Journées AFIG - Bordeaux (2006)
J.-C. Gonzato, J.-P. Jessel (Editeurs)

Utilisation des composants pour la représentation des objets virtuels

S. Gaeremynck[1], S. Degrande[1], L. Grisoni[1] et C. Chaillou[1]

[1]Institut d'Informatique Fondamentale de Lille

Abstract

Les applications 3D interactives sont de plus en plus répandues. Mais, de par leur complexité, seuls des spécialistes peuvent les créer. Dans cet article, nous proposons de faciliter le processus créatif grâce à un intergiciel traitant les objets virtuels comme des composants. Ces composants, adaptés aux applications 3D interactives, sont appelés composants 3D. Ils facilitent la mise au point des objets virtuels et leur réutilisation pour la création d'applications 3D. Dans notre proposition, les composants 3D ont un modèle d'exécution spécifique qui permet aux auteurs d'implémenter sans contrainte tout type de comportement. Un modèle conceptuel de composant 3D et le modèle d'exécution correspondant sont présentés. La mise en œuvre de ces concepts dans le cadre du projet Spin3D est également détaillée.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Three-Dimensional Graphics and Realism]: Virtual reality

1. Introduction

Les applications 3D interactives sont actuellement en pleine expansion aussi bien du point de vue de la recherche que des applications industrielles. Cet engouement est une conséquence naturelle des gains en puissance des ordinateurs personnels. De nos jours, des environnements virtuels complexes peuvent en effet être affichés sur la plupart des ordinateurs. Cependant, la mise au point d'applications 3D, c'est-à-dire d'applications qui utilisent la 3D pour afficher des informations et interagir avec l'utilisateur, est encore une tâche difficile. D'une part, cette activité nécessite des compétences poussées en mathématiques, physique, programmation, graphisme, etc. D'autre part, il n'existe pas à l'heure actuelle d'intergiciel standard dédié à la création d'applications 3D, à la manière de ceux permettant de développer des applications Web (J2EE, .NET).

Afin de faciliter la mise au point d'applications 3D, plusieurs projets (3D Beans, I4D, ...) ont proposé un environnement où les objets virtuels sont implémentés par des composants. Ces approches ont pour point commun de vouloir améliorer la productivité dans le domaine particulier des applications 3D interactives, en utilisant les objets virtuels en tant que boîtes noires et la composition par des tiers. Ces projets ainsi que les nombreux formats de fichiers permettant de représenter les données 3D (VRML, X3D, 3DS) se concentrent principalement sur l'aspect visuel et l'animation des objets virtuels et négligent les autres types d'informations tel que leur comportement. Le comportement des objets sera ainsi spécifique à la plate-forme d'exécution et les utilisateurs seront limités lors de la mise au point de leurs objets virtuels.

Plus généralement, peu de travaux tels que GASP ont été menés afin de mettre au point un intergiciel générique dont le modèle d'exécution n'impose pas de limitations au comportement des objets virtuels. Dans cet article, nous présentons des modèles de composants 3D et d'exécution qui autorisent la cohabitation d'objets virtuels hétérogènes au sein d'un même environnement virtuel, notamment pour faciliter l'intégration de comportements régis par les lois de la physique. Ces deux modèles combinés peuvent servir à la représentation d'objets virtuels autonomes qui correspondent aux objets réels. Le but final est de permettre aux utilisateurs de créer une scène 3D de la même manière qu'ils aménageraient une pièce, c'est-à-dire à partir d'un assemblage d'objets virtuels de natures différentes.

Nous expliquons comment les composants peuvent être utilisés dans le développement d'environnements virtuels

et nous montrons leur potentiel dans le développement d'applications 3D.

2. Approches existantes à base de composants

Un certain nombre de projets utilisent déjà les composants pour représenter les objets virtuels. Ils s'appuient tous sur les avantages de l'approche composant, comme la réutilisation et la composition par des tiers, afin de faciliter la production d'applications 3D.

2.1. X3D

X3D [Web] est un format de graphe de scène autorisant la création de prototypes qui jouent un rôle analogue à celui tenu par les procédures en programmation classique. Un prototype est un graphe de scène local représentant un objet virtuel. Il possède une interface publique regroupant les champs représentatifs de l'état de l'objet. Cette interface permet de réutiliser un prototype sans avoir de connaissances préalables sur son implémentation.

