Adaptation dynamique de modalités d'interaction 3D par règles de filtrage

Nicolas Martin, Samuel Degrande, Christophe Chaillou

Adaptation dynamique de modalités d'interaction 3D par règles de filtrage

Nicolas Martin* Samuel Degrande† Christophe Chaillou‡

Alcove-INRIA Futurs, LIFL (UMR CNRS 8022), Université Lille I.

ABSTRACT

Cet article décrit une méthode originale qui permet de configurer le comportement interactif d'une application 3D en se basant sur un mécanisme de règles d'interaction. Le comportement interactif de l'application finale est défini par les outils d'interaction et les différents éléments de l'interface, les règles permettant de spécifier au système comment les composants interagissent entre eux, en fonction du contexte de l'application.

Keywords: MVC, Outils d'interaction, Règles d'interaction.

Index Terms: H.5.2 [Users Interfaces]: User Interface; D.2.11 [Software Engineering]: Software Architecture

1 INTRODUCTION

Aujourd'hui les interfaces utilisateurs deviennent de plus en plus complexes tant au niveau visuel - richesse graphique, animation, 3D -, qu'interactif - par la grande diversité de périphériques d'entrée et la complexité des tâches à réaliser - . Chacun de ces différents points fait l'objet de nombreux travaux de recherches. En effet, les besoins changent mais la majorité des APIs de développement utilisées restent basées sur le développement d'applications statiques. Les boîtes à outils actuelles sont basées sur la notion de widgets et se limitent au support de techniques d'interaction usuelles simples. La définition de modalités d'interactions évoluées demande souvent des efforts d'implémentation ad hoc. Dans le cas particulier des environnements 3D lorsque les périphériques utilisateur ne se limitent plus à une souris et un clavier, et que les actions de l'utilisateur ne se limitent plus à pointer, cliquer et déplacer, la gestion des interactions est encore plus difficile. Les outils de conception doivent prendre en compte la gestion des interactions au moins au même niveau que les éléments d'interfaces. Les systèmes postWimp vont dans ce sens et rompent avec le modèle de conception classique en mettant en amont la question des techniques d'interactions non usuelles.

Mais s'il on peut concevoir l'interaction en tant que bloc indépendant donc potentiellement réutilisable, il nous semble interessant de pouvoir adapter, à posteriori, ce bloc à l'environnement dans lequel il va être intégré.

La modalité d'interaction définie pour un document n'est pas forcément la même selon le contexte de l'application ou selon la configuration de l'interface. Par exemple dans le cas d'une activité collaborative, selon qu'un document soit dans une partie privée ou publique de l'interface.

Dans le cadre de nos travaux de recherches, nous développons un gestionnaire de bureau 3D permettant la création d'applications 3D collaboratives non immersives. De part sa nature virtuelle, le projet s'inscrit dans le contexte des interfaces post-Wimp [15]. L'architecture de base s'appuie sur le patron de conception Modèle-Vue-Contrôleur [8] et l'utilisation d'outils d'interactions. Dans cet article, nous présentons un modèle d'interaction à base d'outils d'interaction dans un environnement virtuel configurable et capable d'adapter les modalités d'interactions. Ce modèle permet la description et le contrôle des modalités d'interaction de haut niveau. La description se veut intuitive et non réservée à un développeur. Nos travaux se placent dans le cadre des interfaces 3D, mais les concepts peuvent être repris simplement dans le contexte des interfaces 2D.

Dans la suite de cet article nous commencerons par présenter le contexte dans lequel nous nous plaçons et notre problématique. Nous détaillons ensuite nos propositions et présentons quelques aspects d'implémentations. Enfin nous conclurons et présenterons les perspectives de nos travaux.

*e-mail: nicolas.martin@lifl.fr

†e-mail: samuel.degrande@lifl.fr

‡e-mail: christophe.chaillou@lifl.fr

2 CONTEXTE ET PROBLÉMATIQUES

On peut catégoriser plusieurs techniques de réalisation d'interfaces utilisateur. Les premières utilisent des langages de programmation évolués comme le C++ ou Java à l'aide d'API spécifiques (MFC, Swing, ...) permettant au concepteur de l'application d'avoir un contrôle plus fin sur le système tout en ayant accès à un ensemble de briques d'interfaces de bases. Les secondes regroupent les UIMS, qui reposent sur des langages descriptifs d'interface comme UsiXML [14] ou Xul. L'approche descriptive à l'avantage de permettre une conception rapide et plus accessible parfois agrémentés d'outils auteurs graphiques. Les applications restent néanmoins dépendantes du moteur les interprétant, même si ces derniers proposent des mécanismes d'extensions. Notre approche consiste à mêler les deux systèmes et tente d'allier liberté et facilité de conception.

2.1 Contraintes des environnements 3D

En effet dans cadre nouveaux des applications 3D il n'est pas possible de fournir comme en 2D l'ensemble de briques de conception de bases (menus, listes, fenêtres). Principalement parce-qu'il n'existe pas de cadre définissant les éléments bases *génériques* ou utiles à une application 3D [5] [13] ou même la façon de transposer les éléments du monde 2D en 3D. La définition même d'un widget (*window gadget*) peut sembler hors de propos dans le cas d'une scène tri-dimentionnelle interactive qui ne s'appuie plus sur la métaphore de la fenêtre. S'il semble trivial par exemple de pouvoir disposer d'éléments usuels comme des boutons, par la nature même de la 3D, les applications ont la possibilité de calquer le monde réel/virtuel. Selon le contexte applicatif ou les périphériques utilisés, même si sémantiquement le rôle du bouton reste de déclencher une action, sa représentation et les modalités d'interactions nécessaires à son activation seront différentes.

D'une manière plus générale il n'existe pas de modèle comme le Wimp adapté pour les systèmes virutels. Ce constat s'appuit sur deux points de vue :

- Les périphériques utilisés vont être très variés selon l'application, allant du simple couple clavier/souris que l'on retrouve dans la majorité des jeux vidéos, à des périphériques plus spécifiques comme les gants de données, du *head tracking*, etc...

