifications sans changer et recompiler tous les composants de l'application. Un exemple de ceci est la configuration des dispositifs : dans le fichier de configuration le développeur définit les dispositifs et les classes qui vont les contrôler. Ces classes seront chargées pendant l'initialisation assurant ainsi l'indépendance de l'application par rapport à la configuration matérielle utilisée.

Pour charger et avoir accès de façon dynamique aux classes fournies par le développeur, sans avoir connaissance de celles-ci pendant la compilation, nous avons utilisé l'API Reflection de Java. Cette API nous permet d'utiliser une classe (créer des instances, appeler des méthodes) à partir de la chaîne de caractères qui la représente. Nous pouvons donc utiliser les classes développées par le programmeur en récupérant leur nom dans le fichier de configuration et en appelant les méthodes de l'interface qu'elles implémentent. La figure 2.3.7 montre les différentes étapes de l'utilisation de la classe de contrôle d'un dispositif. Le dispositif virtuel (classe VirtualDevice) crée une instance de la classe de contrôle dont le nom est stocké dans la variable nomClass. Pour ce faire, elle récupère le constructeur et initialise le type et la valeur des divers paramètres. Puis, le constructeur est invoqué et le résultat est stocké dans la variable disp de type Object. Le processus est similaire pour utiliser une autre méthode de la classe : le type et la valeur des paramètres sont utilisés pour invoquer la méthode « exécute » sur l'objet disp qui a été créé.

```
String nomClass;
String parameters;

public boolean createDevice() {
       boolean creado = false;
       Class tipos[] = new Class[2];
       Object param[] = new Object[2];

       claseDisp = Class.forName(nomClass);
       tipos[0] = Class.forName("java.lang.String");
       tipos[1] = Class.forName("asset.devices.VirtualDevice");
       Constructor consObj = claseDisp.getConstructor(tipos);
       param[0] = parameters;
       param[1] = null;
       disp = consObj.newInstance(param);
```

Figure 2.3.7 : Utilisation de l'API Reflection

Environnement matériel

Dans ce paragraphe nous présentons la configuration matérielle utilisée dans l'implantation du système ASSET. Notre plate-forme de travail est constituée de PC Windows et l'affichage graphique de l'environnement virtuel s'effectue sur un écran conventionnel d'ordinateur. Ce choix est basé sur le fait que, dans l'immersion partielle, l'utilisateur peut continuer à être en contact avec le monde réel. Ceci lui permet de s'éloigner du système facilement et à tout moment ce qui est un avantage pendant l'étape de développement de l'application.

Plusieurs dispositifs font aussi partie de cette configuration : le joystick SideWinder Force Feedback Pro de Microsoft [SideWinder], le dispositif Spacemouse Classic de Logicad3d [Spacemouse] et le robot Khepera [Khepera], un robot miniature développé par la société K-Team.

Le robot Khepera

Le robot Khepera est un petit robot mobile : 5.5 cm de diamètre, 30 mm de hauteur et un poids d'à peu près 70g. Il peut atteindre une vitesse de 60 cm/s. Le Khepera est équipé d'un microprocesseur, de deux moteurs, et d'un ensemble de 8 capteurs fournissant des informations sur la proximité des obstacles. La figure 2.3.8b montre la position et l'orientation des capteurs intégrés dans le robot Khepera.

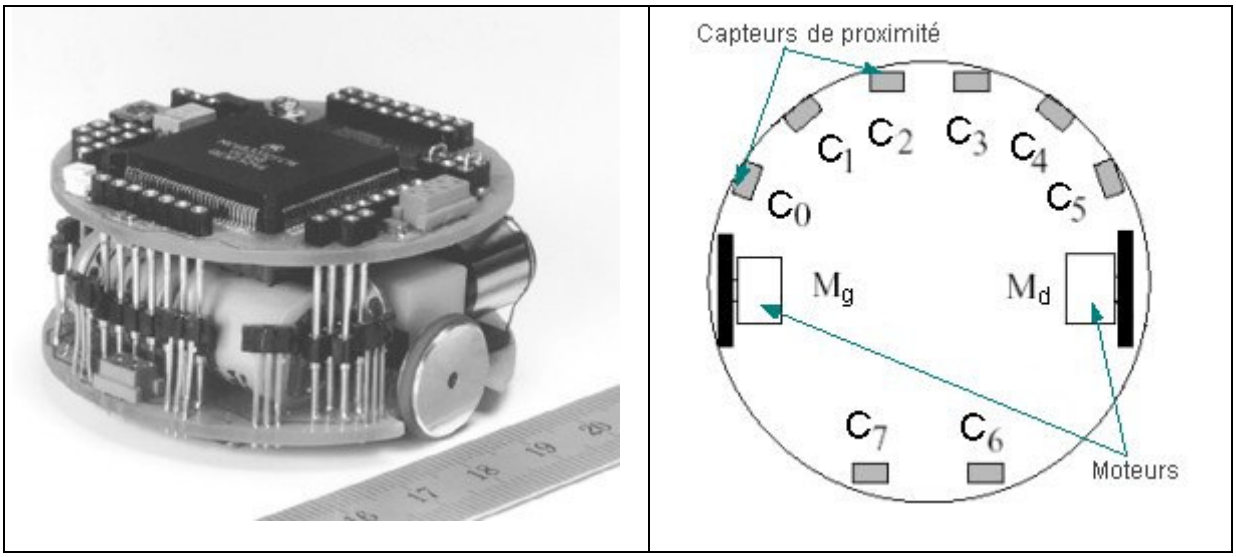


Figure 2.3.8 : Le robot Khepera

Les capteurs du Khepera contiennent un émetteur de lumière infrarouge et un récepteur intégrés qui lui permettent de mesurer la lumière ambiante ainsi que la lumière reflétée par les obstacles.

Les mesures de la lumière ambiante utilisent des valeurs entre 0 et 512; les valeurs entre 495 et 512 correspondant à l'«obscurité totale » et celles au-dessous de 50 à l'éclairage maximum. Pour les mesures de distance les valeurs se situent entre 0 et 1023 ; les lectures entre 0 et 5 signifiant que les capteurs ne détectent aucun objet.

Les roues du robot sont contrôlées par deux moteurs indépendants. Elles offrent au Khepera la possibilité de tourner sur lui-même. Chaque moteur possède un compteur de *pulse*s, l'unité de déplacement du robot, correspondant à 0.08 mm. Il existe deux modes de contrôle du Khepera : le mode « vitesse », dans lequel on assigne une vitesse pour chacune de ses roues, et le mode « incrémental », dans lequel on indique un point de destination exprimé en pulses. Si nous voulons donc faire avancer le robot de 1 cm, le compteur de positions des roues doit être incrémenté de 125 pulses. Puisque dans l'ensemble des commandes du robot il n'existe pas d'instruction pour réaliser de rotations, il est nécessaire de déterminer la relation entre le nombre de pulses et l'angle de rotation. Cette relation est de 5.99 pulses par degré de rotation, étant donné que le diamètre du Khepera est de 55 cm et qu'un pulse correspond à 0.08mm (figure 2.3.9).

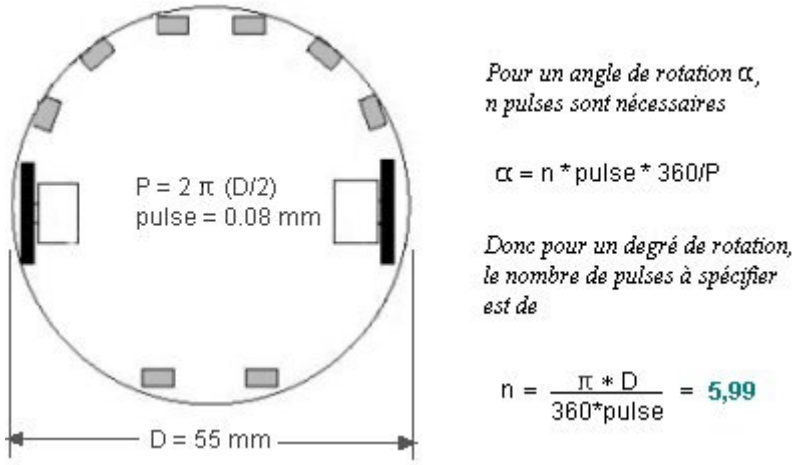


Figure 2.3.9 : Relation entre les pulses et l'angle de rotation

Le dernier élément à considérer pour l'utilisation du robot Khepera est la communication des ordres. Le robot est connecté à l'ordinateur en utilisant une liaison série. Un protocole de communication a été défini par K-team pour permettre le contrôle total du robot via cette liaison. Le protocole est composé de commandes et de réponses sous forme de codes ASCII. Les commandes permettent de configurer et lire la vitesse et les compteurs de position des roues, de connaître l'état du contrôleur du mouvement et de récupérer les valeurs des capteurs de proximité, entre autres. Pour échanger les messages entre la classe de contrôle et le robot, nous

avons utilisé la bibliothèque JavaComm [JavaComm], un paquetage optionnel de Java permettant la gestion des ports séries.

La SpaceMouse

La souris 3D SpaceMouse est un outil permettant la manipulation des modèles 3D dans un environnement virtuel. Ce dispositif contrôle 3 degrés de liberté de translation (x, y et z) et 3 degrés de liberté de rotation (A, B et C) comme indiqué sur la figure 2.3.10.

Figure 2.3.10 : Le périphérique de contrôle 3D SpaceMouse

L'utilisateur manipule la partie mobile en forme de cylindre (cap) dans toutes les directions. Cette poignée peut être déplacée à gauche, à droite, en avant ou en arrière. Elle peut aussi tourner autour des axes. L'amplitude des mouvements est d'environ 1.5mm en translation et environ 4° en rotation. Les capteurs mesurant les déplacements sont basés sur un système optique.

La SpaceMouse est connectée à l'ordinateur par le biais d'une liaison série. Pour la gestion de ce dispositif nous avons modifié la classe de contrôle développé par Joerg Vogel [MglStreams] pour l'adapter à notre interface générique de dispositifs. Plusieurs méthodes ont aussi été ajoutées pour permettre la configuration (dans le fichier d'initialisation) des divers paramètres, notamment le port de connexion et la relation entre les déplacements de la poignée de la SpaceMouse et ceux de l'objet contrôlé.

Le Joystick à Retour d'Effort

Le joystick à retour d'effort permet de donner à l'opérateur qui le manœuvre une sensation, sous forme de retour d'effort, des déplacements réalisés ou des effets de sa commande. Le joystick comporte un axe mobile qui peut être déplacé dans trois directions : deux translations orthogonales (x et y) et une rotation autour de z. Le joystick que nous avons utilisé est le SideWinder Force Feedback Pro de Microsoft (figure 2.3.11) qui est relié à l'ordinateur par un port jeu.

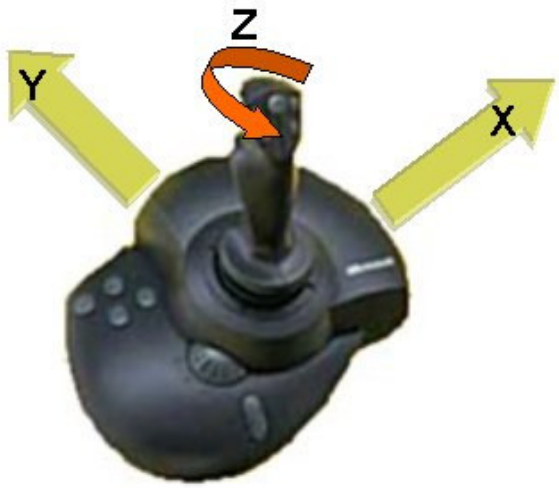


Figure 2.3.11 : Joystick à retour d'effort Microsoft SideWinder

Les forces que l'utilisateur perçoit comme résultat de son interaction avec le joystick sont appelés *effets*. Il est possible d'avoir des effets de contact, de frottements secs ou visqueux et des champs de potentiel répulsif, entre autres. Pour créer et jouer les effets sur le joystick SideWinder, nous avons utilisé le composant DirectInput de DirectX.