X3D peut être considéré comme une approche à base de composants étant donné que les prototypes correspondent à la définition des composants donnée dans [Szy02] : ce sont des unités de déploiement indépendantes, qui peuvent être composées par des tiers et qui n'ont pas d'état observable.

Cependant, X3D est un format réservé à la création d'objets graphiques et ne peut pas servir à la création d'objets applicatifs arbitraires. Son but n'est pas de permettre la définition et la composition de comportements autres que ceux relatifs à l'animation ou à la représentation (visuelle, sonore, etc.) de l'objet. De plus, un prototype ne peut partager que ses champs lors d'une composition. Il ne peut donc pas fournir d'accès aux comportements de l'objet ou utiliser les comportements implémentés par d'autres objets.

2.2. 3D Beans

3D Beans [DG00] est un modèle de composants basé sur les technologies JavaBeans et Java3D. Il tire ainsi parti des fonctionnalités offertes par les JavaBeans telles que l'introspection, la sérialisation et la composition. Un 3D Bean est en fait constitué de deux sous-entités : un bean classique implémente le comportement du composant, la façon dont il réagit aux événements, tandis qu'un autre bean se charge de tous les aspects graphiques du composant.

Un environnement de création, appelé 3D BeanBox, peut être utilisé à la fois pour créer un graphe de scène à partir de 3D Beans et spécifier la gestion des événements par les Beans. Les applications construites avec cet environnement logiciel sont essentiellement réactives vis-à-vis des entrées utilisateur. Il est néanmoins possible d'utiliser des événements basés sur une horloge afin de mettre à jour des objets à intervalles réguliers et jouer automatiquement des animations.

3D Beans est cependant limité par l'utilisation de Java3D, notamment en ce qui concerne les dispositifs d'interactions, seuls la souris et le clavier étant supportés. De plus, il n'est pas clair que la technologie JavaBeans et les performances de la JVM soient adaptées à l'implémentation d'applications gourmandes en mémoire et en temps de calcul comme les applications 3D interactives.

2.3. I4D

I4D [GPRR01] est un modèle de composants développé dans le contexte de la création d'applications 3D interactives et augmentée. Dans ce modèle, les composants, appelés acteurs, peuvent être des sources de lumière, des caméras, des objets virtuels ou des éléments logiciels sans représentation visuelle. Les acteurs peuvent être organisés hiérarchiquement de manière à former des scènes complexes.

Seuls les comportements relatifs à l'animation des acteurs sont décrits dans I4D. Ces comportements sont créés en utilisant des composants spécialisés, prédéfinis ou développés par des tiers, qui modifient en permanence les champs des acteurs. [GPRR01] fournit un exemple d'animation où un composant modifie la position des articulations des pattes d'une araignée. Il n'est cependant pas précisé dans [GPRR01] si les acteurs peuvent implémenter d'autres types de comportements.

2.4. CONTIGRA

CONTIGRA [DHM02] est une approche orientée document pour la spécification de scènes 3D basée sur XML. Les documents CONTIGRA contiennent des données qui décrivent les composants et la façon dont ils doivent être combinés. Des animations peuvent donc être créées sans avoir recours à des connaissances en programmation, par l'utilisation de composants spécialisés prédéfinis et d'une méthode de composition particulière.

CONTIGRA se concentre uniquement sur la description des composants. Ainsi, il n'est donné aucun détail sur l'environnement d'exécution, le modèle de composants et ses limitations. En particulier, aucune précision n'est apportée quant à la façon dont se comporte la plate-forme dans la gestion de scènes complexes ou quant à la possibilité de définir des comportements autres que ceux relatifs à l'animation.

2.5. GASP

GASP [DM00] propose une approche plus générique de la définition des comportements. Il s'agit d'une plate-forme dont le but est d'animer des agents autonomes ou dirigés par des utilisateurs. Les objets virtuels, appelés entités, y sont définis comme des agrégats de comportements. Chaque comportement possède une interface publique composée

d'entrées (respectivement de sorties) qui peuvent être connectées aux sorties (respectivement aux entrées) d'autres comportements. La représentation visuelle d'une entité est ainsi implémentée par un comportement spécialisé accessible via une interface.