Cette diversité est du à la complexité potentielle de la tâche à réaliser dans l'environnement virtuel qui peut demander un grand nombre de degrés de libertés en entrée.

Le choix des périphériques devient plus délicat lorsque les tâches à réaliser sont hétérogènes. Si un Phantom est adapté à la tâche de sélection d'objets 3D, il l'est moins qu'une souris 2D pour la sélection de texte dans un document. Le choix des périphériques doit se faire en faisant des compromis entre toutes les tâches à réaliser. Ceci implique la mise en place de techniques d'interaction particulières pour permettre de réaliser la tâche avec un périphérique moins adapté (par exemple le *trackball* ou le *ray-casting* qui permettent des interactions pour la sélection et la manipulation d'objets 3D avec un simple périphérique 2D).

- Le choix de représentation et les métaphores utilisées ne répondent pas non plus à des règles fixées. Si le choix de représentation se veut réaliste, comme dans le cadre d'un simulateur médical, les interactions le seront également avec les contraintes que l'on rencontre dans le monde réel. À l'inverse, si le choix de représentation s'oriente vers un environnement non réaliste, on va voire apparaître des métaphores simples pour réaliser des actions complexes. Le choix de ces métaphores est dépendant des périphériques que l'on a décidé d'utiliser et de la cohérence que l'on veut conserver sur la globalité de l'application. Encore une fois pour une tâche donnée, il existe un nombre non fixé de métaphores d'interaction envisageable.

Pour prendre en compte ces contraintes, nous orientons donc notre plate-forme pour qu'elle permette la création des différents éléments de l'application (qui une fois développés restent réutilisables pour d'autres applications). La conception est séparée en deux partie :

- La conception des outils d'interaction, responsables de gérer les interactions sur les documents indépendemment du cadre de l'application.
- la conception des élément de l'interface en s'appuyant sur le patron de conception Modèle-Vue-Contrôleur.

L'ensemble est assemblé à l'aide de fichiers de descriptions au format xml.

2.2 Architectures Logicielles

Les applications souvent sont formées de composants d'interface (*widgets*) indépendant, ce découpage étant à la base de la réutilisabilité. Selon les types d'architectures logicielles, le découpage n'intervient pas au même niveau (PAC [6], MVC ou ses dérivés Document/Vue). La tendance commune consiste à séparer la partie fonctionnelle de l'application d'avec l'interaction et la représentation. Dans ces approches le comportement d'interaction est attaché aux objets de l'interface, ce qui fait que, dans beaucoup d'implémentation, la représentation et l'interaction sont gérés par le même composant logiciel.

Nous pouvons également citer virtools[1] comme modèle de conception, basé sur un outil auteur graphique, il permet la création d'application en connectant des composants entre eux. Ce système à l'avantage de permettre une réalisation rapide des applications à la condition que les blocs fonctionnels soient disponibles. Le gestion des interactions reste néanmoins sommaire pour un environnement 3D. Le modèle s'apparante ici à la gestion des interactions dans les documents 3D de type VRML/X3D qui utilisent les mécanismes senseurs, routes et scripts. Les senseurs permettent de rendre actif au pointeur des parties des documents, les routes vont permettre de nourrire le script qui va pouvoir modifier le document.

[1] http://www.virtools.com

Dans une application interactive, les modalités d'interaction sont donc portées par la représentation visuelle des objets de l'interface(l'interaction sur une barre de défilement s'appuie sur sa représentation), mais également par l'état de l'application et sont plus liées à la sémantique (le fait qu'un fichier en cours d'utilisation ne puisse être supprimé n'est pas lié à la représentation que peut prendre ce dernier dans l'interface sous forme d'icône, texte, vignette, ...). L'approche modulaire outre l'avantage de réutilisabilité permet également de pouvoir modifier ou adapter plus facilement une application. On pourra raffiner les composants au fil de la phase de conception.

Ces modèles d'architecture ne sont cependant pas adaptés dans le cas des environnements de type Réalité Virtuelle où l'interaction avec les objets virtuels est réalisée par l'intermédiaire d'outils d'interaction. On se retrouve dans un système où, d'un coté, le comportement d'interaction est défini par les fonctionnalités offertes par l'outil et, de l'autre, par les capacités de manipulation que supportent les documents de l'application.

2.3 Modèles pour l'interaction

Pour la gestion spécifique des interactions, des boîtes à outils comme [7][9] permettent de découpler la gestion des périphériques du reste de l'application. Une interface graphique permet de connecter les données émisent par les blocs représentant les périphériques aux éléments applicatifs. Des blocs issues de la bibliothèque peuvent également être intégrés dans le graphe pour définir des techniques d'interactions ou adapter les périhiphériques à l'application. Le langage graphique simple reste accessible à l'utilisateur final, il peut ainsi adapter l'application à ses besoins. Les changements du graphe en cours d'execution restent possibles mais demande de suspendre l'application. Par exemple si pour une application nous voulons changer les techniques d'interactions en cours d'utilisation, le graphe de configuration devra comporter toutes les techniques avec un composant *filtre* chargé de sélectionner la technique courante. La configuration peut alors rapidement devenir très complexe par le nombre de cas à traiter, même si les auteurs proposent des systèmes de factorisation.

A l'inverse, d'autres propositions tentent d'abstraire les couches basses pour offrir la même technique d'interaction quelque soit le périphérique physique utilisé. Dans VRED [10] les interactions sont définies à l'aide d'un langage graphique pour décrire les interactions, en utilisant des machines à états, mais dans le cas d'interactions complexes, la lisibilité devient difficile.

OpenDPI, ou son successeur DopiDom [3], introduit la notion d'interaction instrumentale [12]. Dans cette boîte à outil, l'interaction est centrée autour des documents structurés. Le choix des documents structurés fait qu'elle devrait être facilement adaptable à un environnements 3D où les formats des scènes, comme x3d ou VRML97, sont hiérarchiques et structurés.