DirectInput est composé de trois objets : DirectInput, DirectInputDevice et DirectInputEffect. L'initialisation de base des dispositifs d'interaction se réalise avec DirectInput. Cet objet est utilisé ensuite pour créer l'objet DirectInputDevice qui donne l'accès aux dispositifs spécifiques connectés au système. Il existe un objet DirectInputEffect pour chaque effet à utiliser définissant sa direction, sa durée, sa puissance, etc. Les effets ainsi créés peuvent être joués et modifiés à tout moment. Etant donné que la création des effets et la gestion du joystick en utilisant DirectInput est une tâche très complexe, nous nous sommes servis du paquetage IFC de Immersion Corporation [IFC02]. IFC est un ensemble d'outils et de bibliothèques qui simplifient le développement des classes de contrôle pour une ample gamme de dispositifs à retour d'effort. IFC fourni un ensemble de classes C++ qui utilisent DirectX pour l'intégration et la gestion des dispositifs à retour d'effort, et des outils comme Immersion Studio pour faciliter la création d'effets. Avec Immersion Studio les effets, construits et testés dans un environnement graphique, sont sauvegardés dans un fichier afin d'être chargés et utilisés dans différentes applications.

La classe de contrôle que nous avons développé pour le joystick SideWinder est écrite en C++ pour avoir la fonctionnalité de retour d'effort. L'intégration de cette classe dans l'application Java est réalisée grâce à l'interface native de Java, JNI (Java Native Interface). L'interface JNI permet au code Java d'opérer avec des applications et des bibliothèques écrites dans d'autres langages de programmation [JNI]. Pour l'intégration de la classe de contrôle C++, nous avons développé et compilé une classe java qui déclare les méthodes natives propres au joystick. Nous avons ensuite utilisé l'outil javah pour générer un fichier d'en-tête qui est compilé avec la classe C++ pour créer une bibliothèque dll. Lorsque l'application est exécutée, la bibliothèque est chargée par la classe qui la demande. Ceci nous permet donc d'utiliser le retour d'effort dans les applications Java construites avec ASSET.

Conclusion

Dans ce chapitre nous avons présenté les différentes bibliothèques externes utilisées dans l'implantation de ASSET. Notre système a été développé en suivant une approche orienté objet, en utilisant le langage Java. Nous nous sommes appuyés sur plusieurs paquetages de la plate-forme Java pour réaliser le rendu graphique (Java3D), l'interface utilisateur (Swing) et la communication avec les dispositifs (JavaComm). L'API Reflection de Java [JavaReflection98] est utilisée pour implanter le chargement et l'utilisation dynamique des classes fournies par le développeur.

Ensuite, nous avons décrit la plate-forme de développement utilisée dans l'implantation de notre système. Dans cette plate-forme, constituée de PC Windows, l'environnement virtuel est affiché sur un écran ordinaire d'ordinateur. Différents dispositifs (robot Khepera, spacemouse Logicad3D, joystick Microsoft SideWinder) sont aussi présentés. Ces dispositifs ont été utilisés dans la construction des applications détaillées dans le chapitre suivant.

chapitre 4

Construction d'Applications avec ASSET

Dans les chapitres précédents nous avons présenté l'architecture générale pour la téléopération définie dans le système ASSET et les détails de son implantation. Pour tester cette architecture et évaluer l'implantation qui a été réalisée, nous avons développé plusieurs applications touchant différents aspects de la télérobotique. Trois exemples représentatifs de notre travail ont été retenus pour décrire l'utilité du système ASSET dans la construction d'applications. Ce chapitre présente d'abord l'environnement d'expérimentation et les modèles de simulation utilisés. Ensuite, nous décrivons les comportements qui ont été réalisés pour gérer des robots téléoperés et des robots autonomes. Enfin, nous présentons le système ROVE, un exemple de l'application d'ASSET dans le contexte de la téléopération coopérative.

L'environnement d'expérimentation

Dans cette section nous décrivons l'environnement utilisé au cours de nos expérimentations. La plate-forme est constituée de deux PC Windows (figure 2.4.1), auxquels sont connectés respectivement le robot Khepera et les dispositifs d'interaction (joystick et spacemouse). Même si cette configuration n'est nullement imposée par notre système, les modules (Gestionnaire Utilisateur, Administrateur, Gestionnaire Système Réel) peuvent être configurés et exécutés chacun sur une machine différente. Les modules peuvent aussi être exécutés sur la même machine, configuration très convenable pendant le développement de l'application.

Figure 2.4.1 : Schéma de la plate-forme expérimentale

Le chapitre précédent présentait les caractéristiques principales des dispositifs d'interaction utilisés dans nos applications. Maintenant, nous allons voir comment ces dispositifs sont intégrés dans une application. La classe de contrôle d'un dispositif d'interaction doit relever les commandes de l'utilisateur en utilisant les méthodes spécifiques du dispositif. Puis elle doit transformer ces commandes, transmises ensuite à l'objet de simulation, afin qu'elles puissent être interprétées.

Le traitement de commandes de la SpaceMouse est simplifié grâce à l'utilisation de la classe de contrôle Java3D fournie par Logicad3D. Cette classe récupère les données de la spacemouse, pour les stocker ensuite dans deux matrices représentant la rotation et la translation effectuées. Nous avons modifié légèrement la classe de contrôle pour récupérer uniquement les rotations autour de l'axe y, afin d'utiliser la SpaceMouse pour diriger un robot mobile.

La méthode `command` de la figure 2.4.2 est appelée à chaque intervalle de simulation pour transmettre les commandes générées par les dispositifs à l'espace de données. Dans cette méthode est réalisée la lecture des consignes (méthode pollAndProcessInput) et donc des transformations de rotation et translation. Les transformations sont combinées dans un objet Transform3D et utilisées ensuite pour créer un objet du type CommandInfo qui permet l'échange d'information entre l'espace de données, les dispositifs et les objets de simulation. Nous remarquons que les objets CommandInfo ne sont pas limités au stockage des transformations, la méthode setMessage peut être utilisée pour stocker des consignes sous la forme de chaînes de caractères.

```
public void pollAndProcessInput(){

              (…)
       rotationRead.rotY( MagellanData[4] * Rscale );
   translationRead.set(0,0,MagellanData[2] * Tscale);

              (…)

}


public CommandInfo command() {
       CommandInfo com;
       Vector3d vecTrans;

       pollAndProcessInput();
       currXform = new Transform3D(rotationRead,translationRead,1);
       com = new CommandInfo();
       com.setMessage(new Message(Constants.READ_TRANSFORM_3D));
       com.setTransform(currXform);
              (…)
        return com;
}
```

Figure 2.4.2 : Fragment de la classe de contrôle de la Spacemouse

Pour contrôler un objet en utilisant le joystick, nous relevons d'abord les valeurs des coordonnées (x_j,y_j) de celui-ci, en nous aidant des méthodes natives développées. Dans notre classe de contrôle, ces valeurs sont comprises entre [–100 et 100]. Pour calculer la direction et la distance parcourue par l'objet contrôlé par le joystick, nous avons implanté un schéma de contrôle très simple : la distance est indiquée par la coordonnée y et la rotation par l'angle défini par le vecteur $(0,0) \Rightarrow (x_j,y_j)$. La différence entre le système de coordonnées du joystick et celui de l'environnement virtuel fait que l'axe y_j du joystick corresponde à l'axe z_o de l'objet contrôlé. Le vecteur de déplacement ainsi défini est ensuite transformé (aligné dans la direction de la trajectoire du robot) et ajouté à la transformation de translation (figure 2.4.3).

```
Vector3d translation = new Vector3d();
Matrix3d rotation = new Matrix3d();
alfa =  Math.atan(yj/xj);
rotY.rotY(alfa);
rotation.mul( rotY, rotation );
vec = new Vector3d(0,0, yj *Tscale);
rotation.transform(vec);
currXform = new Transform3D(rotation,translation,1);
```

Figure 2.4.3 : Fragment de la clase de contrôle du joystick

Puis, nous créons l'objet Transform3D qui contiendra la rotation et la translation calculées. Le commande comportant cette transformation sera communiquée à l'espace de données pour être ensuite distribuée à l'objet de simulation et, si la communication a été établie, au système distant.

Chaque objet de la simulation doit, dans notre système, être associé à une classe de comportement. Donc pour faciliter la création des entités, ASSET fournit un comportement par défaut implantant un état qui évolue selon les équations cinématiques classiques (figure 2.4.4).

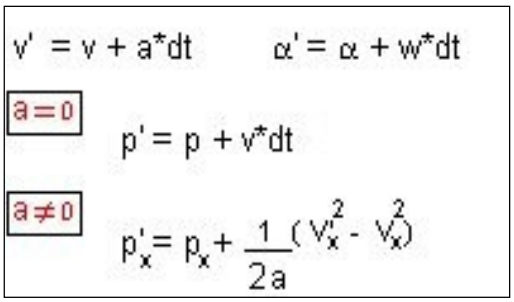


Figure 2.4.4 : Equations du calcul de l'état d'une entité mobile

Ce comportement correspond à un objet mobile, dont la position et la vitesse sont fonctions du temps. En utilisant ce comportement, le développeur peut créer rapidement différents objets autonomes (en configurant leurs paramètres dans le fichier d'initialisation) pour peupler son environnement virtuel.

Comportement d'un robot téléoperé

Le contrôle d'un robot téléoperé, par le biais d'un dispositif d'interaction, implique la collaboration de trois éléments : la classe du contrôle du dispositif d'interaction, le comportement du robot simulé et la classe du contrôle du dispositif effecteur (le robot réel). La classe de contrôle du dispositif d'interaction relève l'état du dispositif et le transforme en une consigne qui est transmise à l'objet de simulation. Le comportement du robot utilise, lui, les consignes pour générer le nouvel état du robot. Enfin, la classe de contrôle du dispositif effecteur traite, si nécessaire, les consignes et les communique au robot réel pour l'exécution.

Les comportements que nous avons développés vont permettre à l'utilisateur de générer des consignes par l'intermédiaire d'un dispositif d'interaction pour faire évoluer le robot dans un

environnement quelconque. Dans notre application, l'espace de travail est l'environnement illustré dans la figure 2.4.5.

