Formalisation des interactions collaboratives en environnements virtuels collaboratifs

Laurent Aguerreche, Thierry Duval

Formalisation des interactions collaboratives en environnements virtuels collaboratifs

Laurent Aguerreche* — IRISA – INSA de Rennes
Thierry Duval† — IRISA – Université de Rennes 1

RÉSUMÉ

Cet article présente un travail en cours visant à décrire ce que sont un outil d'interaction et un objet interactif dans un environnement virtuel collaboratif pour les faire communiquer. Un protocole d'interaction est ainsi décrit. Il a été implémenté et testé sur plusieurs plates-formes de réalité virtuelle et est asynchrone.

La possibilité de décrire des outils, des objets interactifs (notamment la partie comportementale) et de les faire communiquer ouvre la voie à l'élaboration d'un langage de description d'environnements virtuels.

Mots-clés: Environnements virtuels collaboratifs, Protocole d'interaction, Interaction 3D, Formalisation des communications entre outils virtuels et objets interactifs.

1 INTRODUCTION

De nombreux articles décrivant l'architecture de plates-formes de réalité virtuelle ou des techniques d'interactions ont été écrits mais leur implémentation a été rarement étudiée. Cet article détaille un travail en cours cherchant à décrire un outil [2], un objet interactif et la communication entre eux.

Les scénarios d'utilisation envisagés peuvent être basiques : la simple préhension d'un objet par un utilisateur, ou plus complexes : par plusieurs utilisateurs. Le protocole d'interaction présenté ici doit également être utilisable dans le cas d'outils/interacteurs répartis géographiquement ce qui introduit des latences dans les communications. En conséquence des latences possibles de communication, le protocole présenté est asynchrone. Des scénarios plus élaborés devraient être envisageables. Dans [1], un designer, un ingénieur et un responsable marketing travaillent ensemble sur un prototype de voiture. L'ingénieur peut facilement étudier les pièces de la voiture pendant que le designer peut la présenter à un responsable de vente qui enregistre des bouts de la présentation pour les réutiliser plus tard. Ce scénario fait, par ailleurs, intervenir des périphériques aux capacités en puissance de calcul et d'affichage diverses (des PDA face à des stations de travail par exemple).

2 ÉTAT DE L'ART

Nous nous plaçons dans le cas où plusieurs utilisateurs répartis géographiquement sont en train d'interagir sur des objets d'un monde qu'ils partagent. Certains objets peuvent même être manipulés simultanément par plusieurs personnes. Cette répartition géographique nécessite une architecture logicielle adaptée. La littérature propose plusieurs approches : centralisée[10], répliquée[9], pair à pair[1], *etc*. Nous présentons un protocole d'interaction en nous appuyant sur certaines particularités de la plate-forme logicielle de réalité virtuelle OpenMASK qui exploite une architecture réseau hybride[14]. Notre protocole se situe donc dans le cadre d'un ensemble d'objets qui échangent des messages entre eux. Par ailleurs, OpenMASK fournit une couche de communication sûre qui assure l'absence de perte de paquets réseaux.

Les métaphores d'interaction dans des environnements virtuels collaboratifs sont nombreuses [4] et peuvent se classer en deux catégories : métaphores égocentriques et métaphores exocentriques. Les métaphores exocentriques placent l'utilisateur du « point de vue de Dieu » pour lui donner une vue d'ensemble depuis l'extérieur de l'environnement virtuel. Inversement, une métaphore égocentrique place l'utilisateur à l'intérieur de l'environnement virtuel : c'est le type d'action que peut avoir une personne dans le monde réel. Les métaphores égocentriques peuvent se ranger en deux catégories [18] : pointeurs et mains virtuelles (ex : la technique du « Go-Go » [17]). Il y a parmi les pointeurs la métaphore du rayon virtuel qui permet de désigner un objet et, éventuellement, de l'attacher au rayon. Cette technique, au départ mono-utilisateur, peut être transformée en une technique adéquate pour des interactions collaboratives [19]. Il est également possible de combiner des métaphores mono-utilisateurs selon des degrés de liberté [16].

Durant une interaction, la prise de conscience intervient à deux niveaux : d'un côté pour l'utilisateur actionnant un outil, de l'autre pour rendre les outils et les objets interactifs mutuellement au courant de leurs actions. Des métaphores visuelles telles que des flèches ou des curseurs indiquent la direction d'un déplacement. Différentes modalités peuvent être combinées pour améliorer la qualité des retours vers l'utilisateur [21]. Les retours d'informations destinés à l'utilisateur lui permettent de mieux comprendre ce que ses actions sont en train de causer. Dans un environnement virtuel collaboratif, les utilisateurs doivent comprendre que leurs propres actions sont contraintes par les actions des autres utilisateurs et comprendre les actions des autres utilisateurs pour pouvoir prendre la suite de leurs actions, ou pour se coordonner avec eux [11]. L'élargissement du champ de vision [7] peut, par exemple, permettre d'apercevoir la présence d'un autre utilisateur. La matérialisation du champ de vision d'un utilisateur peut donner des indications sur ce qu'il est susceptible de vouloir faire [8].

Peu de travail semble avoir été fait concernant la communication entre les outils et les objets interactifs. Les *Smart Objects* [12] encapsulent dans la description des objets ses caractéristiques, propriétés, comportements et les scripts associés et à exécuter avec chaque interaction [20]. Dans le domaine des environnements virtuels, les outils/métaphores sont souvent décrits puis expérimentés, mais la façon de les implémenter est rarement expliquée. L'article [5] a servi de départ à l'élaboration du protocole d'interaction décrit ici.

