FlowStates: prototypage d'applications interactives avec des flots de données et des machines à états

Caroline Appert, Stéphane Huot, Pierre Dragicevic, Michel Beaudouin-Lafon

FlowStates: Prototypage d'applications interactives avec des flots de données et des machines à états

Caroline Appert[1,2] *Stéphane Huot*[1,2] *Pierre Dragicevic*[2] *Michel Beaudouin-Lafon*[1,2]

[1]LRI - Univ. Paris-Sud & CNRS
Bât. 490, F-91405 Orsay, France

[2]INRIA
Bât. 490, F-91405 Orsay, France

{appert,huot,dragice,mbl}@lri.fr

RESUME

Cet article présente FLOWSTATES, une boîte à outils compatible avec Java Swing qui associe deux modèles de gestion des événements : les flots de données et les machines à états. Le modèle à flot de données permet de facilement interfacer des périphériques non standard et de reconfigurer les interactions en fonction des périphériques disponibles, tandis que les machines à états permettent de réaliser aisément des interactions complexes. À travers des exemples, l'article illustre la puissance et l'expressivité de cette approche hybride et la flexibilité qui résulte du choix explicite de ne pas fixer les limites entre les rôles de chaque modèle.

MOTS CLES : Boîte à outils, flot de données, machine à états

ABSTRACT

This article introduces FLOWSTATES, a user interface toolkit compatible with Java Swing that combines two models for managing events : dataflow and state machines. The dataflow model makes it easy to support non-standard input devices and to reconfigure interactions according to the available devices, while state machines support the programming of complex interactions. The article illustrates the power and expressivity of this hybrid approach and the flexibility afforded by the explicit decision to not set strict limits between the roles of each model.

CATEGORIES AND SUBJECT DESCRIPTORS: H5.m. Information interfaces and presentation (e.g., HCI) : Miscellaneous.

GENERAL TERMS: Languages, Design.

KEYWORDS: User interface toolkit, dataflow, state machine

INTRODUCTION

La gestion des entrées dans les interfaces graphiques est un problème notoirement complexe. La plupart des boîtes à outils d'interface actuelles utilisent le modèle des fonctions de rappel ("callbacks") qui sont appelées lorsqu'un événement se produit. Ce modèle est parfois mis en oeuvre sous la forme d'une délégation dans les langages à objets, comme par exemple avec les "listeners" de Java. Un autre modèle qui a été largement exploré est celui des machines à états, fondées sur des automates à états finis dont le langage d'entrée est le vocabulaire de types d'événements disponibles. Cette approche a par exemple été récemment intégrée à la boîte à outils Java Swing avec l'extension SWINGSTATES [3]. D'autres modèles utilisent des automates plus élaborés comme les machines à états hiérarchiques [6] ou les réseaux de Petri [28]. Un troisième modèle considère les événements comme des signaux qui se propagent selon un flot de données, paradigme issu des langages et modèles réactifs. La boîte à outils ICON [14] en est un bon exemple.

Chacun de ces modèles comporte des avantages et des inconvénients selon les types d'événements qui sont traités, selon le type de techniques d'interaction à implémenter, selon le niveau d'abstraction des événements manipulés, selon le degré de dynamicité souhaité, etc. Jusqu'à présent, à de rares exceptions près [22, 28, 13], la plupart des boîtes à outils ont utilisé un seul de ces modèles, obligeant le programmeur à s'accomoder de ses limitations.

Cet article présente FLOWSTATES, une boîte à outil qui combine ICON et SWINGSTATES et associe ainsi flot de données, machines à états et graphe de scène. Les événements de bas niveau sont d'abord traités par ICON, ce qui permet de facilement interfacer des périphériques non standard et de reconfigurer les interactions en fonction des périphériques disponibles. Les modules ICON qui s'interfacent normalement avec l'application [14] ou un graphe de scène [21] sont connectés à SWINGSTATES : les signaux d'entrée de ces modules sont transformés en événements abstraits traités par les machines à états de SWINGSTATES. Ceci permet de facilement réaliser des interactions complexes, qui sont souvent décrites dans la

littérature par des automates. Les machines à états agissent à leur tour sur le graphe de scène et le noyau applicatif via les actions associées aux transitions.