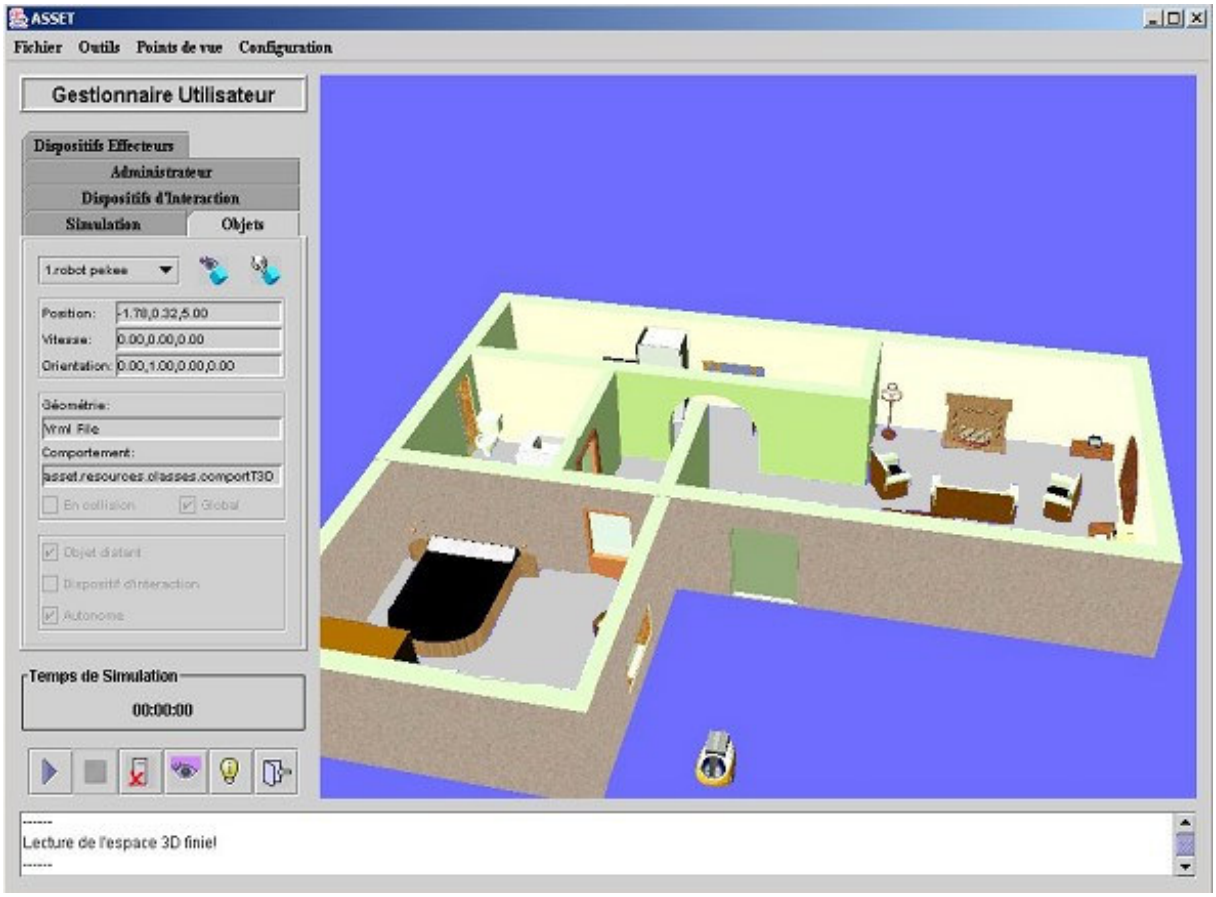


Figure 2.4.5 : Environnement virtuel pour un robot téléoperé

Dans le chapitre précédent nous avons vu que les dispositifs d'interaction génèrent des commandes contenant une transformation Transform3D. Le comportement de notre objet de simulation utilise donc cette transformation pour mettre à jour son état et, en conséquence, la visualisation (figure 2.4.6). Le comportement récupère l'objet CommandInfo et en extrait la transformation. Cette transformation est combinée ensuite à celle de l'objet, mettant ainsi à jour la structure de données du graphe de scène Java3D. Ensuite, la méthode fractionne l'objet Transform3D en un vecteur de translation et une matrice de rotation. Avec ces deux éléments et l'état précédent de l'objet, les valeurs de position, orientation et vitesse sont calculées.

```
void processt3d(CommandInfo command) {
   String ord;
   Transform3D t3d, t3dObj;
   float vit, t;
   Vector3f vec;
        Matrix3f mat;
   AxisAngle4f aangle, aangleObj;

   vec = new Vector3f();
   mat = new Matrix3f();
   aangle = new AxisAngle4f();
   aangleObj = new AxisAngle4f();

   ord = command.message().toString();
   if (ord.compareTo(Constants.READ_TRANSFORM_3D) == 0) {
      //récupération de la matrice de transformation
      t3d = new Transform3D(command.t3d());
      t3dObj = obj.t3d();
      t3d.mul(t3dObj, t3d);
              obj.modify(t3d);

      //recupere les donnees specifiques pour mettre a jour l'etat
      t3d.get(mat,vec);
      //translation
      state.setp(new EnvVector3f(vec));
      //vitesse
      vit = state.p().distance(obj.last().p());
      vit = vit/deltat;
      //direction
      estado.setv(estado.p());
      estado.v().subs(obj.last().p());
      estado.v().normalize();
      estado.v().scale(vit);
      //orientation
      aangle.set(mat);
      estado.setang(aangle);
      //vitesse angulaire
      aangleObj = obj.last().ang();
      float w = (aangle.angle - aangleObj.angle)/deltat;
      estado.setw(w);
   }
}
```

Figure 2.4.6 : Méthode pour mettre à jour l'état d'un objet en utilisant une transformation 3D

Nous utilisons ce même procédé dans la classe de contrôle du robot Khepera. Nous utilisons l'objet Transform3D pour récupérer la translation et la rotation à faire effectuer au Khepera. Ces consignes sont ensuite traduites dans le langage du robot pour être exécutées (figure 2.4.7).

```
Vector3d translation = new Vector3d();
Matrix3d rotation = new Matrix3d();

public void setTransformation(Transform3D t) {
   Vector3d vec3d = new Vector3d();
   Matrix3d mat3d = new Matrix3d();

   t.get(vec3d);
   t.get(mat3d);
   translation.set(vec3d);
   rotation.set(mat3d);
}


public void moveKhep() {
   float y,pulsesTrans, pulsesRot;
   double theta,distance;
   AxisAngle4f orientation;


   orientation = new AxisAngle4f();
   orientation.set(rotation);
   theta = orientation.angle;   //en radians
   y = orientation.y;
   distance = -1*translation.z*Tscale;  // transforme unités en cm
   pulsesTrans = 125*(float)distance;  // 1 cm équivaut à 125 pulses

   // translation
   setPosition(pulsesTrans);

       // rotation
   pulsesRot = theta*PULSES_RAD;

   if (y > 0)  //tourner a gauche
      setPosition(-pulsesRot, pulsesRot);
   else    //tourner a droite
      setPosition(pulsesRot, -pulsesRot);


   if (collisionTest() == true)
        dispClass.error();
}


public void setPosition(int posLeft, posRight) {

   int rg,rd;
   rgAccum += posLeft;
   rdAccum += posRight;
   rg = Math.round(rgAccum);
   rd = Math.round(rdAccum);
   commKhep("C,"+ rg +","+ rd +"\n");
}
```

Figure 2.4.7 : Fragment de la classe de contrôle du robot Khepera

Comportement d'un robot autonome

La mission définie pour le robot autonome, dans le travail réalisé par Alexander Boucard [Boucard02], consiste à emprunter un chemin et à réaliser une tâche simple. Le but du robot est donc de suivre une trajectoire, modélisée par des points de contrôle, en évitant les obstacles disposés sur le parcours puis de réaliser une action : l'exploration du terrain environnant et la découverte d'un objet. La simulation correspondant à ce comportement autonome est représentée sur la figure 2.4.8.

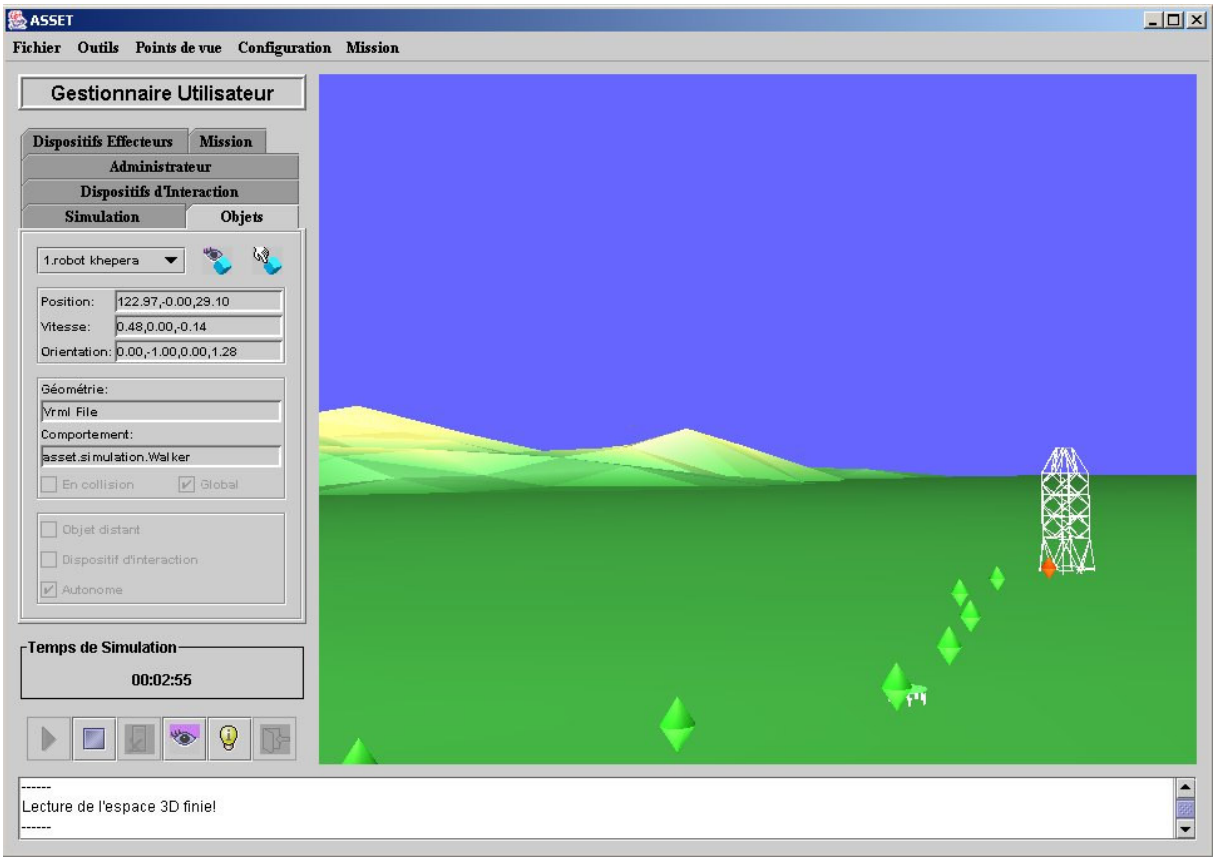
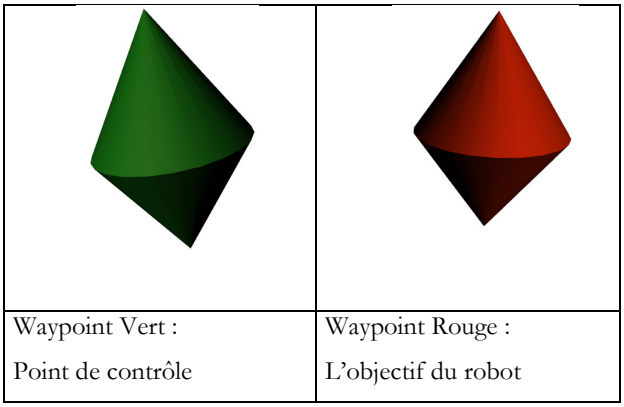


Figure 2.4.8 : Application de suivi de trajectoire dans ASSET

La trajectoire du robot doit être modifiée à chaque fois qu'il rencontre un objet ou qu'il arrive à un point de contrôle. D'ailleurs, chaque point doit être validé dans l'ordre de sa création. La maniabilité réduite des robots doit être compensée par une zone de validation autour d'un point de contrôle. Dès que le robot y pénètre, le point correspondant est considéré comme franchi. La zone de validation initiale est circulaire et centrée sur son point de contrôle. Le rayon de cette zone est modifiable par l'utilisateur. Ce stratagème permet au robot d'avoir une marge de manœuvre entre deux points et de franchir différents obstacles. Par exemple, un objet du décor

comme un pilier peut servir de point de passage : théoriquement, il ne peut pas être traversé, mais si l'utilisateur attribue un rayon supérieur au diamètre du pilier, le robot ne se bloquera pas. Le point de contrôle est souvent associé au centre géométrique de l'objet repère, mais il peut aussi être déplacé ailleurs pour satisfaire aux exigences d'un problème tel que le passage d'une porte, le franchissement d'un pont ou simplement, d'un point de vue plus pratique, pour faire passer le robot à l'aplomb de sa recharge électrique.

Pour implanter ce comportement deux classes ont été développées : la classe « Walker » qui simule le déplacement du robot et la classe « WayPoint » qui contrôle les points de passage. Parmi les attributs d'un « Waypoint » nous trouvons : un état (franchi/non franchi), une position sous forme de coordonnées cartésiennes et un rayon délimitant une zone de validation. La représentation des Waypoints dans le monde virtuel est une forme 3D simple avec deux significations (figure 2.4.9) :



Waypoint Vert : Point de contrôle	Waypoint Rouge : L'objectif du robot

Figure 2.4.9 : Modélisation des points de contrôle d'une trajectoire

Le déplacement du robot, défini dans la classe Walker, joue un rôle prépondérant dans la simulation, il coordonne l'ensemble des ordres à envoyer au dispositif réel et doit s'assurer du bon fonctionnement de celui-ci (éviter certains phénomènes comme le glissement...) . Le robot se dirige toujours vers le prochain point pour éviter tout écart de trajectoire (sauf en cas de collision) puis s'arrête à sa destination, s'oriente vers le prochain Waypoint et redémarre. Le déplacement du robot doit être guidé par un vecteur colinéaire à la trajectoire entre le robot et son prochain point de passage. Ce vecteur permet également de calculer l'orientation du robot pour rejoindre ce point. Entre deux points, si le robot rencontre un obstacle il s'arrête puis il se dirige perpendiculairement à sa trajectoire pour contourner l'objet (figure 2.4.10). Cette séquence peut être effectuée en plusieurs fois suivant le volume englobant de l'objet. Le robot change de

configuration et passe en mode collision. Une fois les détecteurs de collision éteins, le robot recherche sa trajectoire à partir de sa nouvelle position. Lors d'une collision avec un objet, le robot doit déterminer quelle direction prendre en fonction de sa position et de la position du point de contrôle à atteindre.