HsmTk est une boîte à outil post-Wimp utilisant des machines à états hiérarchiques [4] pour décrire l'interaction indépendamment des objets qu'elles manipulent. HsmTk offre au concepteur la possibilité de décrire les machine à états dans directement dans le code de son application. Ceci permet d'intégrer au langage un moyen de décrire proprement les techniques interactions et facilite la prise en compte de celles-ci dans la phase de conception. Les techniques d'interaction développées restent réutilisables et peuvent être enrichies.

Patch Panel[2] propose un système qui permet d'abstraire la couche de communication entre les périphériques et les éléments interactifs. Les auteurs s'appuient sur les services qu'offrent et peuvent recevoir les composants de l'application. Un médiateur central *EventHeap* permet de les mettre en relation. Le système offre l'avantage de pouvoir être reconfigurer à l'execution et permet

à l'aide de machines à états de permettre des *mapping* plus complexes.

Les sytèmes actuels isolent la partie de description de l'interaction de interface qui prend désormais une part importante de leur boite à outils. L'interaction n'est plus fonction de l'interface graphique. Les avantages sont nombreux et selon le niveau traité, ils vont permettre d'adapter des périphériques physiques aux techniques d'interaction (ICON) où pouvoir choisir entre plusieurs techniques d'interactions (HsmTk).

Néanmoins aucun ne porpose d'adapter une technique où un outil d'interaction. Afin de modifier la façon d'interagir avec les éléments de l'interface en fonction des actions de l'utilisateur où du lieu dans l'interface où il agit.

3 Survol de notre propositions

Nos objectifs sont les suivants:

1. Proposer au développeur les outils logiciels pour la conception de briques d'interaction de bases ou spécifiques à l'application.

2. Proposer un modèle d'interface configurable, accessible au concepteur de l'application qui offre la possibilité d'adapter l'interaction au contexte de l'application.

3. Une approche modulaire qui permet de modifier ou adapter plus facilement une application et qui propose un processus incrémental de la conception d'une interface. Ce processus consistant à raffiner les comportements au fil des étapes de la conception.

Notre modèle d'interaction s'appuie sur une approche instrumentale de l'interaction concrétisée sous forme d'outils d'interaction. L'instrument outil devient médiateur entre les actions de l'utilisateur et les documents compatibles disponibles dans l'application. Notre proposition (figure 1) consiste à pouvoir modifier l'interaction entre un outil et un document en fonction de la zone d'interaction où se trouve le document. Les outils d'interaction sont basés sur des automates à états finis, modifier l'interaction d'un outil correspond donc à modifier son automate.

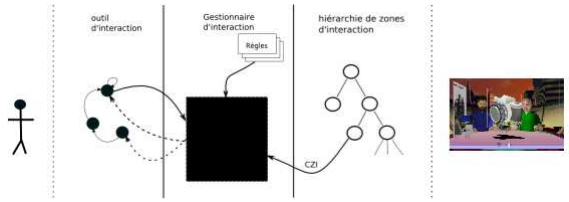

Figure 1: Cette figure présente une vue globale de l'architecture de la plate-forme. Elle présente l'outil comme médiateur entre l'utilisateur et l'interface, et le gestionnaire d'interaction comme élément central de l'adaptation de l'interaction.

Dans nos travaux antérieur nous avons développé un bureau 3D collaboratif (Spinl3D) basé sur une architecture MVC [11]. La plate-forme permet de concevoir chaque élément du M, V ou C dans un module dynamique. L'application finale est entièrement définie à l'aide d'un fichier de configuration au format XML décrivant l'assemblage de tous les modules et documents structurés (VRML97/X3D). Dans notre proposition, l'interface est composée de zones d'interaction contenant les documents, et d'outils d'interaction capables d'agir sur les documents compatibles. La plate-forme possède une couche d'abstraction des périphériques qui est utilisée par les outils d'interaction. Le système gère ainsi plusieurs périphérique pour permettre par exemple des interactions bi-manuelles ou l'usage de la reconnaissance vocale pour piloter les outils.

Du point de vue applicatif, nous gérons des évènements interactifs de haut niveau. Dans un contexte 3D, une gestion des interactions bas-niveau basée sur les évènements des periphériques devient trop complexe lorsque l'on ne se limite plus à du simple clic/action où de la visualisation d'information 3D (rotation/translation).

De la même façon, les manipulations d'attributs sur l'arbre des documents par les outils ne sont pas suffisamment discriminantes. Quand les modifications appliquées deviennent complexes comme dans le cas d'un assemblage d'objet 3D il devient difficile de determiner les actions réalisés par l'utilisateur.

Les évènements de haut-niveau (ayant un sens sémantique dans le contexte de l'application) sont générés par les outils d'interaction lorsqu'ils agissent sur les documents. Ils vont permettre à la plate-forme d'être informée des différentes actions de l'utilisateur dans l'interface.

3.1 Mécanisme d'interaction global

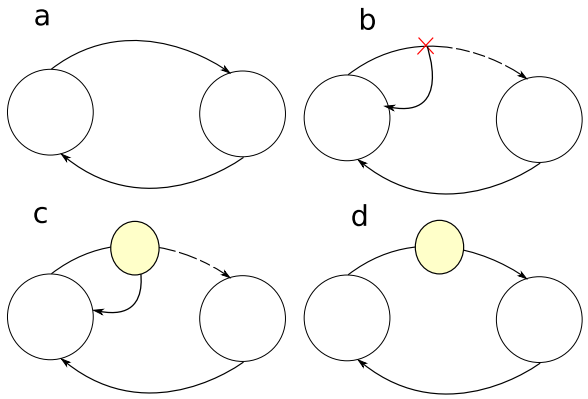

Figure 2: Cette figure illustre les différents types de modifications possibles sur l'automate de l'outil d'interaction.