Cette organisation en trois couches (*dispositifs d'entrée*, *logique d'interaction* et *actions sur les objets de l'application*) résulte de notre expérience de ces différents modèles et de leur adéquation respective à différents niveaux d'abstraction lors du traitement des événements. Les différents styles de spécification (langage visuel pour ICON et langage impératif pour SWINGSTATES) reflètent également des besoins différents : la gestion des périphériques d'entrée est hautement variable alors que la logique de l'interaction (également appelée dialogue) est plus figée et plus fortement couplée au noyau applicatif [1]. Cependant les limites entre les modèles ne sont pas fixées, de telle sorte que le programmeur garde un contrôle total sur l'utilisation qu'il fait de chaque modèle. En outre, les machines à états peuvent réinjecter des événements vers le flot de données et être déconnectées du noyau applicatif, ce qui permet de s'affranchir des limites du modèle à couches et de combiner les techniques d'interaction à la manière du modèle de l'interaction instrumentale [5].

Dans la suite de cet article, nous passons en revue les différents types de boîtes à outils d'interface. Nous présentons ensuite FLOWSTATES et donnons un aperçu de son pouvoir d'expression à travers des exemples de prototypage d'interactions avancées. Enfin, nous discutons et mettons en évidence les différences entre notre approche et les précédents travaux d'intégration de modèles de machines à états et de flots de données.

APPROCHES POUR PROGRAMMER L'INTERACTION

De nombreux travaux de recherche ont eu pour objet de rendre la programmation de l'interaction avancée possible et plus facile. Nous donnons ici un aperçu des systèmes à écouteurs d'événements classiques et de leurs évolutions, des systèmes à états et des systèmes à flots de données.

L'approche basée sur les écouteurs d'événements

C'est l'approche que le programmeur doit suivre avec les boîtes à outils largement utilisées comme Java Swing ou Gtk. La construction d'une interface avec ces boîtes à outils se résume à un assemblage de composants graphiques ("widgets") dont les liens internes et les liens avec les périphériques d'entrée se font à l'aide d'écouteurs d'événements. Ce type d'application interactive est connu pour être difficile à modifier et à maintenir [25]. Cet état de fait a motivé le développement de boîtes à outils expérimentales avec une gestion des entrées plus flexible. Deux contributions importantes dans ce domaine sont *subArctic* [20] et *Garnet/Amulet* [26, 27]. La force de subArctic est de rendre plus transparente la politique d'aiguillage des événements AWT et de permettre la redéfinition de celle-ci par le programmeur. Quant aux boîtes à outils Garnet/Amulet, elles introduisent le concept d'*interacteur*, un objet qui intercepte les événements et les transforme en opérations sur un objet graphique. Garnet/Amulet propose 6 interacteurs prédéfinis qui reposent sur des machines à états très simples de type "press-drag-relase" mais ne fournit pas de support permettant la programmation d'autres types d'interacteurs. Dans ces deux exemples, le vocabulaire d'interaction n'est pas vraiment enrichi et les moyens de l'enrichir ne sont pas transparents pour le programmeur d'interfaces.

L'approche dirigée par les états

Contrairement au modèle d'écouteurs dans lequel chaque type d'événement doit spécifier sa fonction de rappel, certains modèles sont centrés autour des états du système interactif et ne considèrent que les événements ayant une sémantique dans ces états. Le formalisme des machines à états, basé sur les automates à états finis, a fait ses preuves quant à sa capacité à décrire intelligiblement des interactions complexes "sur le papier" [8, 19]. La boîte à outils SWINGSTATES intègre cette structure de contrôle directement dans le langage Java, afin de faciliter son adoption par les programmeurs d'applications interactives.