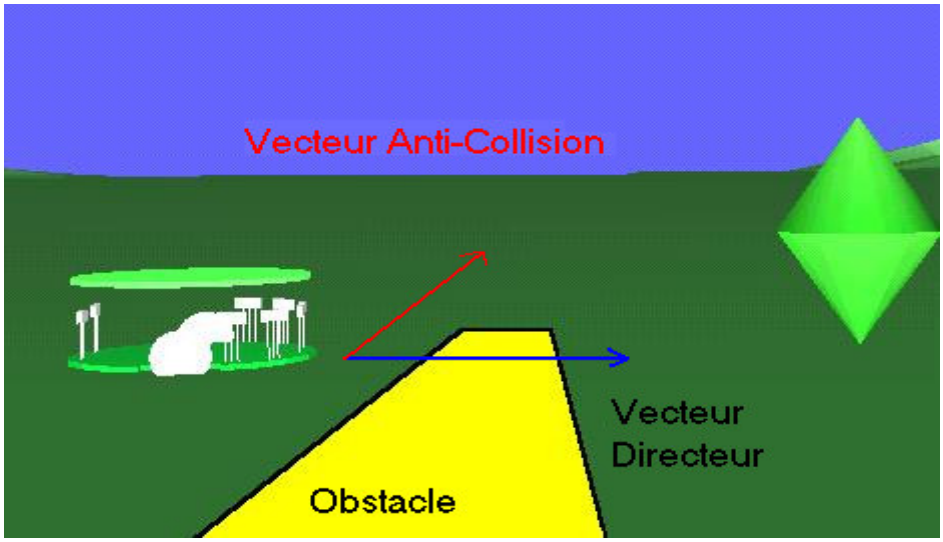


Figure 2.4.10 : Vecteurs de collision et direction

La dernière partie développée du comportement est l'action, elle est réalisée à la fin du trajet. Le robot effectue une spirale à partir du dernier point de contrôle (l'Objectif) jusqu'à déterminer la position d'un objet. L'action est un processus totalement indépendant utilisant des Waypoint. L'action place des points de contrôles sur un cercle avec une précision de n-divisions et à chaque fin de boucle de recherche sans résultats, le rayon du cercle est agrandi. Si un objet est découvert, la position du robot est renvoyée. Quelques problèmes concernant les collisions ont été identifiés. En effet, la gestion de collisions de Java3D signale les collisions mais n'offre pas d'autre information. Nous avons donc développé un algorithme simple, qui utilise des sphères englobantes, pour gérer les collisions.

Les méthodes de contrôle utilisées dans l'élaboration des classes de comportement permettent de simuler plusieurs comportements de la robotique. L'emploi des points de contrôle permet d'étendre les fonctionnalités du robot. Par exemple, le point de contrôle peut être mobile ou déporté sur un autre robot en mode coopératif, il suffit alors d'étendre les méthodes des classes de comportement. Par exemple, un bras articulé à besoin de ramasser un cube posé sur le sol, un robot assistant va se diriger vers lui pour découvrir le cube et communiquera les informations sur les dimensions et la position de ce cube. L'implantation de ce type de comportement est discutée dans le paragraphe suivant.

Le système ROVE : Robotique coopérative

Les robots d'assistance complètent les facultés humaines et permettent au système de profiter des capacités de la machine pour réaliser des travaux répétitifs ou difficiles au niveau physique, et de l'habilité de l'expert humain pour regarder, sentir et réagir au moment précis. L'assistant peut être réel comme les robots utilisés pour la téléchirurgie, la manipulation coopérative d'objets ou dans des missions comme l'exploration spatiale ou le déplacement dans des environnements inaccessibles (notamment comme aide pour les personnes handicapées). Dans la téléchirurgie, le but est de combiner la haute technologie avec l'expérience chirurgicale pour obtenir des chirurgies moins traumatisantes et qui demandent moins de ressources [Ottensmeyer96]. Dans la manipulation coopérative, on cherche une coordination efficace entre l'homme et la machine pour manipuler des objets. Par exemple, Arai et al [Arai00] ont développé un système dans lequel un robot aide un opérateur humain à transporter un objet long. Grâce à la combinaison de la perception et de l'interprétation de l'environnement de la part de deux entités, il est possible d'accomplir une tâche impossible à réaliser par une entité seule. Les assistants s'avèrent aussi utiles dans le contexte de la réadaptation. Dans ce cas, cette technologie est employée pour entraîner un utilisateur dans certaines tâches et pour la réalisation d'exercices de renforcement [Popescu02]. L'assistant peut aussi être virtuel comme dans le système Steve [Rickel99], un humain virtuel utilisé pour l'entraînement d'une équipe d'étudiants. Les humains virtuels prennent le rôle des instructeurs lorsque ceux-ci ne sont pas disponibles ou des élèves pour compléter une équipe, permettant ainsi aux étudiants de s'entraîner à tout moment (figure 2.4.11).

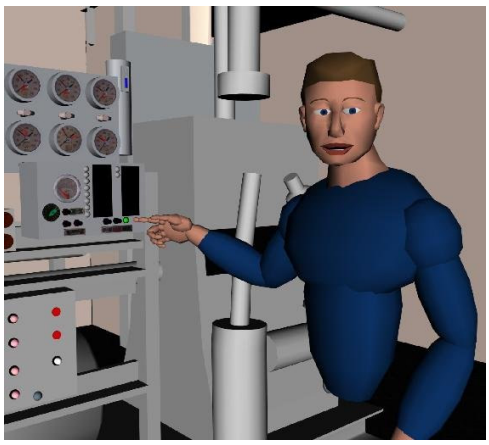


Figure 2.4.11 : L'agent Steve

Le projet ROVE a pour but la construction d'un système qui utilise la réalité virtuelle et les systèmes adaptatifs pour permettre la participation et l'interaction entre des opérateurs humains et des robots autonomes. Avec cette plate-forme, nous nous proposons d'étudier la coopération

homme-machine pour la réalisation de missions téléopérés. Le système ROVE est composé d'un système gérant l'environnement dynamique (ASSET) et d'un système de simulation comportementale appelé A^3 (Adaptative Autonomous Agents) qui contrôle les robots autonomes d'assistance.

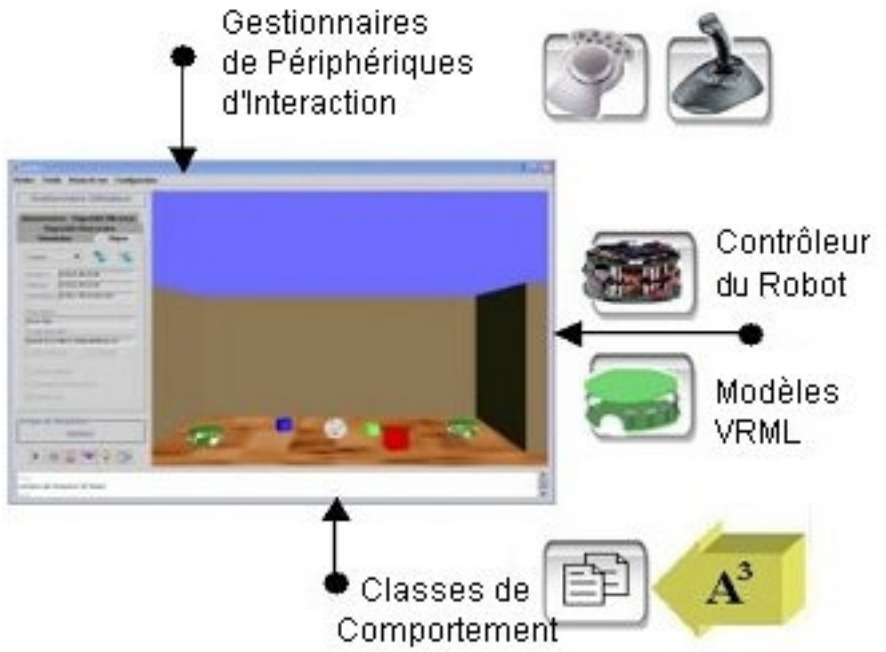


Figure 2.4.12 : Configuration du système ROVE

Le système A^3, développé par Olivier Heguy dans notre laboratoire [Heguy01], a comme but non seulement de fournir au robot un comportement autonome pour réagir dans un environnement dynamique, mais aussi de lui fournir les outils nécessaires afin de pouvoir aider l'utilisateur dans sa mission. Les modèles comportementaux utilisés en robotique sont sujets à certaines contraintes, à la différence de ceux utilisés dans les environnements virtuels, car les temps de réaction – parfois importants - peuvent entraîner des erreurs d'évaluation de l'espace dans lequel on évolue. De plus, l'espace est bruité, faussant les entrées du robot et pouvant causer des différences entre l'action souhaitée et l'action réalisée. Pour remédier à cela, le module comportemental est composé de différents comportements élémentaires qui permettent de relier les capteurs et les effecteurs en établissant une structure contrôlée a priori ou établie dynamiquement afin de pouvoir évoluer par apprentissage. L'approche suivie dans le système A^3 est celle d'un contrôleur hybride [Gat98] composé de trois couches : une couche de planification qui sert de centre de décision, une couche réactive qui représente un modèle élémentaire

d'actions pouvant être effectuées par l'entité (avance, tourne à droite, attrape…) et une couche intermédiaire qui compense les limitations des systèmes de planification et de réaction.

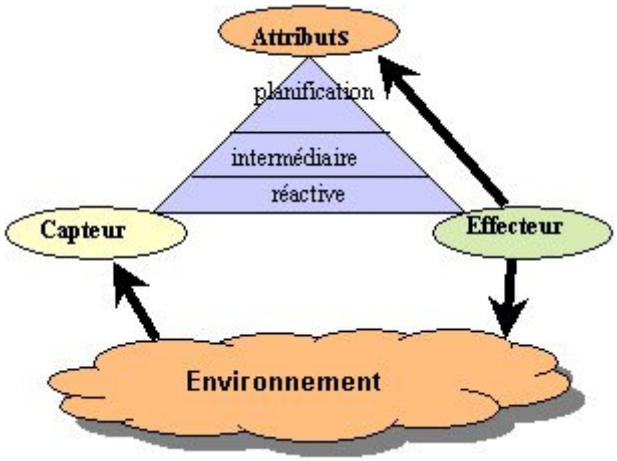


Figure 2.4.13 : Architecture de A³

La couche de planification permet au système de raisonner sur les actions qu'il effectue tandis que le contrôleur réactif assure la sécurité du robot dans l'environnement réel. La couche intermédiaire réalise le travail le plus complexe : elle doit compenser les limitations des autres couches, réconcilier leurs différentes échelles de temps et résoudre les conflits. Cette couche est composée d'une machine à états finis et d'une zone tampon de messages, de façon à ce que tous les changements "importants" découverts par le contrôleur de bas niveau soient communiqués au planificateur.

Le système A^3 fournit plusieurs comportements parmi lesquels la planification de trajectoires et le « push planning ». Le push planning est utilisé pour compenser les actions d'un utilisateur qui essaie de pousser un objet. Pour ce faire, le système utilise le centre géométrique de l'objet et la direction de la force exercée par l'opérateur. Pour calculer la direction de la force auxiliaire à appliquer, le planificateur utilise une variable α dont la valeur varie pendant l'exercice afin d'obtenir un angle approximatif. Puis, en comparant l'état initial et le nouvel état, et en utilisant l'apprentissage par renforcement, la valeur d'α est ajustée pour atteindre l'angle optimal.

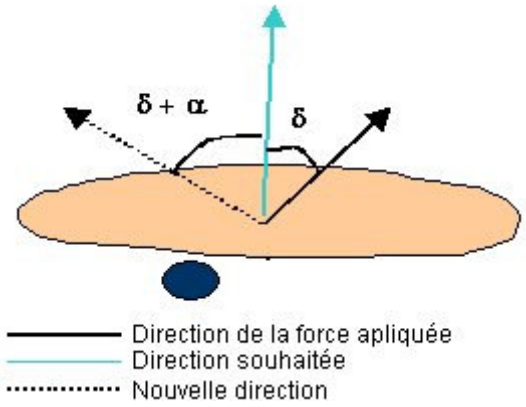


Figure 2.4.14 : Representation du push planning

Il existe deux types de planification de trajectoires dans le système A^3. La première utilise l'algorithme d'Astar pour trouver la trajectoire la plus courte dans un graphe. La deuxième utilise un algorithme génétique qui travaille avec un ensemble de solutions initialisé aléatoirement. Ensuite, grâce aux opérateurs génétiques tels que la sélection, la mutation et le croisement, l'algorithme converge vers une solution. Le planificateur est utilisé pour trouver les endroits que le robot peut atteindre ainsi que les chemins qu'il devra emprunter en tenant compte de son orientation et de son volume.