Nous définissions le contexte d'interaction comme un ensemble d'informations qui relate l'activité de l'utilisateur à un moment précis de l'activité. Il contient, entre autre l'information spatiale de l'activité de l'utilisateur : le lieu dans l'interface où se trouve le pointeur, dans quel document, quelle zone, et sur ses actions : quel outil l'utilisateur manipule, dans quel état est l'outil... C'est en fonction de ce contexte, et d'un ensemble de règle de modification des interactions - que nous détaillerons plus loin -, que le gestionnaire d'interaction de notre plate-forme va modifier l'automate de l'outil. Dans le schéma d'interaction de base, l'automate de l'outil va suivre son schéma d'évolution classique, Nous sommes par défaut dans un mode d'interaction temporel. L'utilisateur utilise la modalité d'interaction prévue par l'outil jusqu'à ce qu'il change d'outil ou que la tâche pour laquelle l'outil était prévu soit terminée. Dans ce cas, selon les spécifications de l'application il peut par exemple repasser à l'utilisation d'un outil par défaut.

Notre but est de pouvoir modifier l'évolution de l'automate de l'outil afin d'intégrer à posteriori une modalité d'interaction spatiale différente. D'une manière générale, une activité va se traduire par une transition d'un état vers un autre, ou vers lui-même. Nous avons déterminé trois schémas de bases :

- L'outil évolue naturellement, ce qui correspond au comportement par défaut (figure 2 a).

- L'outil poursuit son évolution, mais un état supplémentaire - définie par le contrôleur de la zone d'interaction (CZI) -

est invoqué. On parle alors d'augmentation de la modalité d'interaction (figure 2 d).

- L'outil n'effectue pas sa transition, elle est refusée et un CZI est invoqué, on a ici substitution de comportement (figure 2 c).
- Un dernier cas consiste à ne pas autoriser la transition, mais il se rapporte au point précédant avec comme particularité un CZI vide (figure 2 b).

4 MODÈLE D'INTERACTION

4.1 Outils d'interaction

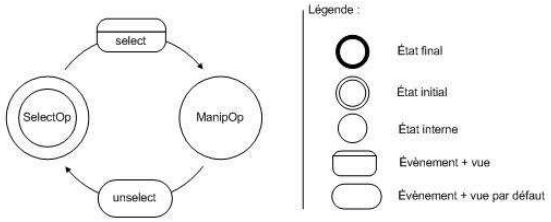

Figure 3: Automate de l'outil de sélection et manipulation décrit dans la figure 5.

Les outils d'interaction permettent d'instrumentaliser l'interaction. Ils ont pour rôle de supporter la réalisation d'une tâche. Ils sont animés par l'utilisateur à l'aide des périphériques d'entrée et possèdent une représentation dans l'environnement. L'outil d'interaction sert de médiateur entre l'utilisateur et les documents de l'interface pour réaliser ses actions. Nos outils d'interaction sont réalisés sous la forme d'automates à états finis décris en XML (figure 5). Cette déclaration autorisant l'assemblage de modules de programmation dynamiques représentant les états de l'outil permet de pouvoir facilement adapter la modalité d'un outil en remplaçant l'un des états. Prenons l'exemple d'un outil de sélection et de manipulation (figure 3), l'automate est composé de deux états, l'un servant à réaliser la sélection, l'autre la manipulation. Le fait de pouvoir configurer l'automate, permet par exemple de substituer le module responsable de la sélection à l'aide d'un périphérique de pointage comme la souris par un module capable de gérer la sélection à l'aide d'une commande vocale. Avoir des composants séparés permet de pouvoir réutiliser les modules déjà développés. Ceci se fait dans la limite de spécificité. Des actions " génériques " comme la sélection ont potentiellement plus de chance d'être réutilisé qu'une action spécifique à un type d'application particulière. L'implémentation de l'opérateur de sélection (figure 4) est définie est définie dans un module dynamique (dll/dso). Elle prend en entrée les différents périphériques (virtuels) dont l'opérateur a besoin, et agit sur l'environnement en fonction du contexte d'interaction. Lorsqu'il agit, l'opérateur génère un évènement nommé, représenté par une chaîne de caractère. Un opérateur de sélection génère par exemple l'évènement " select ". Cet événement n'est pas nécessairement de bas niveau (comme les déplacements du périphérique physique) mais relate l'action réalisée par l'opérateur. Les transitions de l'automate de l'outil d'interaction sont étiquetées par les évènements générés par les opérateurs. Elles sont décrites en XML à la configuration.

4.2 Contrôleur de zone d'interaction (CZI)

Les outils d'interaction définissent le mode d'interaction par défaut dans l'application en agissant sur les documents et leurs propriétés. Dans notre environnement l'interface est

```
<operatorDecl name="selectOp"
          instanceOf="file://select.dll" />
  <description> ... </description>
  <devices>
    <device name="pointing" />
  </devices>
  <events>
    <event value="select"/>
  </events>
</operatorDecl>
```

Figure 4: Déclaration xml d'un opérateur.

```
<Tool name="selectAndManip">
  <operators>
    <operator name="Select"
              instanceOf="selectOp" />
    <operator name="Manip"
              instanceOf="manipOp" />
  </operators>
  <start operator="Select" />
  <transitions>
    <transition>
      <from operator="Select" />
      <to   operator="Manip" />
      <on   event="select" />
    </transition>
    <transition>
      <from operator="Manip" />
      <to   operator="Select" />
      <on   event="unselect" />
    </transition>
  </transitions>
</Tool>
```

Figure 5: Déclaration xml d'un outil d'interaction. La première section liste les instances d'opérateur à utiliser en référence à une déclaration d'opérateur. Ensuite on retrouve la description de l'automate où est spécifié l'état initial, et l'ensemble des transitions caractérisant l'automate.

composée d'une hiérarchie de composants MVC, appelés zones d'interactions, ces zones contiennent d'autres zones et les documents de l'application. A chaque zone est associé un modèle, une représentation et un comportement d'interaction propre. Alors que l'outil spécifie le comportement par défaut, les contrôleurs des zones d'interaction permettent de définir un comportement alternatif ou adapté de l'interaction prévue par défaut. Le contrôleur possède les mêmes caractéristiques qu'un opérateur : il va être alimenté par les périphériques qui l'intéresse et le contexte d'interaction. Le système peut modifier dynamiquement l'automate de l'outil d'interaction pour y ajouter un CZI . Lorsque le CZI est invoqué, il génère à son tour un événement sous la forme d'un couple contenant le nom de l'action qu'il a réalisé et un deuxième élément dont la valeur peut être break ou continue. Cette valeur indique au système le type de modification à effectuer sur l'automate (figure 1).