GASP est très flexible car il n'impose pas de contrainte sur les types de comportements qu'une entité peut implémenter. Il a toutefois quelques limitations. Tout d'abord, les interfaces ne sont composées que de champs disponibles en écriture ou en lecture et ne permettent pas d'appeler un service implémenté par une entité. Ensuite, un objet virtuel n'a pas d'indépendance de déploiement et ne peut pas être utilisé comme une unité par des tiers.

3. Notre proposition: les composants 3D

Toutes les approches présentées jusqu'ici utilisent principalement les composants pour faciliter la création d'animations. En particulier, elles ne profitent pas ou peu du fait que le développement à base de composants n'impose pas de limite sur les comportements pour gérer des aspects des applications 3D interactives autres que l'animation. Nous introduisons ici la notion de composants 3D, et nous détaillons leurs spécificités par rapport à l'approche classique décrite dans [Szy02] ainsi que leur apport dans le cadre de la conception d'applications 3D interactives. Dans cette section nous décrivons, tout d'abord, l'implémentation d'un composant 3D puis ses propriétés structurelles.

3.1. Définition

Au cours du développement d'un objet virtuel, deux aspects sont à prendre en compte : sa représentation et ses comportements. Cette séparation est similaire à celle qui existe pour les applications Web et les interfaces graphiques. Dans ces applications, un modèle de conception couramment utilisé est l'architecture *Modèle-Vue-Contrôleur* (MVC) [Ree79]. Dans le contexte d'applications 3D interactives, cette approche peut se dériver ainsi:

- La géométrie, qui spécifie la forme de l'objet (son aspect visuel, ses couleurs, etc.), correspond à la *Vue*. Il est à noter qu'un objet peut définir plusieurs géométries mais dans ce cas, une seule sera affichée à un instant donné. Ceci permet à un objet de changer de représentation lorsque l'utilisateur le manipule par exemple.
- Les comportements définissent la façon dont l'objet évolue, interagit avec les autres et crée des animations. Il s'agit du *Modèle*.
- Dans cette approche, un objet virtuel est alors implémenté en agrégeant ses différents aspects (représentation et comportements) dans une entité, appelée composant 3D, faisant office de *Contrôleur*.

3.2. Concept d'interface

Pour que les composants 3D soient utilisables et paramétrables par des tiers, ils doivent partager des informations concernant leur représentation et leurs comportements au travers d'interfaces nommées:

- Les informations relatives à la représentation sont contenues dans les champs issus de la partie géométrique du composant. Elles peuvent, par exemple, permettre de publier la couleur de l'objet virtuel.
- Les informations relatives aux comportements sont fournies par des attributs ou des points d'accès aux services implémentés par les comportementsă: un service de base de données par exemple.

Un composant 3D, qui veut publier une information, le fera au travers d'une interface dite *fournie*. De la même façon, un composant qui a besoin d'une information publiée par un autre composant, devra déclarer une interface qui sera dite *requise*. Les liaisons entre interfaces fournies et requises peuvent être réalisées de manière:

- implicite : dans ce cas, la destination accède directement à la source sans utiliser de connecteur et aucun traitement supplémentaire n'est effectué lors de l'échange d'informations.
- explicite : dans ce cas, un connecteur (un composant passif sans représentation) est effectivement utilisé pour modifier les informations échangées entre la source et la destination. Ce mécanisme est notamment utile pour adapter le format des données lors du dialogue entre deux composants.

3.3. Composition

A l'instar des composants classiques, les composants 3D sont généralement développés indépendamment les uns des autres. Les applications 3D sont alors créées à l'aide d'une méthode de composition spécifique qui tire partie de la séparation entre vue et modèle.

3.3.1. Graphes de vue

Un des principaux avantages des graphes de scène est de permettre l'organisation spatiale des objets virtuels les uns par rapport aux autres. Afin de transposer ce type de structure à notre approche, nous définissons les composites comme les composants 3D dont la représentation est la racine d'une arborescence de représentations d'autres composants 3D. L'arbre de composites permet alors d'organiser les composants les uns par rapport aux autres, à condition de positionner chaque composant fils par rapport à un repère local défini par son composite parent. Cet arbre est appelé *graphe de vue*.