Les systèmes A^3 et ASSET forment la base du projet ROVE. La figure 2.4.15 montre un exemple d'application du système, avec un utilisateur collaborant avec un robot autonome. L'utilisateur génère les consignes pour le robot Khepera 1 en utilisant un dispositif d'interaction (spacemouse ou joystick) et l'interface graphique utilisateur. Ces consignes sont communiquées au simulateur local pour produire l'information de retour. Si l'état résultat de l'action est un état valide, les commandes sont envoyées au système réel via l'Administrateur. Le Gestionnaire Système Réel met à jour sa simulation permettant ainsi à l'unité comportementale A^3 de connaître l'état de chaque objet de simulation et de réagir de façon appropriée au nouvel environnement. L'unité A^3 filtre l'information pour obtenir les données nécessaires lui permettant de choisir un comportement et de diriger le robot autonome. Enfin, les deux robots exécutent leurs consignes. Le robot autonome est contrôlé uniquement par l'unité comportementale associée au simulateur du Gestionnaire Système Réel, ce qui permet au système A^3 de travailler sur des données globales déjà validées. L'unité dans le Gestionnaire Utilisateur permet de générer l'information de retour pour l'utilisateur. Pour cette application, les variables d'état sont les positions de deux robots et les collisions.

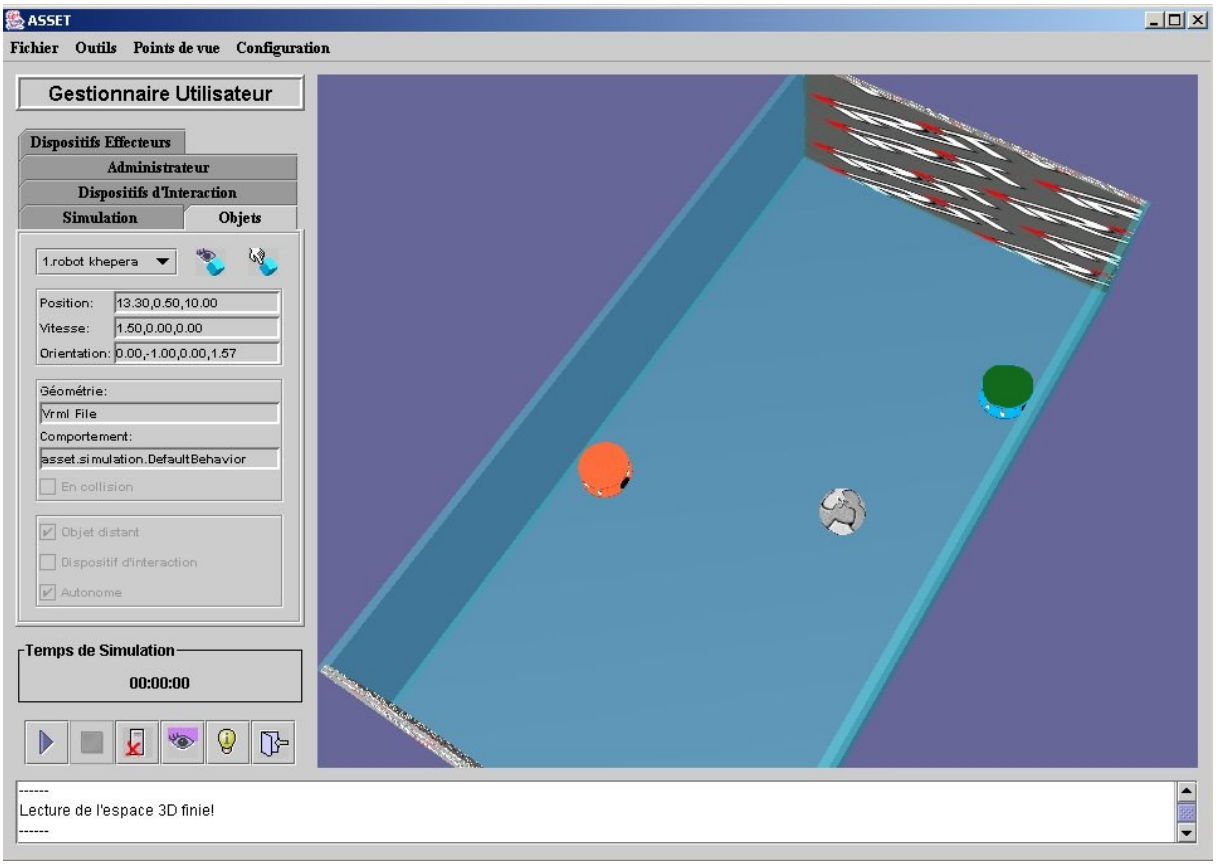


Figure 2.4.15 : Le système ROVE

Les classes constituant le système A^3 sont chargées et associées au robot autonome à travers le fichier de configuration du système ASSET. Les classes de contrôle des dispositifs d'interaction que nous avons développées font aussi partie du fichier de configuration afin que les changements dans la configuration matérielle ne requièrent aucune modification dans le code de l'application.

Conclusion

Dans ce chapitre nous avons présenté différentes applications dont la construction nous a permis de valider l'architecture de télérobotique proposée et les divers mécanismes offerts par ASSET. Ces expériences illustrent la facilité de mise en œuvre d'une application de télérobotique en utilisant notre système. En effet, l'évaluation des algorithmes de comportement et l'intégration de nouveaux dispositifs se trouve simplifiée par l'utilisation du fichier de configuration.

Nous avons aussi décrit le système ROVE, conçu pour l'étude de la téléopération coopérative. L'objectif de ce projet est d'offrir à l'utilisateur de l'aide via un robot autonome assistant. Le robot autonome est contrôlé par le système de simulation comportementale A^3. L'architecture proposée par le système ASSET a été développée de façon indépendante de celle du système de simulation comportementale, illustrant les capacités de notre système pour l'intégration de composants existants.

Ce chapitre clôt la description du système ASSET. Dans la dernière partie de ce document, nous discutons la manière dont notre système réponds aux exigences d'un environnement de développement pour la téléopération et les perspectives envisagées pour ce travail.

CONCLUSION

conclusions et perspectives

Le résultat de cette thèse est le système ASSET, une plate-forme de développement d'applications de télérobotique utilisant un environnement de réalité virtuelle distribuée. ASSET offre une base commune pour la conception d'applications sous la forme d'une architecture générale pour la téléopération. Elle offre aussi un ensemble de composants Java implémentant les services et collaborations définis dans l'architecture. Nous avons cherché dans notre travail à fournir un système simple à utiliser et très flexible, sans exigences coûteuses au niveau de la configuration logicielle ou matérielle. Pour cette raison nous avons choisi Java pour développer notre système de sorte qu'il puisse fonctionner sur plusieurs plate-formes sans changements majeurs. Notre système peut ainsi être considéré comme une solution de réalité virtuelle très accessible pour la recherche en télérobotique.

L'approche que nous avons suivie est la gestion abstraite des éléments principaux d'une application de téléopération (communications, état, gestion de dispositifs…). Avec cette approche les applications peuvent intégrer facilement leurs ressources (classes, modèles géométriques…) dans la plate-forme générique fournie par ASSET, et la construction de prototypes ou de programmes de tests devient nettement plus rapide.

Dans le chapitre 3 nous avons énuméré les différents aspects qui doivent être pris en compte par un environnement de développement pour la téléopération. Nous discutons maintenant, comment le système ASSET répond à ces exigences :

- **Générique**

Pour avoir un système générique facilement configurable, nous avons réalisé son implantation en suivant un *développement orienté-objet*. Ceci assure la modularité du système global, car les composants sont utilisés uniquement à travers leur interface. Ils peuvent donc être modifiés de façon indépendante les uns des autres. En outre, pour avoir un système flexible qui puisse s'adapter aux diverses applications, nous avons réalisé une *séparation entre le traitement et le transport des données*. En effet, les composants d'ASSET réalisent le transport et l'échange de données sans considérer leur signification. Le traitement, et donc la connaissance de la structure et de la sémantique des données, est réalisée par le développeur. Un fichier de configuration permet d'intégrer les classes implantant le traitement : classes de comportement, protocoles de communication, classes de contrôle de dispositifs. Ce fichier, qui décrit d'ailleurs les détails d'une application spécifique, est utilisé ensuite par ASSET pour le *chargement dynamique* des ressources fournies par le développeur.

En outre, avec l'utilisation des *dispositifs virtuels* que nous avons défini, la liaison directe entre l'application et les dispositifs est éliminée, permettant ainsi la construction d'applications indépendantes des périphériques utilisés.

- **Communications**

Notre système prend en charge les communications entre les différents sites utilisés par l'application. Ces communications sont réalisées par défaut en suivant le protocole TCP/IP. Cependant, ce protocole n'est pas imposé, il peut être changé par l'utilisateur dans les fichiers de configuration des sites (et en fournissant logiquement les classes implantant le nouveau protocole). Pour gérer le délai de communication, notre système utilise une approche classique : une simulation prédictive génère l'information de retour pour l'utilisateur et valide les consignes avant leur envoi au système distant. Mais nous avons aussi intégré une *simulation dans le système distant* qui nous permet de définir, pour les applications de téléopération, un mécanisme de synchronisation utilisé dans le domaine de la simulation distribuée. Ce mécanisme, basé sur la différence d'état entre le système réel et le système simulé, permet de réduire le nombre de synchronisations entre le système local et le système distant et donc le trafic du réseau.

- **Contrôle de dispositifs**

Le système ASSET comprend un environnement virtuel habité par des entités téléopérées ou autonomes. Pour ce faire, nous permettons la *définition du comportement de chaque entité* de la

simulation et de la classe de contrôle de chaque dispositif. Comme pour les autres aspects concernant la configuration d'une application particulière, les classes de comportement des entités doivent être spécifiées dans le fichier de configuration du système ASSET. Grâce à cette approche, l'intégration et l'évaluation des algorithmes de contrôle d'une entité ou d'un dispositif est facilitée.

- **Interface utilisateur**

Dans ASSET, l'affichage graphique de l'environnement virtuel se réalise sur un écran conventionnel d'ordinateur. Nous considérons que *l'immersion partielle* est la plus adaptée à un système de développement car l'utilisateur ne perd pas le contact avec le monde réel et peut s'éloigner facilement du système. L'environnement virtuel est complété par une interface 2D classique permettant à l'utilisateur de connaître l'état des objets de la simulation, des dispositifs et du système distant.

Grâce à ces caractéristiques, le système ASSET permet aux utilisateurs de diminuer le temps de construction et d'évaluation de leurs prototypes. Par exemple, pour tester un comportement, le développeur doit simplement fournir le modèle géométrique et la classe de contrôle de son entité, ASSET fournit tous les autres composants nécessaires à l'exécution de l'application. Les utilisateurs peuvent donc tester et améliorer leur travail sans avoir besoin de connaître le système sous-jacent qui contrôle l'exécution.

Nous avons utilisé le système ASSET pour programmer une application de téléopération coopérative appelée ROVE. Pour ce faire, nous avons intégré un système de simulation comportementale, A^3, responsable du contrôle des robots autonomes. Nous avons montré avec cette application comment utiliser les ressources disponibles dans ASSET pour développer un projet complexe.

Le travail de conception et de développement du système ASSET nous a permis d'identifier plusieurs améliorations. Tout d'abord, l'intégration d'un moteur de détection de collisions plus puissant s'avère nécessaire pour la construction d'applications avec des besoins de précision élevés. D'un autre côté, les mécanismes nécessaires à l'interaction efficace entre plusieurs collaborateurs humains doivent être étudiés et implantés dans l'Administrateur.

Parmi les perspectives envisagées pour notre travail, nous considérons que l'intégration d'outils communs dans le domaine de la téléopération est la plus intéressante. Notre système fournit les services de base indispensables à toute application de télérobotique. Il est donc possible d'envisager l'intégration de services plus sophistiqués mais très utiles comme la programmation hors-ligne, la planification de trajectoires, la localisation, et la modification des données de simulation suivant les « découvertes » ou les différences perçues par le système robotique. Enfin, étant donné que notre système est un outil de développement, de nombreuses applications intégrant des algorithmes de comportement, de nouveaux dispositifs d'interaction (gant de données ou périphérique à retour d'effort de type Phantom) ou d'action (robot Pekee) peuvent être envisagées.