Enfin, le but de ce protocole est de permettre la communication entre outils et objets interactifs. La description des outils, des objets interactifs et des communications entre eux devraient conduire, à plus long terme, vers un langage de descriptions d'environnements virtuels [6].

3 LES OUTILS ET LES OBJETS INTERACTIFS

Un environnement virtuel collaboratif est vu comme un ensemble d'outils et d'objets interactifs. Un utilisateur, un humain ou un humanoïde par exemple, agit sur un objet interactif au travers

*e-mail : laurent.aguerreche@irisa.fr
†e-mail : thierry.duval@irisa.fr

[1] En anglais « *Peer to Peer* » (P2P).

d'un outil. Un outil peut également être un objet interactif pour pouvoir être pourvu de capacités relatives à celles d'un objet interactif : changements de propriétés comme la couleur, la forme, la position, *etc*. Avec d'un côté des outils et de l'autre des objets interactifs, qui doivent être amenés à communiquer, un protocole de communication pour les interactions dans des environnements virtuels est nécessaire.

Le rôle d'un outil est de proposer des données à un objet interactif lorsque l'outil et l'objet interactif sont en cours d'interaction. Ainsi, lorsqu'un outil cherche à modifier la position d'un objet interactif, il doit lui envoyer une nouvelle valeur désignant une nouvelle position.

De son côté, l'objet interactif a pour rôle de recevoir des données envoyées par un outil et de les traiter : l'outil propose, l'objet interprète. Lorsque plusieurs outils agissent simultanément sur un même objet interactif, pour modifier une même propriété (ex : la position), l'objet interactif va recueillir les différentes propositions des outils et les combiner. Avec des outils proposant des positions, l'objet peut, typiquement, calculer la moyenne des positions proposées. L'objet contient donc son propre comportement vis-à-vis d'outils.

4 Messages entre outils et objets interactifs

Nous allons à présent détailler les messages que les outils et les objets interactifs s'échangent via un réseau fiable. Nous commencerons par le cas d'un outil avec un seul objet interactif, puis nous verrons le cas où deux outils manipulent en même temps un même objet interactif. Nous parlerons également d'une gestion des permissions d'accès aux attributs des objets interactifs.

4.1 Contrôle simple

L'outil va passer dans différents états pour parvenir à contrôler un objet interactif. L'outil va commencer par ouvrir une communication avec un objet qui a été désigné par un utilisateur de l'environnement virtuel grâce à, par exemple, un rayon ou un menu. Avant d'essayer de contrôler des attributs d'un objet interactif, un outil doit savoir quels attributs il pourra manipuler :

1. l'outil demande à l'objet interactif les propriétés qu'il pourrait modifier en lui envoyant le message *get_accessible_parameters* ;
2. l'objet répond par le message *accessible_parameters* en l'accompagnant de la liste des paramètres accessibles ;

L'outil peut prendre le contrôle de certains attributs accessibles. La séquence de communication d'un outil contrôlant un objet interactif est la suivante :

1. l'outil essaie de prendre le contrôle des propriétés accessibles. Il pourrait ainsi essayer de prendre le contrôle de la position de l'objet interactif en envoyant le message *control_take_over_and_get_current_values* à l'objet interactif en l'accompagnant de l'identifiant « Position » ;
2. l'objet interactif renvoie la valeur de la position en utilisant le message *current_value* qui est suivi de l'identifiant de la valeur envoyée ainsi que de la valeur elle-même. Un fois que toutes les valeurs des propriétés dont l'outil voulait prendre le contrôle ont été envoyées à l'outil, l'objet interactif envoie le message *control_taken_of* pour indiquer la fin des envois. Avec cette étape, l'outil prend connaissance des propriétés qu'il contrôle à présent et les valeurs qu'elles contenaient. Ces valeurs sont utilisables par l'outil pour qu'il puisse initialiser ses propres calculs pour envoyer des demandes à l'objet interactif (ex : la position).
3. l'outil peut à présent proposer des valeurs à l'objet interactif avec le message *send_value*. Il y a autant de messages envoyées que de valeurs à transmettre ;
4. l'objet interactif informe l'outil d'une nouvelle valeur pour l'une de ses propriétés grâce au message *effective_value* qui est accompagnée de la valeur correspondante ;
5. l'outil peut relâcher le contrôle d'une propriété de l'objet interactif en lui envoyant le message *control_release* accompagné de l'identifiant de la propriété à relâcher.

La séquence qui vient d'être donnée est présente sur la figure 1). Un outil qui essaie de prendre le contrôle de la position d'un objet interactif va soit réussir, soit échouer. Si l'outil échoue lors de la prise de contrôle, la liste des propriétés dont il a pris le contrôle est vide.

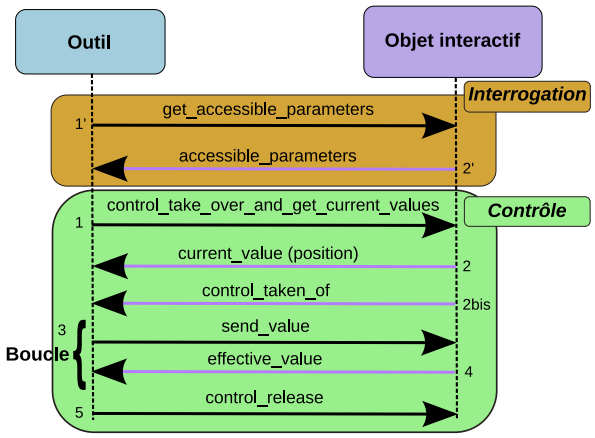

FIG. 1: Séquence de dialogue entre un outil et un objet interactif.