5 MODIFICATION DE LA MODALITÉ

5.1 Règle d'interaction

Le mécanisme de règles d'interaction peut être considéré par analogie au système de règles que l'on retrouve dans les pare-feux sur les systèmes d'exploitation. Les pare-feu analysent le trafic qui transite sur le réseau et en fonction du type des paquets (TCP/UDP), des machines communiquant (adresse IP/masque) ou des services utilisés (http, ssh, ...) les paquets vont être bloqués ou non. Dans notre plate-forme nous considérons le même principe. Pour adapter l'interaction, en fonction de l'outil utilisé, de l'action réalisée, des documents et des zones d'interaction entrant en jeu (i.e. le contexte d'interaction), le noyau va pouvoir autoriser, ou non, les transitions des outils et faire éventuellement appel à un CZI. Le comportement du gestionnaire d'interaction du noyau est défini par une liste de règles. Une règle d'interaction est attachée à un CZI et spécifie le contexte d'interaction dans lequel elle doit se déclencher (figure 6).

```
<rule>
  <focus>
    <zone name=""
          propagate="true|false"
          negate="true|false" />
    <document name="" fqn="" />
  </focus>
  <on>
    <tool value="" />
    <operator value="" />
    <event value="" />
  </on>
  <behavior>
     <bind controller="" />
     <type
           value="break|continue|
                  func(controller)" />
  </behavior>
</rule>
```

Figure 6: Déclaration d'une règle d'interaction. On y retrouve la spécification du contexte d'interaction avec les tags focus et on. Le comportement à adopter par le système lorsque le contexte match est spécifié par le tag behavior, qui va contenir l'éventuel contrôleur à appeler et le type de contrôle du flux d'interaction (statique ou dynamique)

Focus. Le focus d'une règle correspond à localiser la partie de l'interface utilisateur dans laquelle la règle est potentiellement activable. Le focus peut être spécifié pour un document donné de l'interface ou une sous-partie d'un document dans le cas des objets 3D de type VRML97 ou X3D. La seconde partie du focus paramétrable est relative à la zone d'interaction dans laquelle est réalisée l'interaction (figure 7). Nous utilisons trois paramètres, le premier représente le nom de la zone d'interaction, ce qui correspond à un noeud dans le graphe de scène MVC. Le second paramètre propagate nous permet de spécifier si la règle s'applique également pour la sous hiérarchie de cette zone. Enfin le dernier paramètre negate permet de valider la règle lorsque l'interaction se déroule en dehors du focus précisé. Nous avons fait ce choix suite aux différentes applications que nous avons réalisées, ils traduisent ces quatre descriptions intuitives de zone de l'interface :

1. un élément particulier de l'interface;
2. partout sauf sur cet élément;
3. sur tout ce qui est contenu par cet élément;
4. partout sauf sur ce qui est contenu par cet élément. Nous pouvons citer par exemple le cas d'une fenêtre modale, où toute interaction en dehors de la fenêtre et de ce qu'elle contient n'est pas autorisée.

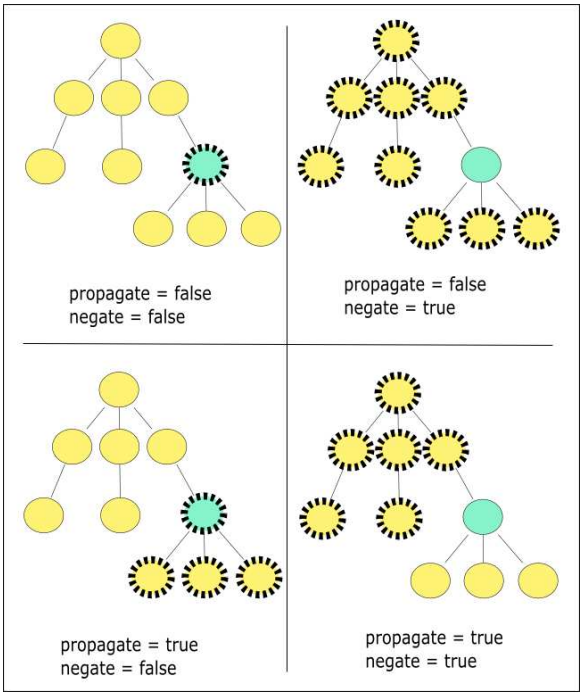

Figure 7: Cette figure illustre la représentation du focus dans le graphe de zones de l'interface selon la valeur des attributs propagate et negate. Les noeuds en pointillés représentent les zones où la règle d'interaction est active.

On. Le paramètre **on** permet de spécifier l'action de l'utilisateur qui doit déclencher la règle. L'outil en tant qu'automate évolue, les règles spécifient les transitions vont déclencher l'appel à un CZI. Les transitions sont identifiées par un triplet reprenant l'identifiant de l'outil, l'identifiant de l'opérateur et l'évènement généré par l'opérateur. Ainsi la transition qui fait passer l'outil de sélection/manipulation à l'état manipulation, est définie par le triplet SelectAndManip, SelectOp, Select. Nous autorisons l'utilisation de jocker pour la spécification des triplets, un jocker valide tous les identifiants. Ceci permet de baser des règles sur des comportements généraux d'interaction. Pour reprendre le cas du triplet précédent, si l'on remplace SelectAndManip par un jocker (*wildcard*), la règle deviendra valide dès qu'un opérateur SelectOp effectuera une sélection et ce quelque soit l'outil d'interaction utilisé.