BIBLIOGRAPHIE

bibliographie

Références

[Aditya00] A. Aditya, B. Riyanto, *Implementation of Java 3D Simulation for Internet Telerobotic System*, IASTED International Conference Modelling and Simulation (MS'2000), USA, Mai 2000.

[Aleotti02] J. Aleotti, S. Bottazzi, S. Caselli, M. Reggiani, *A Multimodal User Interface for Remote Object Exploration in Teleoperation Systems*, IARP International Workshop on Human Robot Interfaces: Technologies & Applications, Italie, Novembre 2002.

[Alvarez00] B. Álvarez, A. Iborra, A. Alonso, J.A. de la Puente, J.A. Pastor, *Generic Software Architecture for Teleoperation Systems: Application to Service Robots in Nuclear Power Plants*, Nuclear Engineering International, Vol. 45, No. 548, pp 24-28, Mars 2000.

[Alvarez01a] B. Álvarez, A. Iborra, A. Alonso, J.A. de la Puente, *Reference Architecture for Robot Teleoperation: Development Details and Practical Use*, Control Engineering Practice, Vol. 9, No. 4, pp395-402, 2001.

[Alvarez01b] B. Álvarez, A. Iborra, J.A. Pastor, C. Fernández, A. Alonso, J.A. de la Puente, *Software Architecture for Development of Mechatronic Systems: Service Robots*, Dedicated Systems Magazine, No. 4, pp17-22, 2001.

[Anderson93] R.J. Anderson, *SMART : a Modular Architecture for Robotics and Teleoperation*, IEEE International Conference on Robotics and Automation, USA, Mai 1993.

[Anderson95] R.J. Anderson, *A Generic Simulation System for Intelligent Agent Designs*, Applied Artificial Intelligence No. 5, pp.527-562, 1995.

[Arai00] H. Arai, T. Takubo, Y. Hayashibara, K. Tanie, *Human-Robot Cooperative Manipulation Using a Virtual Nonholonomic Constraint*, IEEE International Conference on Robotics and Automation, USA, Avril 2000.

[Backes93] P. Backes, M. Long, R. Steele, *The Modular Telerobot Task Execution System for Space Telerobotics,* IEEE International Conference on Robotics and Automation, USA, Mai 1993.

[Backes98] P.G. Backes, K.S. Tao, G.K. Tharp, *Mars Pathfinder Mission Internet-based Operation using WITS*, IEEE International Conference on Robotics and Automation, Belgique, Mai 1998.

[Balaram92] J. Balaram, H. Stone, *Automated Assembly in the JPL Telerobot Testbed*, Intelligent Robotic Systems for Space Exploration, Kluwer Academic Publishers, 1992.

[Balcisoy00] S. Balcisoy, M. Kallmann, P. Fua, D. Thalmann, *A Framework for Rapid Evaluation of Prototypes with Augmented Reality*, ACM Symposium on Virtual Reality Software and Technology (VRST 2000), Korea, Octobre 2000.

[Bares97] J. Bares, D. Wettergreen, *Lessons from the Development and Deployment of Dante II*, 1997 Field and Service Robotics Conference, Australia, December 1997.

[Bejczy91] A. Bejczy, Z. Szakaly, *An 8-D.O.F. Dual-Arm System for Advanced Teleoperation Performance Experiments*, Fifth Annual Workshop on Space Operations Applications and Research (SOAR '91), USA, Juillet 1991.

[Benokratis98] A. Benokratis, *What is a CAVE?,* Document accessible via l'URL http://www.sv.vt.edu/future/vt-cave/whatis/, Mai 1998.

[Blackmon96] T. Blackmon, L.Stark, *Model-Based Supervisory Control in Telerobotics*, Presence Vol. 5, No. 2, pp205-223, 1996.

[Blank99] D. Blank, J.H. Hudson, B.C. Mashburn, E.A. Roberts, *The XRCL Project: The University of Arkansas' Entry into the AAAI 1999 Mobile Robot Competition*, Technical Report CSCE-1999-0, 1999.

[Borrego96] J. Borrego, F. Free, *Porting High Quality Graphics Simulations to a low-cost computer architecture*, thèse Master of Science in Computer Science, Naval Postgraduate School, 1996.

[Boucard01] A. Boucard, *Une Application de Téléopération Construite sur ASSET*, Rapport de stage, IRIT, 2002.

[Brady98] K. Brady, T.J. Tarn, *Internet-based Remote Teleoperation*, IEEE International Conference on Robotics and Automation, Belgique, Mai 1998.

[Burdea96] G. Burdea, *Force & Touch Feedback for Virtual Reality*, John Wiley & Sons, 1996.

[Burdea99] G. Burdea, *Invited Review: The Synergy Between Virtual Reality and Robotics*, IEEE Transactions on Robotics and Automation, Vol. 15, No 3, Juin 1999.

[Burgard98] W. Burgard, A.B. Cremers, D. Fox, G. Lakemeyer, D. Hâhnel, D. Schulz, W. Steiner, S. Thrun, *The interactive museum tour-guide robot*, 15th National Conference on Artificial Intelligence, USA, Juillet 1998.

[Cannon97] D. Cannon, G. Thomas, *Virtual Tools for Supervisory and Collaborative Control of Robots,* Presence Vol. 6, No. 1, pp1-28, 1997.

[Capin99] T.K. Capin, D. Thalmann, *A Taxonomy of Networked Virtual Environments*, International Workshop on Synthetic – Natural Hybrid Coding And Three Dimensional Imaging (IWSNHC3DI'99), Grèce, Septembre 1999.

[Clark98] J. Clark, *May the Force Feedback Be with You : Grappling with DirectX and DirectInput*, Microsoft System Journal, Document accessible via l'URL http://www.microsoft.com/msj/0298/force.htm, Février 1998.

[Cox94] M. J. Cox, J.E.F. Baruch, *Robotic Telescopes: An Interactive Exhibit on the World-Wide Web*, 2nd International Conference of the World-Wide Web, USA, Octobre 1994.

[Dalton98] B. Dalton, K. Taylor, *A Framework for Internet Robotics*, IEEE International Conference On Intelligent Robots and Systems (IROS) : Workshop on Web Robots, Canada, Octobre 1998.

[Debus] T. Debus, J. Stoll, R.D. Howe, P. Dupont, *Cooperative Human and Machine Perception in Teleoperated Assembly*, Experimental Robotics VII. The Fifth International Symposium, USA, Décembre 2000.

[Drascic93] D. Drascic, J. Grodski, P. Milgram, S. Zhai, *Argos: a Display System for Augmenting Reality*, ACM Conference on Human Factors in Computing Systems, Pays-Bas, Avril 1993.

[Dvorak00] D. Dvorak, Rasmussen R., G. Reeves, A. Sacks, *Software Architecture Themes in JPL's Mission Data System,* IEEE Aerospace Conference, USA, Mars 2000.

[Espiau01] B. Espiau, *La Robotique, Histoire et Perspectives*, La Science au Présent, Encyclopædia Universalis Editions, 2001.

[Ferrel65] W.R. Ferrel, *Remote Manipulation with Transmission Delay*, IEEE Transactions on Human Factors in Electronics HFE-6, No. 1, 1965.

[Ferworn99] A. Ferworn, R. Roque, I. Vecchia, *MAX: Wireless Teleoperation via the World Wide Web*, IASTED Conference on Robotics and Applications, USA, Octobre 1999.

[Finke99] M. Finke, J. Strassner, J. Speier, L. Peters, M. Pauly, M. Göbel, H. Surmann, *An Interactive Test Environment for Autonomous Robots*, Topical Workshop on Virtual Reality and Advanced Human-Robot Systems, 15th World Congress of the International Measurement Confederation, Japon, Juin 1999.

[Flückiger98] L. Flückiger, *Interface pour le Pilotage et l'analyse des Robots basée sur un Générateur de Cinématiques*, Thèse de doctorat, Ecole Polytechnique Fédérale de Lausanne, 1998.

[Fong93] T. Fong, *A Computational Architecture for Semi-Autonomous Robotic Vehicles*, Computing in Aerospace 9 Conference, USA, Octobre 1993.

[Fong99] T. Fong, C. Thorpe, C. Baur, *Collaborative Control: A Robot-Centric Model for Vehicle Teleoperation*, Agents with Adjustable Autonomy, Papers from the AAAI 1999 Spring Symposium, Technical Report SS-99-06, The AAAI Press, 1999.

[Fong00] T. Fong, C. Thorpe, C. Baur, *Advanced Interfaces for Vehicle Teleoperation: Collaborative Control, Sensor Fusion Displays, and Web-based Tools,* Vehicle

Teleoperation Interfaces Workshop, IEEE International Conference on Robotics and Automation, USA, Avril 2000.

[Fong01] T. Fong, C. Thorpe, *Vehicle Teleoperation Interfaces*. Autonomous Robots No. 11, pp9–18, 2001

[Gat98] E. Gat, *On Three-Layer Architectures*, Artificial Intelligence and Mobile Robots, AAAI Press, 1998.

[Ghiasi99] S. Ghiasi, M. Seidl, B. Zorn, *A Generic Web-based Teleoperations Architecture: Details and Experience*, SPIE Telemanipulator and Telepresence Technologies VII, USA, Septembre 1999.

[Gill02] C.D. Gill, W.D. Smart, Middleware for robots ? , Proceedings of the AAAI Spring Symposium on Intelligent Distributed and Embedded Systems, USA, Mars 2002.

[Goertz52] R. C. Goertz, *Fundamentals of General-Purpose Remote Manipulators*, Nucleonics, pp36-45, Novembre 1952.

[Goertz54] R. C. Goertz, *Electronically Controlled Manipulator*, Nucleonics, pp46-47, Novembre 1954.

[Goldberg95a] K. Goldberg, M. Mascha, S. Gentner, N. Rothenberg, C. Sutter, J. Wiegley, *Desktop tele-operation via the World Wide Web*, IEEE International Conference on Robotics and Automation, Japon, Mai 1995.

[Goldberg95b] K. Goldberg, M. Masha, S. Gentner, N. Rothenberg, C. Sutter, J. Wiegley, *Beyond the Web : Manipulating the Physical World via the WWW*, Computer Networks and ISDN Systems Journal, Vol. 28, No. 1, Décembre 1995.

[Goldberg98] S. Goldberg, G.A. Bekey, Y. Akatsuka, *DIGIMUSE: An Interactive Telerobotic System for Remote Viewing of 3D Art Objects*, IROS'98: Workshop on Web Robots, Canada, Octobre 1998.

[Goldberg00] K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, G. Smith, *Collaborative Teleoperation via the Internet*, IEEE International Conference on Robotics and Automation (ICRA), USA, Avril 2000.

[Gosling96] J. Gosling, H. McGilton, *The Java Language Environment, A white paper*, Document accessible via l'URL http://java.sun.com/docs/white/langenv, Mai 1996.

[Gradecki94] J. Gradecki, *The Virtual Reality Programmer's Kit*, John Wiley & Sons, 1994.

[Grange00] S. Grange, T. Fong, C. Baur, *Effective Vehicle Teleoperation on the World Wide Web*, IEEE International Conference on Robotics and Automation (ICRA 2000), USA, Avril 2000.

[Heguy01] O. Heguy, N. Rodriguez, J-P. Jessel, Y. Duthen, H. Luga, *Virtual Environment for Cooperative Assistance in Teleoperation*, International

Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG'2001), Vol. 3, ppII9-II12, République Tchèque, Février 2001.

[Hine94] B. Hine, C. Stoker, M. Sims, et. Al, *The Application of Telepresence and Virtual Reality to Subsea Exploration*, 2nd Workshop on Mobile Robots for Subsea Environments (ROV'94), USA, Mai 1994.

[Hine95] B. Hine, *VEVI : A Virtual Environment Teleoperation Interface for Planetary Exploration*, 25th International Conference on Environmental Systems, USA, Juillet 1995.

[Hirukawa97] H. Hirukawa, T. Matsui, H. Onda, *Prototypes of Teleoperation Systems via a Standard Protocol with a Standard Human Interface*, IEEE International Conference on Robotics and Automation, USA, Avril 1997.

[Hu01] H. Hu, L. Yu, P. Wo Tsui, Q. Zhou, *Internet-based Robotic Systems for Teleoperation*, International Journal of Assembly Automation, Vol. 21, No. 2, pp143-151, Mai 2001.

[Inostroza] P. Inostroza, *Technologie de connexion et de composition pour environnements virtuels*, Thèse de Doctorat, Université Joseph Fourier, 2002.