4.2 Contrôle partagé

Lors d'un contrôle partagé, au moins deux outils sont en train d'agir sur un même objet. Dans cette situation, chaque objet a besoin de savoir qu'il n'est pas seul à interagir et doit suivre les évolutions de l'objet interactif. L'outil doit savoir qu'il n'est pas seul à agir pour informer l'utilisateur de l'outil afin qu'il comprenne que les effets des actions qu'il propose sont contraintes par les actions des autres outils, et afin qu'il puisse coordonner ses actions avec les effets produits par les autres outils, voire se coordonner avec les autres utilisateurs. L'outil doit également savoir qu'il n'est pas seul à interagir pour éventuellement adapter ses propres calculs à la situation.

Prenons le cas où un outil, nommé T_1, est en train d'interagir avec un objet interactif nommé O. Si un autre outil, nommé T_2, prend le contrôle d'attributs de O alors T_1 recevra un message *control_taken_of* accompagné de l'identifiant de T_2 ainsi que de l'identifiant dont il prend le contrôle. Lorsque T_2 relâchera le contrôle, T_1 recevra le message *control_released_of* qui est suivi par les mêmes valeurs que pour *control_taken_of*.

4.3 Gestion des accès à un objet interactif

Les applications de réalité virtuelle peuvent mettre en action différentes situations de la vie réelle : par exemple, un enseignant qui montre comment faire à des étudiants, un chef qui a accès à des actions avancées par rapport aux gens qu'il dirige, *etc*. De fait, les outils et les objets interactifs doivent incorporer une gestion des accès. Chaque propriété d'un outil interactif est munie d'un niveau accès qu'un outil doit posséder s'il veut pouvoir la modifier. Nous arrivons ainsi à une situation où la prise de contrôle d'une propriété par un outil peut échouer. De plus, les niveaux d'accès permettent d'envisager une situation où des outils peuvent entraîner « l'éjection » d'un autre outil déjà en cours d'interaction pour prendre sa

place : cette situation se produit lorsque l'objet est placé dans un mode d'interaction exclusif. Lorsqu'un objet est éjecté par l'objet interactif qu'il contrôle suite à la venue d'un outil, il reçoit le message *control_ended* suivi de l'identifiant de la propriété dont il a perdu le contrôle.

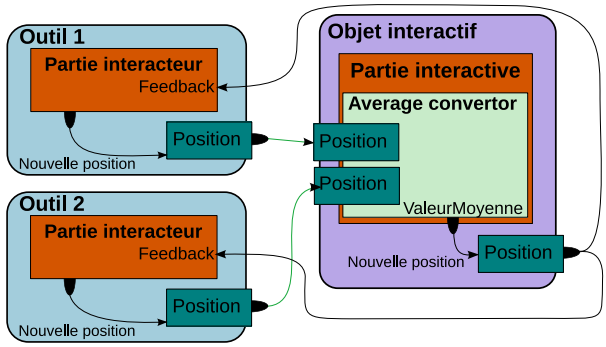

FIG. 2: Schéma montrant deux outils agissant sur la position d'un objet interactif.

4.4 Observations concernant le protocole d'interactions

Nous venons de donner l'ensemble des messages définis et utilisés par le protocole d'interaction. Ce protocole est actuellement implémenté sur OpenMASK et Virtools.

Dans la partie qui suit, nous allons montrer une implémentation du protocole sur OpenMASK qui se veut exploitable par d'autres plates-formes de réalité virtuelle. Nous verrons également le cas de la gestion d'objets complexes pour lesquels il faudra étendre le protocole présenté.

5 IMPLÉMENTATION

Nous expliquerons ici comment nous avons implémenté sur OpenMASK le protocole décrit, ainsi que les méthodes qui pourraient être employées d'une façon plus générale.

5.1 Quelques principes d'OpenMASK

OpenMASK[13] est une plate-forme de réalité virtuelle employant un ensemble d'*objets simulés*. Chaque objet simulé est une classe C++ qui contient une méthode *computeParameters* appelée selon une fréquence qui fait, elle aussi, partie de l'objet simulé.

Un objet simulé est composé d'attributs pouvant être munis d'entrées et de sorties. Certaines entrées peuvent être connectées à des sorties de divers objets. À chaque tic de son horloge, l'objet simulé voit sa méthode *computeParameters* être appelée ce qui peut entraîner la lecture des valeurs placées sur des sorties d'autres objets simulés, et l'écriture de valeurs sur ses propres sorties.

5.2 Spécialisation par agrégation de composants

Tout le comportement d'un objet simulé en fonction des données qu'il lit via ses entrées est dépendant de sa méthode *computeParameters*. La modification du comportement d'un objet simulé peut donc passer par l'héritage de la classe qui l'implémente afin de modifier la méthode *computeParameters*. En pratique, l'héritage de classes devient rapidement source de difficultés : l'arbre représentant les héritages devient profond et rend délicat l'ajout de nouvelles classes tout en évitant des duplications de code et des problèmes de conception.

Une approche alternative à l'héritage consiste à modifier le comportement d'un objet en agrégeant plusieurs composants logiciels qui seront exécutés à tour de rôle. C'est l'exécution de ces composants qui modifie le comportement de l'objet simulé qui les agrège. Dans OpenMASK, ces composants sont des extensions.