Behavior. Les deux paramètres précédant permettent d'identifier les contextes interactifs dans lesquels doit se trouver l'application pour activer la règle. Le comportement que doit prendre le gestionnaire d'interaction est lui décrit par le tag behavior. Celui-ci permet de spécifier le CZI qui doit être invoqué par le système. Le type de comportement associé à la règle par le gestionnaire d'interaction est défini par le tag type. Nous pouvons déterminer soit un comportement statique qui sera défini une fois pour toute à l'initialisation de l'application, soit un comportement dynamique qui sera déterminé à l'exécution en fonction du code de retour du CZI. Dans les deux cas, le gestionnaire d'interaction

utilise deux mots clefs : continue et break. Continue est relatif à une augmentation de la modalité d'interaction, l'action en cours est autorisée et le CZI associé est invoqué. A l'inverse, break définit un comportement de substitution, l'action en cours n'est pas effectuée et le CZI associé est invoqué à sa place. Le comportement dynamique est défini par le code de retour de l'appel du CZI (figure 8 et 9). La valeur de retour du CZI est un couple contenant deux attributs :

1. le premier : **event** décrit l'action que le contrôleur à effectué,
2. le second : **behavior** permet au contrôleur de spécifier lui-même le comportement à appliquer dans l'évolution de l'automate de l'outil. Il prend donc deux valeurs possibles, **continue** ou **break**.

Pour ce faire, nous utilisons une évaluation du code de retour à l'aide de tests booléens. Enfin, l'attribut **bind** permet de réferencer le conrôleur à invoquer lorsque la règle est déclenchée.

```
<type value="controller.behavior" />
```

Figure 8: Exemple de spécification du comportement du gestionnaire d'interaction en fonction du comportement désiré par le contrôleur.

```
<type value="if( controller.event ==
                  "lock_success" )
                continue;
              else
                break;"
/>
```

Figure 9: Exemple de spécification du comportement du gestionnaire d'interaction en fonction de l'action réalisée par le contrôleur.

Pour la conception d'une application particulière, il est nécessaire de définir un ensemble de règles, dont le nombre peut augmenter rapidement selon la taille de l'application et le nombre d'outils d'interaction disponibles. Comme nos règles permettent de modifier le schéma d'exécution de l'application en fonction du contexte, guidé par les actions de l'utilisateur, on ne peut pas prédire quand une règle sera activée. De plus, plusieurs règles peuvent être valides au même instant. Pour répondre à ces problèmes, nous fixons l'ordre de parcours de la liste de règles. De cette façon il est plus facile au concepteur de comprendre les impacts des règles et d'adapter la configuration de son application progressivement, comme dans le cas de gestion des règles dans un pare-feu.

5.2 Gestionnaire d'interaction

Dans l'architecture de notre plate-forme, le gestionnaire d'interaction est responsable de la gestion des outils d'interaction, des règles d'interaction et des périphériques virtuels. La gestion générique des périphériques physiques au travers des périphériques virtuels sort du cadre de cet article. Nous retiendrons uniquement que chaque périphérique physique est accédé au travers d'un pilote de périphérique offrant au noyau une interface vers un périphérique virtuel. Les périphériques virtuels proposent un mécanisme d'observer/observable qui permet de notifier le gestionnaire d'interaction des changements d'états. Le gestionnaire gère ensuite la correspondance entre les périphériques virtuels et les clients consommateurs (opérateurs et contrôleurs). Le schéma d'exécution d'une boucle d'interaction est le suivant :

1. Le gestionnaire d'interaction regroupe les périphériques ayant changé d'état.
2. Il met à jour l'opérateur courant de l'outil d'interaction actif en lui fournissant les périphériques qui ont changé d'état.
3. A la suite de la mise à jour de l'opérateur (déclenchement d'une transition de l'automate), le gestionnaire d'interaction détermine le contexte d'interaction.
4. Il évalue alors toutes les règles d'interaction validant le contexte les unes à la suite des autres jusqu'à ce qu'une règle possède le comportement break ou que la liste soit vide (figure 10).
5. Si le traitement s'est terminé par un break, le gestionnaire d'interaction effectue un roll back et l'automate de l'outil reste dans l'état précédant. Si au contraire aucune règle n'a été trouvée ou que chacune avait un comportement continue, la transition est autorisée et l'automate évolue naturellement.

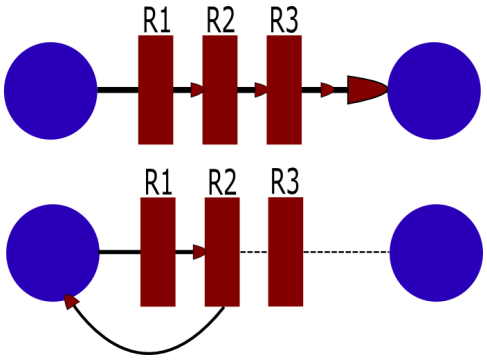

Figure 10: Cette figure illustre les différents types de chaînage dans le processus de traitement des règles valides. L'image du haut présente le cas où trois règles sont valides et ont comme comportement continue. L'image du bas le cas où la seconde règle à un comportement break, le flux d'exécution est alors interrompu la transition n'est pas effectuée.

Le gestionnaire d'interaction peut être amené à devoir gérer de nombreuses règles. Afin de préserver du temps processeur, plusieurs techniques sont utilisées pour réduire le temps nécessaire pour trouver l'ensemble des règles valides pour un contexte d'interaction donné. La première consiste en un cache qui conserve la dernière liste de règles valides. Si le contexte d'interaction reste inchangé alors la liste n'est pas recalculée. La seconde optimisation repose sur les comparaisons entre chaînes de caractères : à chaque chaîne est associé un entier unique, comparer deux chaînes revient alors à comparer deux entiers.