[Jia01] S. Jia, K. Takase, *An Internet Robotic System Based on the Common Object Request Broker Architecture*, 2001 IEEE International Conference on Robotics & Automation, Korea, May 2001.

[Jung99] B. Jung, J.T. Milde, *An Open Virtual Environment for Autonomous Agents Using VRML and Java*, 4th Symposium on Virtual Reality Modeling Language, Allemagne, Février 1999.

[Kaber99] D. Kaber, R. Zhou, D. Song, *Design and Prototyping of an Economical Teleoperations Testbed for Human factors Research: cost, resource requirements and capability assessment*, 25th International Conference on Computers & Industrial Engineering, USA, Mars 1999.

[Kheddar97] A. Kheddar, C. Tzafestas, P. Coiffet, *The Hidden Robot Concept – High Level Abstraction Teleoperation*, IEEE/RSJ International Conference on Intelligent Robotics Systems (IROS'97), Vol. 3, pp1818-1824, France, Septembre 1997.

[Kheddar98] A. Kheddar, J.G. Fontaine, P. Coiffet, *Mobile Robot Teleoperation in Virtual Reality*, IEEE/SMC IMACS Multiconference on Computational Engineering in Systems Applications (CESA'98), Tunisie, Avril 1998.

[Komerska02] R. Komerska, C. Ware, M. Plumlee, *Haptic Interface for Center-of-Workspace Interaction*, 2002 IEEE Virtual Reality – Haptics Symposium, USA, Mars 2002.

[Kondraske93] G.V. Kondraske, R.A. Volz, D.H. Johnson, D. Tesar, J.C. Trinkle, *Network-based Infrastructure for Distributed Remote Operations and Robotics*

Research, IEEE Transations on Robotics and Automation, Vol. 9, No. 5, pp702-704, Octobre 1993.

[Kosuge98] K. Kosuge, J. Kikuchi and K. Takeo, *VISIT: A Teleoperation system via computer Network*, Proceedings IROS'98: Workshop on Web Robots, Canada, Octobre 1998.

[Kress00] R.L. Kress, *Teleoperation and Telerobotics*, Robotics and Automation Society, Document accessible via l'URL http://www.engr.utk.edu/maes/ff/rlk/ieee/, Juillet 2000.

[Lefebvre92] D.R. Lefebvre, G.N. Saridis, *Integrating Robotic Functions and Operator Supervision using Petri Nets*, IEEE/RSJ International Conference on Intelligent Robots and Systems, USA, Juillet 1992.

[Lelevé98] A. Lelevé, P. Fraisse, A. Crosnier, P. Dauchez, F. Pierrot, *Towards Virtual Control of Mobile Manipulators*, 3rd World Automation Congress (WAC'98), USA, Mai 1998.

[Lelevé99] A. Lelevé, *Modelisation et Simulation d'une Téléopération a Longue Distance via Internet*, 10èmes Journées Jeunes Chercheurs en Robotique (JJCR'10), France, Novembre 1999.

[Lelevé01] A. Lelevé, P. Fraise, P. Dauchez, *Telerobotics over IP Networks: Towards a Low-level Real-time Architecture*, IROS - International Conference on Intelligent Robots and Systems, USA, Octobre 2001.

[Lin95] I. Lin, F. Wallner, R. Dillmann, *An Advanced Telerobotic Control System for a Mobile Robot with Multisensor Feedback*, 4th International Conference on Intelligent Autonomous Systems, Allemagne, Mars 1995.

[Maneewarn99] T. Maneewarn, *Haptic Feedback of Manipulator Kinematics Conditioning for Teleoperation*, Thèse de Doctorat, Université de Washington, 1999.

[Meehan99] M. Meehan, *Survey of Multi-user Distributed Virtual Environments*, In course notes: Developing Shared VDs, SIGGRAPH'99, USA, Août 1999.

[Michel97] O. Michel, P. Saucy, F. Mondada, *KhepOnTheWeb: An Experimental Demonstrator in Telerobotics and Virtual Reality*, ProceedingsVSMM'97, Suisse, Septembre 1997.

[Milgram95] P. Milgram, D. Drascic, J. Grodski, A. Restogi, S. Zhai, C. Zhou, *Merging Real & Virtual Worlds*, IMAGINA'95, Monaco, Fevrier 1995.

[Miller91] D. Miller, C. Lennox, *An object-oriented environment for robot systems*, IEEE Transactions on Control Systems, Vol. 11, No. 2, pp14-23, Février 1991.

[Mirecourt99] A. Mirecourt, P-Y. Saumon, *Le dévéloppeur Java2*, Osman Eyrolles Multimedia, 1999.

[Mosher60] R.S. Mosher, B. Wendel, *Force reflecting electro-hydraulic servo-manipulator*, Electro-Technology, pp138, 1960.

[Murphy96] R. Murphy, E. Rogers, *Cooperative Assistance for Remote Robot Supervision*, Presence, Vol. 5, No. 2, pp224-240, 1996.

[Muscettola98] N. Muscettola, *Remote Agent: To Boldly Go Where No AI System Has Gone Before*. Artificial Intelligence, No. 103, pp5–48, 1998.

[Nehmzow96] U. Nehmzow, A. Buhlmeier, H. Durer, M. Nolte, *Remote control of mobile robot via Internet*, Department of Computer Science, University of Manchester, Technical Report Series, UMCS-96-2-3, 1996.

[Nguyen00] L. Nguyen, M. Bualat, L. Edwards, L. Flueckiger, C. Neveu, K. Schwehr, M.D. Wagner, E. Zbinden, *Virtual Reality Interfaces for Visualization and Control of Remote Vehicles*, Vehicle Teleoperation Interfaces Workshop,.IEEE International Conference on Robotics and Automation, USA, Avril 2000.

[Ottensmeyer96] M. Ottensmeyer, J. Thompson, T. Sheridan, *Cooperative Telesurgery: Effects of Time Delay on tool Assignment Decision,* Human Factors and Ergonomics society 40th Annual Meeting, USA, Septembre 1996.

[Paolini97] C. Paolini, M. Vuskovic, *Integration of a Robotics Laboratory Using CORBA*, IEEE International Conference on Systems, Men and Cybernetics, USA, Octobre 1997.

[Park01] S. Park, R.D. Howe, D.F. Torchiana, *Virtual Fixtures for Robotic Cardiac Surgery*, 4th International Conference in Medical Image Computing and Computer-Assisted Intervention (ICCAI), Pays-Bas, Octobre 2001.

[Piguet95] L. Piguet, T. Fong, B. Hine, P. Hontalas, E. Nygren, *VEVI: a Virtual Reality Tool for Robotic Planetary Exploration*, Virtual Reality World Conference, Allemagne, Février 1995.

[Piguet96] L. Piguet, *The Virtual Environment Vehicle Interface : a Dynamic Distributed and Flexible Virtual Environment*, Imagina'96, Monaco, Février 1996.

[Pook95] P. Pook, D. Ballard, *Remote Teleassistance*, IEEE International Conference on Robotics and Automation, Japon, Mai 1995.

[Popescu02] G.V. Popescu, G. Burdea, R. Boian, *Shared Virtual Environments for Telerehabilitation*, Medicine Meets Virtual Reality 2002 Conference, USA, Janvier 2002.

[Raimondi88] T. Raimondi, *Lectures*, Advances in Teleoperation for International Center for Mechanical Sciences, Italie, Mai 1988.

[Rastogi95] A.Rastogi, P.Milgram, *Augmented Telerobotic Control: A visual interface for unstructured environments,* KBS/Robotics Conference, Canada, Octobre 1995.

[Robb95] R.A. Robb, B. Cameron, *Virtual Reality Assisted Surgery Program*, Biomedical Imaging Resource, Mayo Foundation, Rochester, Minnesota. Document accessible via l'URL http://www.mayo.edu/bir/HTMLs/robb_MMVR3_95.htm

[Rosenberg93] L.B. Rosenberg, *Virtual Fixtures : Perceptual Tools for Telerobotic Manipulation*, IEEE Virtual Reality Annual International Symposium (VRAIS'93), USA, Septembre 1993.

[Rickel99] J. Rickel, W. Johnson, *Virtual Humans for Team Training in Virtual Reality*, 9th World Conference on Artificial Intelligence in Education, France, Juillet 1999.

[Rodriguez00a] A.N. Rodriguez, J-P. Jessel, *ASSET: Virtual Environments in Teleoperation Systems*, First French-British International Workshop on Virtual Reality, France, Juillet 2000.

[Rodriguez00b] A.N. Rodriguez, O. Heguy, J-P. Jessel, H. Luga, *Assistance Coopérative pour la Téléopération*, 13 Journées Jeunes Chercheurs en Robotique, France, Septembre 2000.

[Rodriguez01a] A.N. Rodriguez, J-P. Jessel, P. Torguet, *ASSET: A Testbed for Teleoperation Systems*, WSES/IEEE International Conference on Simulation, Malte, Septembre 2001.

[Rodriguez01b] A.N. Rodriguez, J-P. Jessel, P. Torguet, *Virtual Reality in Cooperative Teleoperation*, 1st Ibero-American Symposium on Computer Graphics (SIACG 2002), Portugal, Juillet 2002.

[Rodriguez01c] A.N. Rodriguez, J-P. Jessel, P. Torguet, *A Tool for Cooperative Teleoperation Research*, 33th International Symposium on Robotics (ISR 2002), Suede, Octobre 2002.

[Rodriguez02] A.N. Rodriguez, J-P. Jessel, P. Torguet, *A Virtual Reality Tool for Teleoperation Research*, Virtual Reality Journal, Springer-Verlag London Ltd, A paraître courant 2002.

[Rogers95] E. Rogers, R.R. Murphy, A. Stewart, N. Warsi, *Cooperative Assistance for Remote Robot Supervision*, IEEE International Conference on Systems, Man and Cybernetics, Canada, Octobre 1995.

[Sahin99] E. Sahin, P. Gaudiano, *KITE : The Khepera Integrated Testing Environment*, First International Khepera Workshop, Allemagne, Décembre 1999.

[Salmi00] S. Salmi, *Potential Use of XML in Robotics*, Document accessible via l'URL : http://www.automationit.hut.fi/julkaisut/documents/seminars/sem_a00/seminar.htm, 2000

[Sanza99] C. Sanza, C. Destruel, Y. Duthen, *Autonomous actors in an interactive real-time environment*, ICVC'99, International Conference on Visual Computing, Inde, Février 1999.

[Schulz00] D. Schulz, W. Burgard, D. Fox, S. Thrun, A.B. Cremers, *Web Interfaces for Mobile Robots in Public Places*, IEEE Robotics & Automation Magazine, Vol. 7, No. 1, pp48 – 56, Mars 2000.

[Shaw93] C. Shaw, M. Green, J. Liang, Y. Sun, *Decoupled Simulation in Virtual Reality with the MR Toolkit*, Information Systems, Vol. 11, No. 3, 1993.

[Sheridan92] T. Sheridan, *Telerobotics, Automation and Human Supervisory Control*, The MIT Press, 1992.

[Shneiderman92] B. Shneiderman, *Designing the User Interface : Strategies for Effective Human-Computer Interaction*, Addison-Wesley, 1992.

[Siegwart99] R. Siegwart, P. Saucy, *Interacting Mobile Robots on the Web*, International Conference on Robotics and Automation (ICRA'99), USA, Mai 1999.

[Simmons98] R. Simmons, *Xavier: An Autonomous Mobile Robot on the Web*, IROS'98: Workshop on Web Robots, Canada, Octobre 1998.

[Smith99] N. Smith, C. Egert, E. Cuddihy, D. Walters, *Implementing Virtual Robots in Java3D using a Subsumption Architecture*, Association for the Advancement of Computing in Education, USA, Octobre 1999.

[Stafford-Fraser95] Q. Stafford-Fraser, *The Trojan Room Coffee Pot: a (non-technical) Biography*, Document accessible via l'URL http://www.cl.cam.ac.uk/coffee/qsf/coffee.html, Mai 1995.

[Stanney98] K. M. Stanney, R. R. Mourant, R. S. Kennedy, *Human Factors Issues in Virtual Environments: A Review of the Literature*, Presence, Vol. 7, No. 4, pp327-351, 1998.

[Stone96] H. W. Stone, *Mars Pathfinder Microrover A Low-Cost, Low-Power Spacecraft*, AIAA Forum on Advanced Developments in Space Robotics, USA, Août 1996.