Une extension est munie de deux méthodes : *preComputeParameters* et *postComputeParameters* qui seront respectivement appelées avant et après *computeParameters*. Par défaut, ces deux méthodes sont vides et leur exécution ne fait donc rien. Une extension est également munie d'une méthode lui permettant de charger une configuration initiale. La création d'une extension passe par l'héritage d'une classe implémentant une extension de base. À l'intérieur d'une extension, il est possible d'accéder à l'objet simulé auquel elle se rattache pour, par exemple, lire les sorties d'un objet simulé ou modifier les sorties de l'objet courant. Enfin, l'ordre dans lequel chaque extension sera appelée peut être précisé dans le fichier de configuration d'OpenMASK.

5.3 Découpage d'un outil par extensions

Nous allons à présent montrer comment employer les extensions pour transformer un objet simulé quelconque en un outil.

5.3.1 De l'objet simulé à l'outil

D'abord, rappelons qu'un outil est un objet simulé capable de manipuler les attributs d'un objet interactif. Nous allons transformer un objet simulé quelconque en un outil en lui ajoutant une extension implémentant le protocole d'interaction, cette extension est l'extension de *contrôle*. L'utilisation de l'extension de contrôle passe par l'extension de *manipulation*. Enfin, l'extension de manipulation agit sur les objets interactifs précédemment sélectionnés dont les identifiants sont enregistrés dans l'extension *sélection*.

L'étape de *sélection* suivie de l'étape de *manipulation* fait référence aux étapes communément décrites pour la manipulation d'objets dans des environnements virtuels [15, 3].

Les explications données ci-après sont illustrées par la figure 3.

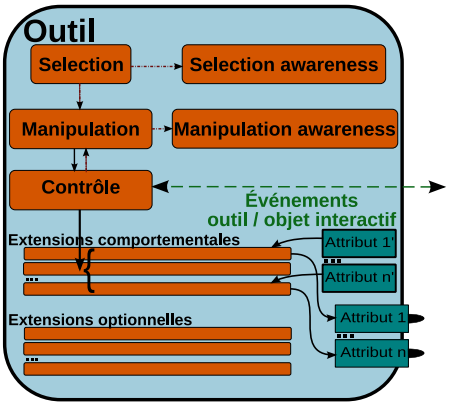

FIG. 3: Un outil simple, ses extensions, ses attributs internes et ceux qu'il expose en sortie. Pour le contrôle, des événements sont échangés entre l'outil et l'objet interactif au travers de l'extension de contrôle de l'outil.

5.3.2 Sélection

Cette extension contient la liste des derniers objets sélectionné par l'utilisateur de l'outil. Prenons, par exemple, le cas d'une sélection par *ray-casting* (lancer de rayons). Un événement est envoyé à l'outil virtuel pour lui demander de lancer un rayon. Cet événement peut être issu de l'appuie par l'utilisateur d'un bouton d'un périphérique matériel, ou être purement logiciel. Le rayon est lancé dans la direction de l'objet. Selon l'implémentation de l'extension *sélection*, ce lancer de rayons peut être effectué par le moteur de

visualisation de l'application ou, par exemple, un moteur physique. Les objets sélectionnés sont alors enregistrés dans l'extension.

Une extension de sélection peut proposer la sélection par divers mécanismes : par l'affichage d'un menu à l'utilisateur, par l'utilisation d'un mécanisme de reconnaissance vocale, *etc*. Ces possibilités sont fonction de son implémentation.

5.3.3 Manipulation

L'extension *manipulation* ordonne la prise de contrôle d'attributs des objets sélectionnés. Cette extension est responsable du déclenchement de la prise de contrôle, ou bien du relâchement, d'attributs d'un objet interactif. Elle a la responsabilité de proposer à l'utilisateur diverses possibilité d'interaction avec un objet interactif, ou encore de savoir interpréter les données provenant de périphériques matériels complexes pour l'interaction si besoin.

Pour connaître la liste des objets sélectionnés au travers de l'extension *sélection*, l'extension *sélection* présente une interface *ISelectedObject* comportant une méthode *updateSelectedObjects* qui est appelée chaque fois que des objets sont sélectionnés. L'extension *manipulation* implémente la méthode *updateSelectedObjects*. Nous utilisons donc ici un mécanisme de *listeners*. L'extension *manipulation* connaît l'existence d'une extension *sélection* grâce au fichier de configuration de l'application, et va s'enregistrer auprès d'elle.

L'extension *manipulation* va interroger un objet interactif pour connaître ses attributs accessibles et essayer de prendre le contrôle de certains d'entre eux. Cette communication se fait au travers de l'extension *contrôle* qui, pour cela, expose des méthodes publiquement. De plus, l'extension *contrôle* propose une interface d'écoute de la même façon que l'extension *sélection* qui permet à l'extension *manipulation* de connaître l'évolution du contrôle qu'elle effectue.

5.3.4 Contrôle

Cette extension implémente le protocole d'interaction. C'est elle qui permet d'interroger un objet interactif pour connaître ses attributs accessibles par l'outil et pour pouvoir les manipuler. Cette extension expose par conséquent un ensemble de méthodes pour déclencher interrogation et contrôle :

- ouverture/fermeture de communication :
  startSessionWithObject(objectId),
  stopSessionWithObject(objectId) ;
- interrogation :
  getAccessibleParameters(objectId) ;
- contrôle :
  controlTakeOverAndGetCurrentValues(objectId, attributesToControl),
  releaseControlWithObject(objectId, attributesToRelease),
  controlEnded(objectId, attributesReleased) ;
- suivi du contrôle par d'autres outils :
  controlTakenBy(objectId, attributesControlled),
  controlReleasedBy(objectId, attributesReleased).