Les deux principales limites des machines à états sont l'explosion potentielle du nombre d'états et une gestion limitée du parallélisme. Certains travaux ont donc considéré des formalismes plus complexes comme la boîte à outils hsmTk [6] qui propose des machines à états hiérarchiques dans le langage C++ ou encore l'environnement PetShop [28] qui permet de spécifier la logique de l'interaction par des réseaux de Petri. Mais si la puissance de ces formalismes a été démontrée pour modéliser des interactions complexes, les machines à états simples comme celles de SWINGSTATES ont l'avantage d'être rapides à maîtriser. Notre expérience avec des étudiants de Master a d'ailleurs montré que des développeurs non aguerris étaient capables d'utiliser ce formalisme pour décrire une grande variété de techniques publiées dans la littérature en IHM ces dernières années [3]. En outre, le parallélisme peut être décrit en exécutant simultanément plusieurs machines à états et/ou en les faisant communiquer [3].

L'approche à flot de données

Dans les systèmes à flot de données, chaque changement de valeur déclenche immédiatement le recalcul des valeurs qui en dépendent. Ils sont couramment utilisés dans les domaines du traitement d'images, de vidéos et du son [11, 24], ainsi que pour le prototypage d'interfaces 3D [32, 12]. Auparavant confinée à ces domaines, cette approche est de plus en plus appliquée au prototypage d'interfaces homme-machines avancées [14, 4, 23]. L'un de ces systèmes, ICON, a été couplé avec un modèle de graphe de scène [21], permettant ainsi de décrire non plus seulement des traitements de données mais aussi des techniques d'interaction comportant du graphisme riche. La boîte à outils Qt propose également une extension au langage C++ pour proposer une gestion des écouteurs d'événements selon une logique flot de données. Un objet source peut émettre des signaux alors qu'un objet cible

peut déclarer des slots d'entrée avec des fonctions de rappel qui seront exécutées chaque fois que le slot d'entrée correspondant recevra un signal. La connection entre signal et slot doit être déclarée à l'objet application.

Les flots de données ont plusieurs avantages : ils peuvent être hautement modulaires (les calculs peuvent être encapsulés dans des briques indépendantes), ils permettent de décrire des traitements séquentiels et parallèles avec la même facilité, et ils se prêtent bien à la programmation visuelle. Le principe d'ICON est d'exploiter cette flexibilité pour rendre les applications interactives plus facilement configurables et personnalisables en fonction des périphériques d'entrée disponibles. Par exemple, une application de dessin peut facilement être livrée avec une configuration standard de type souris-clavier, une configuration exploitant le canal de pression des tablettes graphiques, et une configuration d'accessibilité reposant sur la reconnaissance vocale. Ces configurations peuvent ensuite être personnalisées par des utilisateurs experts en fonction de leurs besoins et préférences personnelles, ou dans le but d'exploiter au mieux leur configuration matérielle (par exemple, exploiter l'orientation du stylet fournie par certaines tablettes graphiques). Les configurations d'entrée d'ICON peuvent être vues comme des spécifications InTml [15]. Toutefois, elles sont éditables et exécutables par le moteur réactif d'ICON alors que InTml reste une spécification XML purement descriptive des composants d'une interface de réalité virtuelle.

Malgré sa flexibilité, l'approche à flots de données a un inconvénient majeur : les comportements décrits avec ce paradigme peuvent devenir visuellement très complexes, en particulier si les interactions comportent beaucoup de contrôle (traitements conditionnels et modes). C'est pourquoi FLOWSTATES adopte une *approche hybride* qui couple flot de données et machines à états. Les autres solutions hybrides existantes seront discutées et comparées à FLOWSTATES à la fin de cet article.

SWINGSTATES ET ICON : RAPPELS

Les articles [3] et [14] décrivent respectivement les boîtes à outils SWINGSTATES et ICON en détail. Nous ne rappelons ici que les éléments nécessaires à cet article.