[Surmann99] H. Surmann, M. Theiinger, *Robodis: A dispatching system for multiple autonomous service robots*, FSR'99, Robotics Applications for the Next Millenium, USA, Août 1999.

[Taylor95] K. Taylor, J. Trevelyan, *Australia's telerobot on the web*, 26th International Symposium on Industrial Robotics, Singapour, Octobre 1995.

[Taylor97] K. Taylor, B. Dalton, *Issues in Internet Telerobotics*, FSR'97 International Conference on Field and Service Robotics, Australia, Décembre 1997.

[Terzopulos94] D. Terzopulos, T. Xiaoyuan, R. Grzeszczuk, *Artificial Fishes with Autonomous Locomotion, Perception, and Learning in a Simulated Physical World*, Artificial Life IV, pp17-27, 1994.

[Volpe94] R. Volpe, J. Balaram, *Technology for Robotic Surface Inspection in Space*, AIAA Conference on Intelligent Robots in Field, Factory, Service, and Space (CIRFFSS), USA, Mars 1994.

[Volpe00] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, *CLARAty: Coupled Layer Architecture for Robotic Autonomy*, JPL Technical Report D-19975, Décembre 2000.

[Volpe01] R. Volpe, I. Nesnas, T. Estlin, D. Mutz, R. Petras, H. Das, *The CLARAty Architecture for Robotic Autonomy*, 2001 IEEE Aerospace Conference, USA, Mars 2001.

[Ware94] C. Ware, R. Balakrishnan, *Reaching for Objects in VR Displays: Lag and Frame Rate*, ACM Transactions on Computer-Human Interaction, Vol. 1, No. 4, pp331-356, 1994.

[Weast99] Weast, Kitts, Ota, Bulich, Laurence, Lwin, Wigle, *Integrating Digital Stereo Cameras with Mars Pathfinder Technology for 3D Regional Mapping Underwater*, IEEE Aerospace Conference, USA, Mars 1999.

[Werry00] I. Werry, K. Dautenhahn, W. Harwin, *Challenges in Rehabilitation Robotics: A Mobile Robot as a Teaching Tool for Children with Autism*, International Workshop on Recent Advances in Mobile Robots, UK, Juin 2000.

[Wettergreen95] D. Wettergreen, H. Pangels, J. Bares, *Behavior-based Gait Execution for the Dante II Walking Robot*, IEEE International Conference on Intelligent Robots and Systems (IROS), USA, Août 1995.

URLs

[EAI] *External Authoring Interface*, http:// ww.web3d.org/WorkingGroups/vrml-eai

[IFC02] Immersion Corporation, *Immersion Tools and Libraries, Overview and Documentation, Version 2.3 Reference Manual,* http:// www.immersion .com/developer/downloads/IFC/HTML/ifchlp.htm, Mars 2002.

[DirectInput] MSDN Library, *Direct Input Documentation,* http://msdn.microsoft.com/library/enus/intinput/hh/intinput/di_0pgp.asp

[Java] http://java.sun.com/j2se/

[Java3D] http://java.sun.com/products/java-media/3D/

[Java3DTutorial01] Sun Microsystems, *Getting Started with the Java3D API*, http://java.sun.com/products/java-media/3D/collateral/

[JavaComm] *Java Communications API*, http://java.sun.com/products/javacomm/

[JavaComm98] Sun Microsystems, *Java Communications API User Guide*, http://java.sun.com/products/javacomm/javadocs/API_users_guide.html, 1998

[JavaDescription01] Software Engineering Institute, Carnegie Mellon University, *Java, Software Technology Review*, http://www.sei.cmu.edu/str/descriptions/java.html, Décembre 2001.

[JavaEventModel97] Sun Microsystems, *AWT : Delegation Event Model*, http://java.sun.com/j2se/1.3/docs/guide/awt/designspec/events.html, Février 1997.

[JavaReflection98] Sun Microsystems, *Java Core Reflection*, http://java.sun.com/j2se/1.3/docs/guide/reflection/spec/java-reflectionTOC.doc.html, 1998

[JavaTutorial02] Sun Microsystems, *The Java Tutorial: A practical guide for programmers*, http://java.sun.com/docs/books/tutorial/index.html, Mars 2002.

[Jini] Sun Microsystems, *Jini Network Technology,* http://wwws.sun.com/software/jini/

[JNI] Sun Microsystems, *Overview of the JNI*, http://java.sun.com/docs/books/tutorial/native1.1/concepts/index.html

[Khepera] *Khepera Robot*, http://www.k-team.com/robots/khepera/index.html

[KheperaManual99] K-Team, *Khepera User Manual*, http://www.k-team.com, 1999.

[MglStreams] Joerg Vogel, *Spacemouse integration into Java and Java 3D applications*, http://www.robotic.dlr.de/Joerg.Vogel

[Pixel] A. Van de Rostyne, *Pixel Gallery*, http://www.planetinternet.be/pixel/index.htm

[SideWinder] *Joystick Microsoft Sidewinder*, http://www.microsoft.com/hardware/sidewinder

[SpaceMouse] SpaceMouse Classic, http://www.logicad3d.com/products/Classic.html

[VRML] The Virtual Reality Modeling Language, http://www.vrml.org

[Xj3D] *Xj3D Open Source VRML/X3D Toolkit*, http://www.web3d.org/TaskGroups/source/xj3d.html

Systèmes de Réalité Virtuelle

AVANGO

[Tramberend99] H. Tramberend, *Avango: A distributed virtual reality framework*, IEEE Virtual Reality Conference, USA, Mars 1999.

[Avango] http://www.avango.de

BAMBOO

[Smith00] M.L. Smith, *Object Signing in Bamboo*, Thèse Master of Science in Modeling Virtual Environments and Simulation, Naval Postgraduate School, Mars 2000.

[Watsen98a] K. Watsen, M. Zyda, *Bamboo - A Portable System for Dynamically Extensible, Real-time, Networked, Virtual Environments*, IEEE Virtual Reality Annual International Symposium (VRAIS'98), USA, Mars 1998.

[Watsen98b] K. Watsen, M. Zyda, *Bamboo - Supporting Dynamic Protocols for Virtual Environments*, 1998 IMAGE Conference, USA, Août 1998.

[Oliveira00] M. Oliveira, J. Crowcroft, M. Slater, *Component Framework Infrastructure for Virtual Environments,* 3rd International Conference on Collaborative Virtual Environments, USA, Septembre 2000.

[Bamboo] http://watsen.net/Bamboo/

BLUE-C

[Gross01] M. Gross, O. Staadt, *The Blue-c Project*, ERCIM News No.44, Document accessible via l'URL : http://www.ercim.org/publication/Ercim_News/enw44/gross.html, Janvier 2001.

[Staadt00] O.G. Staadt, A. Kunz, M. Meier, M.H. Gross, *The Blue-c: Integrating real humans into a networked immersive environment*, 3rd International Conference on Collaborative Virtual Environments, Septembre 2000.

[Staadt01] O.G. Staadt, M. Näf, E. Lamboray, S. Würmlin, *JAPE: A Prototyping System for Collaborative Virtual Environments*, Eurographics 2001, The Eurographics Association and Blackwell Publishers, Vol. 20, No. 3, pp8–16, 2001.

[Blue-c] http://blue-c.ethz.ch/

DIVE

[Carlsson93] C. Carlsson, O. Hagsand, *DIVE – A Platform For Multi-User Virtual Environments*, Computer and Graphics, Vol. 17, No. 6, pp663-669, 1993.

[Hagsan96] O. Hagsan, *Interactive Multiuser VEs in the DIVE System,* IEEE Multimedia Magazine, Vol. 3, No. 1, 1996.

[Taxén00] G. Taxén, *A Practician's Guide to DIVE*, Document technique accessible via l'URL http://www.nada.kth.se/~gustavt, Janvier 2000.

[Frécon96] E. Frécon, O. Hagsand, *Dive Architecture*, Document technique accessible via l'URL http://www.sics.se/dive/manual/architecture.html, Septembre 1996.

[Frécon98] E. Frécon, M. Stenius, *DIVE : A Scaleable Network Architecture for Distributed Virtual Environments*, Distributed Systems Engineering Journal (special issue on Distributed Virtual Environments), Vol. 5, No. 3, pp91-100, Septembre 1998.

[Dive] http://www.sics.se/dive/

MRTOOLKIT

[Shaw93a] C. Shaw, M. Green, J. Liang, Y. Sun, *Decoupled Simulation in Virtual Reality with the MR Toolkit*, Information Systems, Vol. 11, No. 3, pp287-317, 1993.

[Shaw93b] C. Shaw, M. Green, *The MR Toolkit Peers Package and Experiment*, IEEE Virtual Reality Annual International Symposium, USA, Septembre 1993.

[Wang95] Q. Wang, M. Green, C. Shaw, *EM - An Environment Manager for Building Networked Virtual Environments*, IEEE Virtual Reality Annual International Symposium (VRAIS'95), USA, Mars 1995.

[MRToolkit] http://www.cs.ualberta.ca/~graphics/MRToolkit/

NPSNET

[Capps00] M. Capps, D. McGregor, D. Brutzman, M. Zyda, *NPSNET-V: A New Beginning for Dynamically Extensible Virtual Environments*, IEEE Computer Graphics and Applications, Vol. 20, No. 5, 2000.

[Macedonia94] M. Macedonia, M. Zyda, D. Pratt, P. Barham, S. Zeswitz, *NPSNET: A Network Software Architecture for Large Scale Virtual Environments*, Presence, Vol. 3, No. 4, 1994.

[Macedonia95a] M. Macedonia, M. Zyda, D. Pratt, D. Brutzman, P. Barham, *Exploiting Reality with Multicast Groups: A Network Architecture for Large-scale Virtual Environments*, IEEE Virtual Reality Annual International Symposium(VRAIS'95), USA, Mars 1995.

[Macedonia95b] M. Macedonia, D. Brutzman, M. Zyda, D. Pratt, P. Barham, J. Falby, J. Locke, *NPSNET: A Multiplayer 3D Virtual Environment over the Internet*, ACM - 1995 Symposium on Interactive 3D Graphics, USA, Avril 1995.

[NPSNETSysOv] NPSNET Research Group, *NPSNET IV.9 System Overview*, 1996

[NPSNET] http://www.movesinstitute.org/~npsnet/

[NPSNET-V] http://www.movesinstitute.org/~npsnet/v/

ORIS-AREVI

[Harrouet00] F. Harrouet, *oRis : s'immerger par le Langage pour le Prototypage d'Univers Virtuels à base d'Entités Autonomes*, Thèse de Doctorat, Université de Bretagne Occidentale, 2000.

[Querrec02] R. Querrec, *Les Systèmes Multi-Agents pour les Environnements Virtuels de Formation*, Thèse de Doctorat, Université de Bretagne Occidentale, 2002.

[Tisseau01] J. Tisseau, *Réalité Virtuelle -Autonomie in virtuo-,* Habilitation à Diriger des Recherches, Université de Rennes 1, 2001.

VIPER

[Torguet98] P. Torguet, *VIPER: Un modèle de calcul reparti pour la gestion d'environnements virtuels*, Thèse de Doctorat, Université Paul Sabatier, France, 1998.

[Torguet00] P. Torguet, O. Balet, JP. Jessel , E. Gobetti, J. Duchon, E. Bouvier, *CAVALCADE a system for collaborative virtual prototyping*, Journal of Design and Innovation Research - Special Virtual Prototyping, Vol. 2, No. 1, 2000.

VRJUGGLER

[Bierbaum00a] A. Bierbaum, *VR Juggler: A Virtual Platform for Virtual Reality Application Development*, Master's thesis, Computer Engineering, Iowa State University, 2000.

[Bierbaum00b] A. Bierbaum, C. Just, P. Hartling, C. Cruz-Neira, *Flexible Application Design Using VR Juggler*, SIGGRAPH 2000, USA, Juillet 2000.

[Just98] C. Just, A. Bierbaum, A. Baker, C. Cruz-Neira, *VR Juggler: A Framework for Virtual Reality Development,* 2nd Immersive Projection Technology Workshop (IPT98), France, Mai 1998.

[Just00] C. Just, *Performance analysis of a virtual reality development environment : Measuring and tooling performance of VR Juggler,* Master's thesis, Computer Engineering, Iowa State University, 2000.

[VRJuggler] http://www.vrjuggler.org/

WORLDTOOLKIT

[WorldToolkit] Sense8 Corporation, *WorldToolkit,* http://www.sense8.com/products/wtk.html

[World2World] Sense8 Corporation, World2World, http://www.sense8.com/products/w2w.html