Le protocole d'interaction est asynchrone. Par conséquent, les réponses attendues seront obtenues par des *callbacks* réagissant aux événements contenant les réponses. Il correspond une réponse à chaque événement envoyé par les méthodes ci-dessus. La méthode *processCurrentValuesSentAndControlTaken* joue un rôle particulier parce que, du fait qu'elle permet de connaître les attributs effectivement contrôlés par l'outil, elle est responsable de l'initialisation d'une extension particulière appelée *extension comportementale*. Une extension de contrôle connaît donc un ensemble d'extensions comportementales qu'elle aura à appeler.

5.3.5 Comportement

Une extension *comportementale* est composée de trois méthodes. La méthode *initComputes* permet une initialisation de l'extension. La méthode *compute* est appelée à chaque tic d'horloge de l'outil auquel elle est rattachée. Chaque appel à cette méthode permet de calculer de nouvelles valeurs à proposer aux sorties de l'outil. Enfin, une méthode *endComputes* permet de gérer le moment où l'outil vient de perdre le contrôle de l'attribut correspondant à l'extension comportementale courante.

L'association entre une extension comportementale et un attribut de l'outil est décrite dans le fichier de configuration d'OpenMASK. Ainsi, lorsque l'attribut « Position », par exemple, de l'outil est mis en correspondance avec un attribut de position d'un objet interactif, une extension comportementale pour une position est appelée.

5.3.6 Gestion de la prise de conscience

Un utilisateur humain employant un outil virtuel a besoin d'informations pouvant être visuelles, sonores, haptiques, *etc.* pour lui faire mieux comprendre l'effet et l'état de ses actions au travers de l'outil virtuel. Ainsi, lorsqu'un utilisateur humain sélectionne un objet virtuel, il lui est utile d'avoir un retour visuel pour comprendre qu'il vient de sélectionner l'objet : par exemple, l'objet change de couleur, ou de forme, ou il apparaît une boîte englobante autour de lui, *etc*.

Afin de suivre les évolutions d'une sélection, une extension dédiée peut être créée, elle correspond à l'extension *selection awareness* sur la figure 3. Cette extension implémente l'interface *ISelectedObject*. Lorsqu'un objet est sélectionné, elle peut, par exemple, envoyer une demande au moteur de visualisation pour modifier l'apparence de l'objet sélectionné.

De la même façon, une extension *manipulation awareness* peut être ajoutée. Elle peut elle aussi renvoyer des informations vers l'utilisateur pour lui signifier qu'il vient, par exemple, de commencer une manipulation.

En découplant suffisamment ces extensions des outils, on parvient à les réutiliser dans des outils divers.

5.4 Extensions et objets interactifs

Un objet interactif est construit à partir d'un assemblage d'extensions. En effet, de la même façon qu'un outil n'est, au départ, qu'un objet simulé, un objet interactif est aussi un objet simulé dont nous allons modifier le comportement grâce à des extensions.

L'extension de base d'un objet interactif est l'*extension interactive*. Cette extension implémente le protocole d'interaction et sait donc répondre aux demandes concernant les paramètres accessibles, la prise de contrôle d'attributs, *etc.*

D'autres extensions d'un objet interactif peuvent suivre son évolution. À l'image des extension d'« *awareness* » dans l'outil, ses extensions viennent s'enregistrer auprès de l'*extension interactive* et implémentent une interface. L'interface implémentée est de type *IInteractiveObjectListener* et contient actuellement deux méthodes :

- *newInteractor(toolId)* est appelée dès lors que l'outil nommé *toolId* a pris le contrôle d'au moins un attribut de l'objet interactif ;
- *leftInteractor(toolId)* est appelée lorsque l'outil *toolId* a relâché le contrôle de tous les attributs qu'il manipulait.

Par la suite, cette interface aura probablement à être étendue pour apporter plus d'informations aux extensions l'écoutant, comme les identifiants des attributs contrôlés ou relâchés.

5.4.1 Combinaison de valeurs

L'objet interactif a pour rôle la réception et l'interprétation de données provenant des outils le contrôlant. Ces données sont réceptionnées par les attributs contrôlés de l'objet interactif.

Chaque attribut d'un objet interactif est muni d'un *connecteur* dont le rôle est de recueillir les valeurs fournies par des sorties de différents outils et de les combiner en une valeur à placer sur l'entrée correspondante de l'objet interactif. Actuellement, un connec-