6 Notion de Taxonomie

Nous venons de présenter l'architecture du gestionnaire d'interaction, Ce dernier permet de manière simple de modifier les modalités d'interaction des outils. Cependant, le système de règles se base sur la notion d'identifiant (comme le nom des zones, des outils, des opérateurs, des événements ...), ce qui amène à devoir multiplier le nombre de règles d'interaction en présence d'objets ayant des identifiants différents mais une sémantique commune. Prenons l'exemple d'un CZI qui doit être appelé lorsque l'utilisateur sélectionne un objet de type bouton. La règle déclenchant ce CZI se base sur le nom de la zone, i.e. simple description spatiale. Il faudra donc déclarer autant de règles que de boutons. Pour résoudre ce problème, nous avons introduit un

mécanisme qui permet de structurer les identifiants. La plate-forme permet de définir une taxonomie sur les identifiants par des relations ensemblistes du type " est un ". Cela s'applique à tous les identifiants utilisés dans la plate-forme. Ce nommage peut être considéré comme une extension des *tags* utilisés dans SMCanvas [1] qui permettent de traiter des éléments par goupe. Notre modèle est hiérarchique, un identidiant peut appartenir à plusieurs groupes, qui eux même peuvent appartenir à d'autres groupes.

Dans le fichier de configuration nous pouvons introduire de nouveaux identifiants " clefs " qui vont correspondre aux groupes et les hiérarchiser. Par exemple, des évènements rotation et translation sont émis par plusieurs opérateurs, on peut définir un nouvel identifiant transformation et décrire que :

- Translation est une transformation
- Rotation est une transformation

On peut alors décrire une règle d'interaction valide pour l'évènement formel transformation. Le gestionnaire d'interaction validera cette règle dès qu'un évènement rotation ou translation est émis. Cette solution en plus de permettre de réduire le nombre de règles d'interaction, permet de contourner le problème des architectures modulaires comme la notre. Dans ce type d'architecture, les composants utilisés ne sont pas tous développés par la même personne. Avec la taxonomie, il est possible d'adapter les conventions de nommages utilisées par le concepteur du module à celles de l'application. On peut également définir des caractérisations plus sémantique qui sont propre à l'application. On peut par exemple déclarer les identifiants private et public dans une application collaborative et associer à chacun de ses identifiants les zones d'interaction de l'interfaces qui sont partagée et celles qui ne le sont pas.

7 Exemples

Nous présentons dans cette section quelques exemples d'utilisation du système de règles extraites d'applications que nous avons développées. Les différentes parties de codes xml présentées dans les exemples sont limitées à la déclarations des règles. Le lecteur pourra se référer à [11] pour la description des outils et zones d'interaction.

7.1 Filtrage des périphériques.

Dans les systèmes d'exploitation graphiques 2D classiques, le déplacement du pointeur dans l'interface reste contraint à la dimension de la fenêtre. Si le périphérique physique utilisé fournit des incréments (une souris par exemple), le système doit pouvoir contrôler les données provenant du périphérique pour stopper les incréments lorsque que le pointeur arrive aux frontières de l'interface. Ce contrôle est simplement mis en place dans notre architecture à l'aide d'une règle d'interaction (figure 11) qui invoque un contrôleur dont le rôle consiste à filtrer les déplacements de tous les outils (l'événement move est généré directement par le noyau lors du déplacement du périphérique). Ce type de règle peut être utilisée également pour effectuer du filtrage passe bas, par exemple dans le but d'atténuer les effets de tremblement lorsque le périphérique utilisé est trop sensible.

7.2 Gestion de verrous.

Notre deuxième exemple illustre l'utilisation des règles d'interaction pour ajouter, à posteriori, un contrôle de sélection exclusif pour une application collaborative. Le but recherché est de pouvoir poser un verrou sur les objets que l'on sélectionne : lorsque qu'un verrou est posé, cela signifie qu'un autre utilisateur accède à cet objet, la sélection de l'objet doit alors échouer. La mise en place du système requiert deux contrôleurs (figure 12). Le premier a pour rôle de poser le verrou sur l'objet en cours de

```
<rule>
  <focus>
    <zone name="RootZone"
          propagate="true" />
  </focus>
  <on>
    <tool     value="*" />
    <operator value="*" />
    <event    value="move" />
  </on>
  <behavior>
     <bind controller="filter" />
     <type value="continue;" />
  </behavior>
</rule>
```

Figure 11: Règle d'interaction qui permet de filtrer les déplacements des outils d'interaction. **RootZone** fait référence à la zone d'interaction racine de l'application, l'attribut **propagate** à *true* indique que la règle sera valide dans toutes les zones d'interaction de l'application. La règle se déclenchera sur tous les évènements **move** qui correspondent au déplacement de l'outil courant. Dans ce cas le CZI nommé **filter** de la zone **RootZone** sera invoqué pour limiter l'espace de déplacement du pointeur 3D.

sélection (l'évènement **select** sera émis par tous les outils effectuant une sélection), le second a pour rôle de le retirer et doit être appelé lorsque l'on désélectionne le document. Les contrôleurs sont invoqués à l'aide de deux règles. Le rôle du contrôleur **LockCtrl** est de tenter de poser un verrou sur le document en train d'être séléctioné. En cas de succés il émet l'évènement *lock_success* en cas d'échec,il déclenche un retour visuel pour prévenir l'utilisateur et génèrera un évènement *lock_failed*. Pour restreindre cette gestion de verrous aux zones publiques de l'interface, toutes les zones partagées sont déclarées appartenir au groupe **Public**.