SwingStates

Une des particularités de SWINGSTATES est l'introduction des machines à états comme structure de contrôle pour programmer l'interaction en Java. La figure 1 met en parallèle la représentation graphique d'une machine à états permettant de faire du déplacement de formes graphiques par "drag-and-drop" et son code SWINGSTATES. La machine à états est une classe Java dont les champs sont les états de la machine (en gras). Ces états sont aussi des classes, et leurs champs sont les transitions sortantes de l'état (en italique). Les transitions sont à leur tour des classes.

```
1  StateMachine sm = new StateMachine() {
2    SMShape dragged = null;
3
4    public State start = new State() {
5      Transition dragOn =
6      new PressOnShape(BUTTON1, ">> drag") {
7          public void action() { dragged = getShape(); }
8      };
9    };
10
11   public State drag = new State() {
12     Transition drag =
13     new Drag(BUTTON1) {
14       public void action() { move(dragged); }
15     };
16     Transition dragOff =
17     new Release(BUTTON1, ">> start") {
18     };
19   };
20 };
```

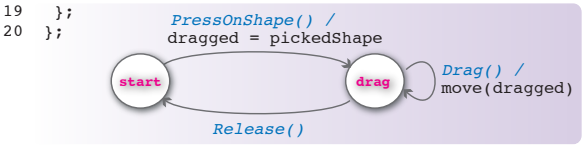

FIGURE ***1*** **:** Illustration de la syntaxe de SWINGSTATES pour programmer une machine à états.

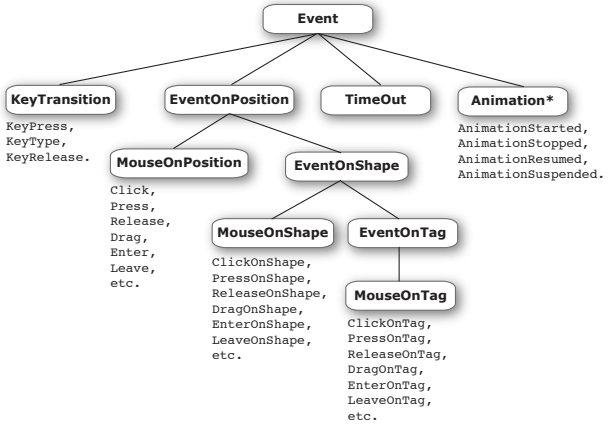

FIGURE ***2*** **:** Les classes de transitions disponibles dans SWINGSTATES.

SWINGSTATES propose un ensemble de classes de transitions qui sont déclenchées par différents types d'événements. Par exemple, la transition *Drag* (Fig.1, lignes 12-13) est déclenchée par un déplacement de la souris avec le bouton (gauche ici) appuyé. Le constructeur d'une classe de transition comporte un argument textuel optionnel qui indique l'état d'arrivée. Par exemple, la transition *DragOff* (Fig.1, lignes 16-17) mène vers l'état *start*).

Comme illustré sur la figure 2, SWINGSTATES propose un ensemble de classes de transitions qui tire profit de son modèle de dessin structuré pour offrir un plus grand pouvoir d'expression que Java AWT en intégrant en particulier des mécanismes de sélection avancés au niveau de la boîte à outils ("picking"). En effet, pour chaque événement positionnel (click, par exemple), SWINGSTATES propose 3 classes de transition qui permettent de s'abonner à des événements de ce type survenant : sur une forme graphique particulière (*ClickOnShape*), sur toutes les formes graphiques portant une certaine étiquette ou *tag* (*ClickOnTag*), ou survenant n'importe où (*Click*). Bien

que cette approche soit plus flexible pour le traitement des événements, SWINGSTATES repose sur le système d'événements AWT et ne permet donc pas de gérer les périphériques d'entrée non standard. Une première extension pour la gestion d'entrées avancées avait été développée en utilisant la librairie jInput [1]. Mais elle ne permet qu'une connexion directe aux canaux physiques détectés par jInput, sans représentation structurée des périphériques, ni possibilité d'adaptation des données ou de configuration dynamique.