Cette définition permet de décomposer un objet virtuel en parties facilement reconnaissables car correspondant à celles de l'objet réel. Par exemple, un appareil photo est composé

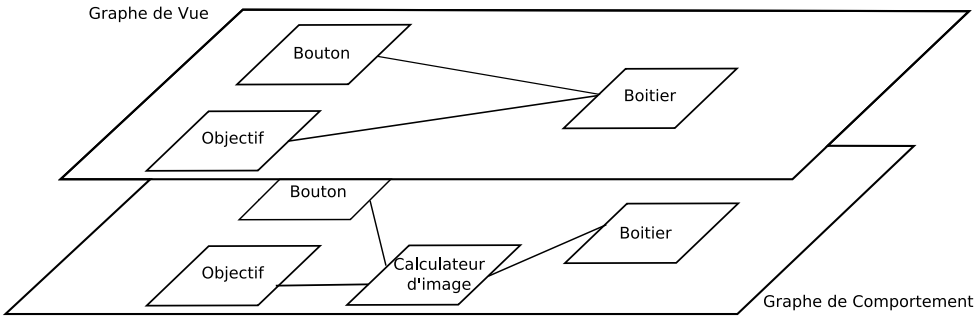

Figure 1: *Exemple de graphes de vue et de comportement*

d'un boîtier, d'un ensemble de boutons (au moins un pour prendre la photo) et d'un objectif. Dans l'environnement virtuel, chaque objet réel ayant son équivalent virtuel, le corps de l'appareil photo pourrait être un composite dont les fils seraient les boutons et l'objectif.

Par définition, un *graphe de vue* est l'équivalent d'un graphe de scène. La principale différence se situe en fait dans la brique de base utilisée pour la construction de l'arbre. Pour les graphes de scène, la brique est un nœud élémentaire comme un cube, une sphère, alors que dans le cas des composants 3D, il s'agit de l'objet virtuel. Ceci entraîne une différence au niveau des traitements effectués sur la structure arborescente. Ils ne sont plus réalisés par appels récursifs sur les méthodes des nœuds du graphe mais par l'invocation de services fournis par les composants 3D. Par exemple, un composite est affiché en appelant son service de rendu qui invoquera à son tour les services de rendu de ses composants fils.

3.3.2. Graphes de comportement

Le graphe de vue permet de créer une relation spatiale entre les objets virtuels. Cependant, il ne permet pas de définir de relation entre les comportements des composants 3D. Pour ce faire, les composants peuvent être connectés par l'intermédiaire des interfaces qu'ils implémentent ou qu'ils requièrent pour fonctionner. Ces connexions définissent alors ce qu'il est possible d'appeler un *graphe de comportement*.

Ce processus de connexion peut être utilisé pour créer des objets virtuels exhibant des comportements complexes à partir d'un ensemble de composants ayant des comportements élémentaires. Dans le cas de l'appareil photo mentionné précédemment, le boîtier pourrait implémenter un service enregistrant la photo calculée par un composant *Calculateur d'image* en fonction des réglages de l'objectif. Ce service serait alors appelé lorsque le bouton de l'appareil est actionné par l'utilisateur.

La création d'un graphe de vue et d'un graphe de comportement, permet donc de créer des objets virtuels et des applications par un processus d'assemblage, aussi appelé composition. Le graphe de vue permet de positionner les objets virtuels les uns par rapport aux autres tandis que le graphe de comportement leur permet de collaborer pour l'exécution du monde virtuel. La figure 1 montre les relations établies par les deux graphes dans le cas de l'appareil photo donné en exemple. Il est à noter que le composant *Calculateur d'image* n'a pas de représentation graphique.

4. Environnement d'exécution

Si les graphes de comportement décrivent la manière dont les composants interagissent les uns avec les autres, cette information n'est cependant pas suffisante pour simuler l'évolution de la scène: certains composants peuvent en effet nécessiter une mise à jour en dehors de toute interaction.