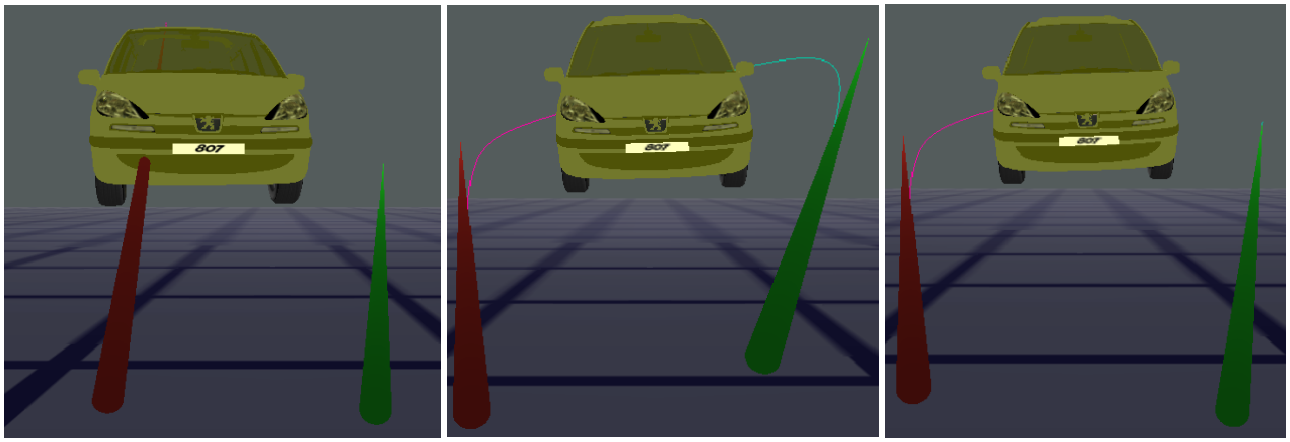

FIG. 4: À gauche, un seul rayon contrôle la position de la voiture. Au centre, deux rayons modifient simultanément la position de la voiture. Ces rayons sont courbés pour indiquer les actions opposées. À droite, un rayon vient de terminer ses actions, celui qui continue à interagir maintient, pour l'instant, son allure. (Modèle 3D de voiture utilisé avec l'autorisation de PSA Peugeot Citroën)

teur se résume souvent au calcul de la moyenne, ou du barycentre, des valeurs qu'il obtient.

5.5 Contrôle d'un objet interactif

Les captures d'écran de la figure 4 illustrent le cas de deux rayons[19], l'un manipulé par un utilisateur, l'autre par une autre personne, qui se courbent pour montrer que les actions pour modifier la position de la voiture que cherche à appliquer un utilisateur sont contraintes par celles de l'autre utilisateur. La position de la voiture est obtenue par une moyenne des positions proposées.

5.6 Contrôle d'un objet interactif complexe

Nous qualifions un objet interactif de complexe lorsque celui-ci dispose d'un grand nombre d'attributs qui impose une approche globale pour le représenter.

Un objet du monde réel, tel qu'une voiture par exemple, peut souvent être décomposé en un ensemble d'objets interactifs. La décomposition d'une voiture pourrait employer un ensemble de portes, un volant, des sièges, *etc*. Il est alors possible de décrire des contraintes entre objets de façon *locale* : par exemple, un siège ne peut glisser que parallèlement au châssis et de façon limitée.

Une molécule composée d'un nombre élevé d'atomes est plus difficile à mettre en œuvre par une approche locale. En effet, il faudrait associer à chaque atome un objet interactif qui serait muni d'une extension interactive et qui calculerait sa position en fonction de celles des autres atomes et de conditions physiques. Pour éviter un grand nombre de « petits » calculs, il peut être souhaitable d'opter pour une approche *globale* où un seul objet interactif fait office de moyen d'accès à la molécule et où c'est lui qui calcule les positions de chaque atome et conserve la cohérence de l'ensemble de la structure.

Pour parvenir à n'interagir qu'avec certaines parties de la molécule alors qu'elle n'est manipulable que par un seul objet interactif, il faut décomposer la molécule en parties et leurs associer des attributs. La molécule possède un ensemble d'attributs dont chaque identifiant est unique. Au moyen du fichier de configuration, nous associons une partie de l'objet géométrique à un identifiant de façon à obtenir cet identifiant lors d'un lancer de rayons sur cette partie. Toujours par le fichier de configuration, nous associons à l'identifiant précédent un ensemble d'attributs de l'objet interactif. Le protocole d'interaction a besoin d'être étendu au niveau de la fonction *getAccessibleParameters* en rajoutant à cette fonction l'identifiant de la partie sélectionnée. À présent, la méthode *getAccessibleParameters* ne renvoie plus l'ensemble des attributs accessibles d'un objet interactif mais uniquement ceux qui ont été associés à la partie dont l'identifiant est donné. L'identifiant sert de filtre des attributs.

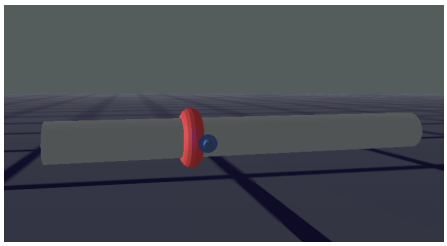

FIG. 5: Un « *slider* » dont la bague a été sélectionnée est représentée. Ici, la bague est en train d'être déplacée vers la gauche de l'image au moyen du pointeur bleu actionné par l'utilisateur.

La figure 5 montre le cas simple d'un « *slider* ». Cet objet peut être utilisé comme un « *widget* » de contrôle de l'application, à l'image des « ascenseurs » employés dans des fenêtres en 2D. Le « *slider* » présenté ici n'emploie qu'un seul objet interactif. La sélection de la bague (en rouge sur la figure) est faite par lancer de rayons et récupération, par la suite, de son identifiant. Le déplacement de la bague le long du « *slider* » est rendue possible par une extension placée dans l'objet interactif qui impose cette contrainte.