8 Conclusions et perspectives

Dans cet article nous avons présenté un système de description à base de règles qui permet de spécifier le comportement interactif d'une application construite autour du patron de conception MVC. Ce système offre l'avantage de pouvoir mêler un modèle d'interaction instrumentale au travers d'outils d'interaction avec le mécanisme usuel des contrôleurs du MVC. L'approche entièrement descriptive facilite le prototypage des applications en permettant une conception progressive. La configuration peut, de plus, être réalisée à l'aide d'un outil auteur graphique (en cours de développement) ce qui rend la phase de conception plus accessible. L'architecture entièrement modulaire, concept de base adopté du génie logiciel, nous apporte tous les avantages liés à celui-ci comme la réutilisabilité, la conception isolée. La conception isolée permet de pouvoir développer plus rapidement, les modules peuvent être testé de façon isolée et offre ainsi un débogage plus aisé que lorsque la fonctionnalité est noyée dans la masse de code d'une application. La principale voie de communication utilisée entre les modules et le noyau de la plate-forme est basée sur les chaînes de caractères. Notre proposition d'identifiants structurés permet de passer outre les problèmes introduits par ce type de système comme les conventions de nommage à imposer aux différents développeurs. Elle permet de pouvoir adapter les modules utilisés au contexte de l'application en développement. En outre, elle permet également de pourvoir définir le vocabulaire sémantique de l'application. Nous pensons maintenant orienter nos travaux sur la finalisation des outils d'interaction pour en augmenter leur flexibilité. Les travaux porterons sur la possibilité de définir des

```
<rule>
  <focus>
    <zone name="Public"
          propagate="false" />
  </focus>
  <on>
    <tool     value="*" />
    <operator value="*" />
    <event    value="select" />
  </on>
  <behavior>
     <bind controller="LockCrtl" />
     <type
       value="if(controller.event ==
                 lock_success)
                 continue;
               else
                 break;" />
  </behavior>
</rule>
<rule>
  <focus>
    <zone name="Public"
          propagate="false" />
  </focus>
    <on>
    <tool     value="*" />
    <operator value="*" />
    <event    value="release" />
  </on>
  <behavior>
     <bind controller="UnLockCrtl" />
     <type value="continue;" />
  </behavior>
</rule>
```

Figure 12: Règles d'interaction qui permettent d'effectuer la gestion de verrous sur la sélection d'objets dans les parties publiques de l'interface.

opérateurs en assemblant ceux préexistants pour augmenter le degré de réutilisabilité. Un autre point porte sur la combinaison d'outils disponibles dans l'interface (un outil compas en conjonction avec l'outil de dessin permet de tracer des cercles dans l'interface). L'introduction de cette notion de méta-instruments demande la résolution de plusieurs problématiques :

- Tout d'abord informatique : comment décrire informatiquement les propriétés offertes et recevables des modules, leur niveau de compatibilité et leur dialogue.
- Métaphorique : comment offrir une interface intuitive à l'utilisateur.

Si la composition des outils peut se faire de manière automatique, l'usage qu'en font les utilisateurs pourrait faire emmerger de nouveaux comportements d'interaction non définis par le concepteur.

ACKNOWLEDGEMENTS

Ces travaux ont été menés dans le cadre du projet Alcove et sont soutenus par l'Ircica (Institut de Recherche sur les Composants logiciels et matériels pour l'Information et la Communication Avancée) et France Télécom R&D.

REFERENCES

[1] C. Appert and M. Beaudouin-Lafon. Smcanvas: augmenter la boîte à outils java swing pour prototyper des techniques d'interaction avancées. In *IHM '06: Proceedings of the 18th international conference on Association Francophone d'Interaction Homme-Machine*, pages 99–106, New York, NY, USA, 2006. ACM Press.

[2] R. Ballagas, A. Szybalski, and A. Fox. Patch panel: Enabling control-flow interoperability in ubicomp environments. In *PERCOM '04: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04)*, page 241, Washington, DC, USA, 2004. IEEE Computer Society.

[3] O. Beaudoux. Dopidom: une approche de l'interaction et de la collaboration centrée sur les documents. In *IHM '06: Proceedings of the 18th international conference on Association Francophone d'Interaction Homme-Machine*, pages 19–26, New York, NY, USA, 2006. ACM Press.

[4] R. Blanch. Programmer l'interaction avec des machines à états hiérarchiques. In *IHM '02: Proceedings of the 14th French-speaking conference on Human-computer interaction (Conférence Francophone sur l'Interaction Homme-Machine)*, pages 129–136, New York, NY, USA, 2002. ACM Press.

[5] B. D. Conner, S. S. Snibbe, K. P. Herndon, D. C. Robbins, R. C. Zeleznik, and A. van Dam. Three-dimensional widgets. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 183–188, New York, NY, USA, 1992. ACM Press.

[6] J. Coutaz. Pac, an object oriented model for dialog design. In *Interact'87 : the IFIP Conference on Human Computer Interaction*, pages 431–436, 1987.

[7] P. Dragicevic and J. Fekete. Input device selection and interaction configuration with icon. In *Actes de la Confrence Internationale IHM-HCI 2001*, pages 543–548, 2001.

[8] S. P. G. E. Krasner. A cookbook for using the model-view-controller user interface paradigm in smalltalk-80. In *Journal of Object OrientedProgramming*, pages 26–49, 1988.

[9] S. Huot, C. Dumas, P. Dragicevic, J. Fekete, and G. Hégron. The MAGGLITE post-wimp toolkit: Draw it, connect it and run it. In *Proceedings of the 17th ACM Symposium on User Interface and Software Technology (UIST 2004)*, pages 257–266, New York, NY, USA, 24–27 Oct. 2004. ACM/SIGCHI, ACM Press.

[10] R. J. K. Jacob, L. Deligiannidis, and S. Morrison. A software model and specification language for non-wimp user interfaces. *ACM Trans. Comput.-Hum. Interact.*, 6(1):1–46, 1999.

[11] N. Martin, S. Degrande, and C. Chaillou. Une utilisation du modèle mvc pour une plate-forme de travail virtuel. In *IHM 2005: Proceedings of the 17th conference on 17ème Conférence Francophone sur l'Interaction Homme-Machine*, pages 131–138, New York, NY, USA, 2005. ACM Press.

[12] M. B.-L. O. Beaudoux. Dpi: A conceptual model based on documents and interaction instruments. In *People and Computer XV - Interaction without frontier (Joint proceedings of HCI 2001 and IHM 2001)*, pages 247–263. Spring Verlag, 2001.

[13] G. G. Robertson, J. D. Mackinlay, and S. K. Card. Cone trees: animated 3d visualizations of hierarchical information. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 189–194, New York, NY, USA, 1991. ACM Press.

[14] USIXML.org. usixml.org.

[15] A. van Dam. Post-wimp user interfaces. *Commun. ACM*, 40(2):63–67, 1997.