4.1. Composants actifs et passifs

Afin de pouvoir gérer ce type de mise à jour, il est nécessaire de distinguer deux familles de composants 3D:

- Un composant est dit actif lorsqu'il possède un ou plusieurs comportements actifs qui évoluent sans intervention de l'utilisateur : un objet virtuel mettant à jour sa position en fonction des lois de la physique (pesanteur, etc.) en est un exemple. Pour que le monde virtuel soit réaliste, l'environnement d'exécution doit donc autoriser ces composants à se mettre à jour de façon autonome.
- Lorsqu'il n'évolue qu'en réaction à un stimulus, un composant est dit passif : c'est généralement le cas des murs d'une pièce, de certains composants dédiés à l'accès de services ou des connecteurs.

4.2. Exécution des comportements actifs

Une application fonctionnant en temps réel doit mettre à jour sa zone d'affichage environ toutes les 40ms (ce qui correspond à 25Hz). Le calcul d'une nouvelle image doit donc être plus rapide. Cependant, les comportements des objets virtuels peuvent être gourmands en temps de calcul. Pour ne

pas gêner l'interactivité avec le monde virtuel, il convient donc d'identifier les comportements dont l'exécution est indispensable à la génération de chaque image afin de les exécuter en priorité.

En pratique, ceci peut se faire au moyen de deux files d'attente mémorisant les comportements en attente d'exécution :

- La première file d'attente contient tous les comportements qui doivent impérativement être mis à jour lors de la génération de chaque image. C'est le cas par exemple de la position d'un outil d'interaction piloté par la souris. Celui-ci met à jour sa position en synchronisation avec l'affichage afin que l'utilisateur ait une bonne perception de son interaction avec le monde virtuel.
- La deuxième file d'attente contient tous les autres comportements. Les tâches gourmandes en temps de calcul, tels que la simulation physique, utiliseront typiquement cette file afin de ne pas gêner les interactions entre l'utilisateur et le monde virtuel.

Lors de la génération d'une image, la première file d'attente est toujours exécutée intégralement tandis que les comportements de la deuxième sont exécutés de manière séquentielle tant que les 40 ms ne sont pas écoulées. Dans le cas où la durée d'exécution de la première file est supérieure au temps alloué pour la génération de l'image, la mise à jour de la zone d'affichage est différée au risque de nuire à l'interactivité.

Ce mécanisme en apparence simple à implémenter recèle néanmoins une difficulté. Les connexions entre composants peuvent en effet induire un ordre sous-jacent dans l'exécution des comportements du fait des interdépendances entre interfaces. Pour tenir compte de cet ordre, les comportements doivent avoir la capacité de suspendre leur exécution en attendant la mise à jour de leurs antécédents.

La figure 2 illustre le potentiel de cette approche dans le cas de la modélisation d'un appareil photo. La vignette correspondant au viseur de l'appareil, affichée en bas à gauche de la figure, est générée par la représentation d'un composant 3D appelé *Viseur*. Au moment de l'exécution, ce composant enregistre dans l'ordonnanceur une tâche qui recalcule périodiquement la scène à photographier à partir du composant 3D représentant l'appareil photo. Il est ainsi possible de voir le skieur représenté sur la vignette se déplacer. Il est également possible de manipuler l'appareil photo afin de modifier dynamiquement sa profondeur de champ et donc la netteté des détails sur la vignette.

5. Mise en œuvre: Spin3D

Nous avons développé un intergiciel qui implémente le modèle de composants 3D décrit précédemment. Sa mise en œuvre a été réalisée dans le cadre de la troisième version du projet Spin3D [Spi]. En effet, la version actuelle, *Spin3D-v2*, n'utilise pas de modèle à base de composants.

5.1. Présentation

Spin3D est un projet en cours de développement au sein de l'équipe GRAPHIX du LIFL. Il vise à créer un environnement virtuel collaboratif synchrone pour des réunions en petit groupe dans des situations d'apprentissage à distance ou de conception collaborative.

Spin3D possède une architecture ouverte : son but est de fournir une plate-forme générique afin de faciliter le développement des applications collaboratives en 3D. Avec Spin3D, les développeurs se concentrent uniquement sur la conception de leur application et leur domaine métier sans se préoccuper des détails d'implémentation relatifs à l'affichage, aux interactions et au partage sur le réseau.