5.7 Contrôle de plusieurs objets interactifs

Un utilisateur humain peut avoir envie de contrôler un ensemble d'objet interactifs par le biais d'un seul outil de la même façon qu'il le fait avec des icônes placées sur son bureau : une fois un ensemble d'icônes sélectionnées, un clic droit permet de lancer une action sur l'ensemble.

Nous souhaitons permettre à un utilisateur humain d'un outil virtuel de sélectionner un ensemble d'objet interactif pour leur appliquer un traitement commun. Le déclencheur du traitement sera un événement, lancé par exemple par un appui sur un bouton d'un périphérique matériel, et l'arrêt sera donné par un autre événement. Une décision devra alors être prise par ceux qui écriront les outils puisque deux comportements seront possibles : soit l'outil essaiera de contrôler l'ensemble des objets sélectionnés et abandonnera l'ensemble du contrôle si au moins un attribut d'un objet interactif ne peut être contrôlé, soit le contrôle se poursuivra avec les seuls attributs dont le contrôle a été obtenu.

Pour permettre à un outil de manipuler plusieurs outils, nous de-

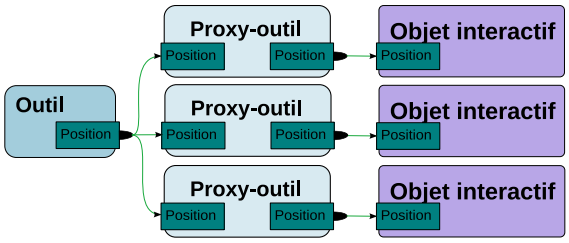

FIG. 6: L'outil maître donne sa position à chaque proxy-outil. À leur tour, chaque proxy-outil calcule une position à proposer à l'objet interactif dont il a le contrôle.

vons introduire l'idée d'*outils intermédiaires*, ou *proxy-outils*, dont le principe de fonctionnement est illustré par la figure 6.

5.7.1 Usage des proxy-outils

Un outil pourrait contenir tout le code lui permettant de manipuler plusieurs objets interactifs simultanément mais cela serait complexe. L'approche par des outils intermédiaires permet de déléguer le contrôle d'un objet interactif à un proxy-outil lui correspondant. L'outil principal devient alors un gestionnaire qui ordonne des prises de contrôle à ses proxy-outils et qui suit les résultats de leurs actions.

Dans ce schéma, une extension de contrôle doit être déplacée dans chaque proxy-outil. Elle écoute des événements provenant de l'outil maître : interrogation de l'objet interactif, début de contrôle, fin de contrôle, *etc.*, et lui renvoie par des événements les résultats des différentes actions. Il est également nécessaire d'ajouter les extensions comportementales à l'intérieur de chaque proxy-outil.

L'extension manipulation de l'outil maître est munie d'une partie lui servant de *gestionnaire de contrôle*. Cette partie est une simple classe C++ qui a pour rôle d'envoyer des événements aux proxy-outils et d'en recevoir en retour. Lorsque le gestionnaire reçoit des informations en retour, il a pour rôle de les router vers l'extension de manipulation. Cette organisation générale est donnée par la figure 7.

Actuellement, un proxy-outil est un objet simulé qui est la copie de l'outil maître dépourvu de ses extensions. Le fichier de configuration permet de préciser un ensemble d'extensions à ajouter dans les proxy-outils créés.

6 SITUATION ACTUELLE ET TRAVAUX FUTURS

Le protocole d'interaction pour des objets interactifs simples est implémenté sur OpenMASK et Virtools. La version pour les objets complexes n'existe que sur OpenMASK tandis que celle permettant la manipulation multiple d'objets est en cours de développement sur OpenMASK également.

Le travail se porte actuellement sur la poursuite du développement de l'implémentation gérant la manipulation simultanée de plusieurs objets interactifs.

Au fil des implémentations de différentes métaphores d'interaction, les interfaces de programmation des différentes extensions s'affinent et deviennent plus réutilisables entre outils et entre objets interactifs. Le fait d'utiliser systématiquement un même ensemble d'extensions permet de développer des outils-auteurs qui les manipuleront.

Enfin, la question de faire communiquer différentes plates-formes logicielles de réalité virtuelle entre elles se pose : OpenMASK, Virtools et Spin3D.

7 CONCLUSION

La séparation outil/objet interactif semble pertinente à travers les tests effectués avec le protocole d'interaction que nous avons décrit. Il permet la création de composants réutilisables entre outils et entre objets interactifs. L'assemblage de ces composants au travers d'un ficher de configuration abouti à la description de ce que sont les outils et les objets interactifs.

Pour l'instant, la façon dont un objet interactif combine un ensemble de données provenant de plusieurs outils est figée par du code écrit en C++. À plus long terme, nous souhaitons décrire le comportement des objets interactifs ce qui nous permettra d'aller vers un langage de description d'environnements virtuels basé sur des composants réutilisables qu'il sera possible de mettre en œuvre à l'aide d'outils-auteurs.

REMERCIEMENTS

Ces travaux ont été réalisés dans le cadre du projet Part@ge 06 TLOG 031 : plate-forme RNTL financée par l'ANR.

RÉFÉRENCES

[1] V. Bayon, G. Griffiths, and J. R. Wilson. Multiple decoupled interaction : An interaction design approach for groupware interaction in co-located virtual environments. *Int. J. Hum.-Comput. Stud.*, 64(3) :192–206, 2006.

[2] D. A. Bowman and L. F. Hodges. User interface constraints for immersive virtual environment applications. Technical report, Graphics, Visualization, and Usability Center GIT-GVU-95-26, 1995.