Figure 2: *Exécution de l'application "Appareil photo" dans Spin3D*

En pratique, Spin3D utilise la métaphore de la table de conférence, c'est-à-dire que tous les participants sont réunis autour d'une table virtuelle sur laquelle sont posés les objets utilisés au cours de la réunion. La plate-forme donne aussi un retour sur l'activité des autres utilisateurs ainsi que sur les possibilités d'interaction au sein de l'environnement virtuel. Pour ce faire, tous les objets virtuels disponibles sont affichés en permanence ainsi que les autres utilisateurs connectés.

5.2. Les composants Spin3D

Pour implémenter les différents types d'objets et les faire cohabiter dans le même environnement virtuel, Spin3D utilise les composants 3D décrits précédemment. La figure 3 montre un schéma UML du modèle adopté. Une classe modèle *TComponent* définit le squelette de base du composant. Cette classe prend en paramètre le type de composant,ă le type de comportement et le type de représentation:

- Le type de composant est implémenté par la classe *Composite* si le composant peut avoir des enfants, ou par la classe *Component* dans le cas contraire.
- L'ajout de nouveaux comportements se fait par dérivation de la classe *Behavior*.
- Le format des fichiers graphiques qui servent à la représentation est encapsulé dans la classe *View*. Il

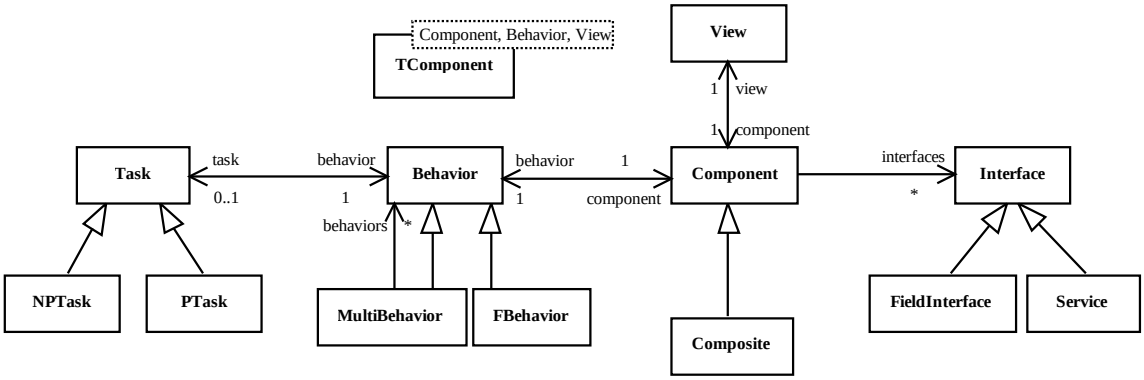

Figure 3: *Schéma UML du framework de composants*

est possible d'ajouter un traitement spécifique sur la représentation par dérivation de cette classe. Par exemple, un composite peut spécialiser sa vue afin de gérer la répartition spatiale de ses fils.

Les interfaces d'un composant peuvent être de l'un des deux types suivants :

- Les interfaces exposant les champs de la représentation ou les attributs des comportements sont implémentées par le type *FieldInterface*.
- Les interfaces exposant les services fournis par le composant sont implémentées par dérivation du type *Service*.

5.3. Exécution des comportements

Les comportements que la plate-forme exécute sont enregistrés dans deux files d'attente distinctes. Une entité de type *Task* se charge de cet enregistrement pendant l'initialisation du comportement auquel elle est associée. Lors du rendu de l'environnement virtuel, les comportements sont ensuite récupérés par des *threads* afin d'être exécutés.

Comme indiqué précédemment, ces comportements doivent également avoir la capacité de suspendre leur exécution lorsqu'ils ont des antécédents. Cette flexibilité a un coût important en terme de mémoire, du fait qu'une pile d'exécution spécifique doit être allouée pour chaque comportement susceptible d'être suspendu. A des fins d'optimisation, Spin3D distingue deux types de tâches afin de spécifier le mode d'exécution de chaque objet virtuel :

- Celles qui peuvent être suspendues: *PTask* (le préfixe P signifie *Preempted*)
- Celles qui ne peuvent pas être suspendues: *NPTask*

Une décision importante concerne le choix de la file d'attente où enregistrer chacun des comportements. Cette répartition a pour objectif de trouver un juste compromis entre l'interactivité de l'environnement virtuel et l'exécution complète de la scène. Dans les cas où le temps d'exécution de la première file d'attente est supérieur au temps alloué à la génération d'une image, le développeur peut également forcer une exécution partielle de la seconde file d'attente à intervalles réguliers.

5.4. Création d'applications

5.4.1. Création d'un composant

Lors de la création d'un nouveau composant, le développeur doit déterminer si un comportement existant peut être utilisé ou si un nouveau comportement doit être programmé. La réutilisation d'un comportement existant se fait à l'aide de la classe *FBehavior* (le préfixe F dénote une *Factory*) qui permet d'instancier un comportement se trouvant dans une librairie. A cet effet, des librairies d'interactions et d'animations physiques sont en cours de développement. Lorsque des comportements plus spécifiques sont requis, la programmation est néanmoins obligatoire, le développeur devant alors les créer en C++. Enfin, la classe *MultiBehavior* permet à un composant de posséder plusieurs comportements.

Un autre choix intervenant au moment de la création d'un composant porte sur la manière dont celui-ci va gérer sa représentation. Lorsqu'aucune gestion particulière n'est requise, la plate-forme utilisera une spécialisation de la classe *View* adaptée au format du fichier graphique. Dans le cas des composites, le développeur doit explicitement spécialiser la classe *View* afin de gérer la répartition spatiale des composants fils selon ses besoins.

Afin de faciliter la réutilisation des comportements et profiter des formats graphiques supportés par Spin3D, des prototypes de composants sont inclus dans la plate-forme. De cette manière, la création d'un nouvel objet virtuel se résume au développement de sa représentation.

5.4.2. Description XML

```
<Application>
 <Component name = "System">

  <Component name = "SelectAndManip">
     <Interface name = "Translation" value = "0.0 0.0 1.0"/>
  </Component>

  <Component name = "Decor">
     <Interface name = "Width" value = "800"/>
     <Interface name = "Height" value = "600"/>
     <Interface name = "Texture" value = "mur3.jpg"/>
  </Component>

  <Component name = "Table">
     <Interface name = "Translation" value = "0.0 0.0 0.0"/>
     <Interface name = "Texture" value = "table.jpg"/>
        <Component name = "Pentax">
        </Component>
  </Component>

 </Component>

 <Bind from = "System.Select" to = "SelectAndManip.Select"/>
</Application>
```

Figure 4: *Exemple de déclaration d'une application*

```
<SpinComponent name="SelectAndManip" factory = "SelectAndManip">

 <View name = "defaultPointer.x3d">
  <!-- Position de l'objet dans l'environnement virtuel -->
  <Field name = "Translation" node = "BaseTransform"
                              field = "translation" shared = "false"/>
 </View>

 <!-- Required Interfaces -->
 <Interface name = "Select"/>
</SpinComponent>
```

Figure 5: *Exemple de déclaration d'un composant*

Comme montré en figure 4 et 5, les applications et les composants sont déclarés dans des documents XML. Ces fichiers fournissent à la plate-forme les informations nécessaires au chargement des composants et détaillent leur configuration. La figure 5 fournit un exemple de déclaration de composant dont le type est *Custom*, ce qui signifie que le composant est programmé sans utiliser l'un des prototypes proposés par la plate-forme. La figure 4 décrit la structure de l'application: le fichier détaille d'abord la hiérarchie des composants puis les relations entre interfaces. La figure 6 montre le résultat de l'interprétation des fichiers de configuration.

6. Comparaison

Dans cette partie, nous effectuons une comparaison entre notre approche, par composants 3D, et le développement classique par graphe de scène. Nous prenons pour exemple le cas simple d'une balle simulée selon les lois de la physique.

Le composant *Ball* est présenté à la figure 7.