[3] D. A. Bowman and L. F. Hodges. An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In *SI3D '97 : Proceedings of the 1997 symposium on Interactive 3D graphics*, pages 35–ff. ACM, 1997.

[4] D. A. Bowman, E. Kruijff, J. Joseph J. LaViola, and I. Poupyrev. *3D User Interfaces : Theory and Practice*. Addison-Wesley/Pearson Education, 2005.

[5] T. Duval and C. L. Tenier. Interactions 3d coopératives en environnements virtuels avec openmask pour l'exploitation d'objets techniques. *Mécanique & Industries*, 5 :129–137, 2004.

[6] P. Figueroa, M. Green, and H. J. Hoover. Intml : a description language for vr applications. In *Web3D '02 : Proceedings of the seventh international conference on 3D Web technology*, pages 53–58, New York, NY, USA, 2002. ACM.

[7] M. Fraser, S. Benford, J. Hindmarsh, and C. Heath. Supporting awareness and interaction through collaborative virtual interfaces. In *UIST '99 : Proceedings of the 12th annual ACM symposium on User interface software and technology*, pages 27–36. ACM Press, 1999.

[8] M. Fraser, T. Glover, I. Vaghi, S. Benford, C. Greenhalgh, J. Hindmarsh, and C. Heath. Revealing the realities of collaborative virtual reality. In *CVE '00 : Proceedings of the third international conference on Collaborative virtual environments*, pages 29–37, New York, NY, USA, 2000. ACM Press.

[9] S. Gaeremynck, S. Degrande, L. Grisoni, and C. Chaillou. Utilisation des composants pour la représentation des objets virtuels. In *Journées AFIG – Bordeaux 2006*, 2006.

[10] R. Gossweiler, R. J. laferriere, M. L. Keller, and R. Pausch. An introductory tutorial for developing multi-user virtual environments. *Presence : Teleoperators and Virtual Environments*, 3(4) :255–264, 1994.

[11] C. Gutwin and S. Greenberg. The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Trans. Comput.-Hum. Interact.*, 6(3) :243–281, 1999.

[12] M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Computer Animation and Simulation '98*, pages 73–86, 1998.

[13] X. Larrodé, B. Chanclou, L. Aguerreche, and B. Arnaldi. Openmask : an open-source platform for virtual reality. In *IEEE VR workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*, Reno, NV, USA, 2008.

[14] D. Margery, B. Arnaldi, A. Chauffaut, S. Donikian, and T. Duval. Openmask : Multi-threaded | Modular animation and simulation Kernel | Kit : un bref survol. In *VRIC 2002, Laval, France*. S. Richir, P. Richard, B. Taravel, juin 2002.

[15] M. R. Mine, F. P. Brooks, Jr., and C. H. Sequin. Moving objects in space : Exploiting proprioception in virtual-environment interaction. *Computer Graphics*, 31(Annual Conference Series) :19–26, 1997.

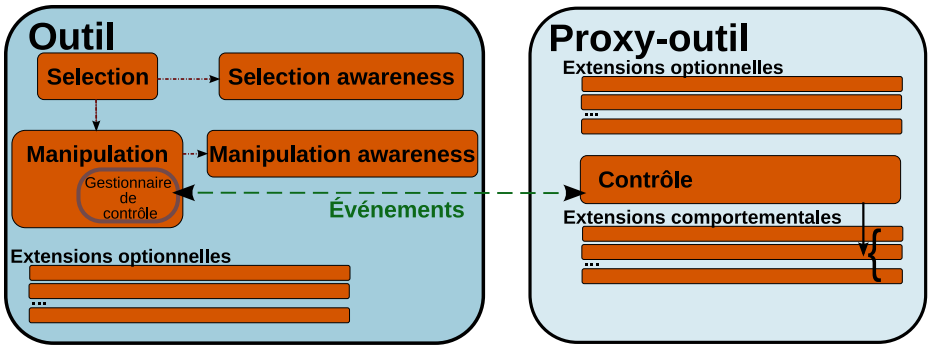

FIG. 7: Vue schématique des extensions d'un outil et d'un proxy-outil, ainsi que de la communication entre un outil et un proxy-outil.

[16] M. S. Pinho, D. A. Bowman, and C. M. Freitas. Cooperative object manipulation in immersive virtual environments : Framework and techniques. In *VRST '02 : Proceedings of the ACM symposium on Virtual reality software and technology*, pages 171–178. ACM Press, 2002.

[17] I. Poupyrev, M. Billinghurst, S. Weghorst, and T. Ichikawa. The go-go interaction technique : Non-linear mapping for direct manipulation in vr. In *Proceedings of the 9th annual ACM symposium UIST*, pages 79–80. ACM Press, 1996.

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

[19] K. Riege, T. Holtkamper, G. Wesche, and B. Frohlich. The bent pick ray : An extended pointing technique for multi-user interaction. In *3DUI '06 : Proceedings of the 3D User Interfaces (3DUI'06)*, pages 62–65. IEEE Computer Society, 2006.

[20] P. Sequeira, M. Vala, and A. Paiva. What can I do with this ? : Finding possible interactions between characters and objects. In *AAMAS '07 : Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, pages 1–7. ACM, 2007.

[21] J. Sreng, A. Lécuyer, C. Mégard, and C. Andriot. Using visual cues of contact to improve interactive manipulation of virtual objects in industrial assembly/maintenance simulations. *IEEE TVCG*, 12(5) :1013–1020, 2006.