Ce composant expose un comportement et quatre propriétés. Le comportement correspond à la mise à jour du

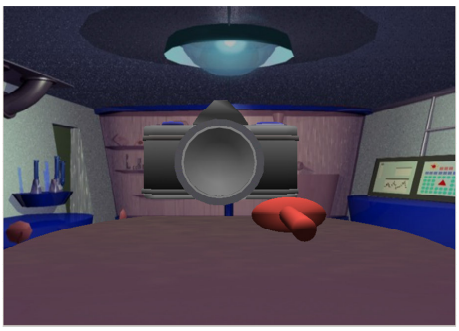

Figure 6: *Exécution de l'application*

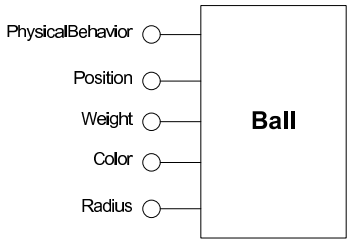

Figure 7: *Un composant représentant un corps rigide*

corps rigide. Les propriétés concernent le paramétrage du poids, de la couleur, de la position et du rayon de la balle. Ce type de composant est ce que manipule directement un utilisateur de la plate-forme.

La figure 8 montre la même application sous forme de graphe de scène.

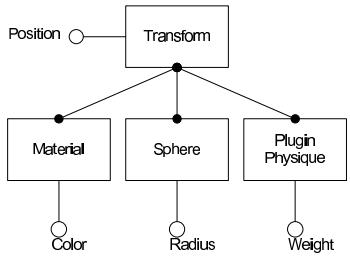

Figure 8: *Un graphe de scène représentant un corps rigide*

Le graphe de scène est composé de 4 nœuds, qui représentent la position de la balle dans le monde, sa couleur, son rayon et ses propriétés physiques. Les propriétés physiques sont gérées par l'intermédiaire d'un plugin.

Pour un utilisateur final de l'objet virtuel, les avantages de l'approche par composants 3D sont les suivants:

- Les implémentations des comportements et des propriétés graphiques d'un objet virtuel sont séparées.

- La présentation des propriétés et des comportements de l'objet virtuel est uniformisée au sein d'une même entité logicielle.

7. Conclusion et perspectives

Pour répondre à la complexité du développement d'applications 3D interactives, nous avons présenté notre modèle basé sur des composants 3D, sa structure et son mode d'exécution. Ce modèle permet notamment de considérer les objets virtuels comme entités autonomes et n'impose pas de contrainte sur le type des comportements que les objets virtuels peuvent implémenter. De plus, la séparation nette en représentation et comportements permet de répartir la charge du développement d'un objet virtuel entre les spécialistes de chaque discipline.

Notre modèle semble adéquat dans le cadre du développement des applications 3D interactives. Certains points restent toutefois ouverts:

- Les tests ont jusqu'ici été réalisés sur de petits exemples. Nous adaptons actuellement des applications plus complexes afin de vérifier le comportement de notre proposition sur ces environnements.
- La composition proposée actuellement permet un assemblage statique. Cependant, dans des cas complexes tels que la conception mécanique, il peut être nécessaire de créer dynamiquement des assemblages. Le problème est alors de gérer correctement la sémantique associée à l'assemblage des objets virtuels.

References

[DG00] DOERNER R., GRIMM P.: Three-dimensional beans : Creating web content using 3d components in a 3d authoring environment. Proceedings of the Web3D-VRML 2000, fifth symposium on Virtual reality modeling language, Monterey CA.

[DHM02] DACHSELT R., HINZ M., MEISSNER K.: Contriga : An xml-based architecture for component-oriented 3d applications. *ACM Web3D Symposium* (2002).

[DM00] DUVAL T., MARGERY D.: Using gasp for collaborative interaction within 3d virtual worlds. Proceedings of the Second International Conference on Virtual Worlds.

[GPRR01] GEIGER C., PAELKE V., REIMANN C., ROSENBACH W.: Structured design of interactive virtual and augmented reality content. International Workshop on Structured Design of Virtual Environments and 3D-Components at the Web3D 2001 Conference, Paderborn Allemagne.

[Ree79] REENSKAUG T.: Models - views - controllers. Technical note, Xerox PARC, Dec. 1979.

[Spi] Spin3d. `http://www.lifl.fr/GRAPHIX/collaborations.php`.

[Szy02] SZYPERSKI C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Westley Longman, 2002.

[Web] `http://www.web3D.org`.