L'architecture de SWINGSTATES reste pourtant ouverte à une gestion plus large des événements grâce à des transitions génériques (*Event*, *EventOnPosition*, *EventOnShape* et *EventOnTag*) qui répondent à des événements virtuels. Le programmeur peut spécifier la classe d'événements capables de déclencher une telle transition dans le constructeur de cette dernière. Par exemple, l'extrait de code ci-dessous montre la définition d'une classe d'événements de type *Zoom* (ligne 1) et une transition qui répond à ce type d'événements (ligne 14) :

```
1 public class Zoom extends VirtualPositionEvent {
2    private double zoomFactor;
3    public Zoom(Point2D zCenter, double zFactor) {
4       super(zCenter);
5       zoomFactor = zFactor;
6    }
7    public double getZoomFactor() {
8       return zoomFactor;
9    }
10 }
11
12 CStateMachine interaction = new CStateMachine() {
13    ...
14    Transition t = new EventOnPosition(Zoom.class) {
15       public void action() {
16          Zoom event = (Zoom)getEvent();
17          zoomBy(event.getPoint(),
18                 event.getZoomFactor());
19       }
20    };
21    ...
22 };
```

ICon

ICON repose sur un modèle à flots de données dont les briques, appelées *dispositifs*, sont une généralisation des dispositifs d'entrée : ils peuvent produire des valeurs de sortie mais peuvent aussi en recevoir (Figure 3, en bas). Ces valeurs sont émises sur des *slots de sortie* et reçues sur des *slots d'entrée*. Dans le langage visuel d'ICON, ces slots varient selon leur type (cercles pour les booléens, triangles pour les entiers, etc.) et peuvent être organisés de façon hiérarchique pour représenter des types structurés.

Une *configuration d'entrée* est un flot de données qui relie des périphériques à un noyau applicatif (Figure 3, en haut). Les configurations d'entrée sont construites en interconnectant des dispositifs dans l'éditeur graphique. Pour cela, ICON fournit des *dispositifs système* (qui décrivent des ressources matérielles comme les périphériques d'entrée) et des *dispositifs utilitaires* (allant de simples opérateurs booléens jusqu'à des techniques d'interaction

1. https://jinput.dev.java.net/

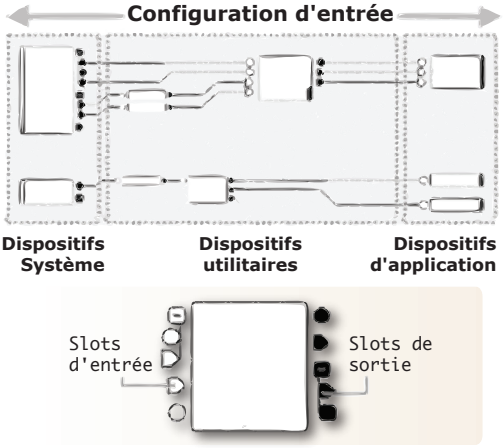

FIGURE *3* : Configuration d'entrée dans ICON.

comme des "toolglass" ou des interpréteurs de gestes, en passant par des retours graphiques tels que des curseurs). De son côté, le programmeur doit concevoir les *dispositifs d'application* qui vont contrôler les fonctionnalités ou les objets de son application.

Comparé au modèle événementiel standard, ce modèle rend la description de l'interaction en entrée plus explicite, plus fine, et plus facilement reconfigurable (y compris pendant l'exécution du programme). Il est aussi beaucoup plus facile à étendre. En effet, avec une boîte à outils classique, prendre en charge un nouveau périphérique d'entrée ou une nouvelle technique d'interaction nécessite de : 1) définir un nouveau type d'événement lié au périphérique ; 2) modifier et adapter le mécanisme de propagation des événements ; 3) étendre les objets qui doivent réagir à ces événements. Avec ICON, il suffit de créer un dispositif qui encapsule le nouveau périphérique ou la nouvelle interaction, et externaliser son interface sous la forme de slots d'entrée et/ou de sortie. Le périphérique ou la technique d'interaction peuvent alors être réutilisés dans de nombreuses configurations d'entrée.

Enfin, avec ICON, le noyau applicatif et les techniques d'interaction ne sont plus câblés à des périphériques d'entrée spécifiques. Il leur suffit de déclarer les canaux d'entrée dont ils ont besoin pour fonctionner. C'est, entre autres, cette propriété dont nous avons tiré parti pour l'association entre SWINGSTATES et ICON, comme nous le décrivons dans la section suivante.

FLOWSTATES

Cette section illustre, via des scénarios, l'utilisation de FLOWSTATES et en détaille les mécanismes sous-jacents.

Contrôle de machines SwingStates avec ICon

Bob est étudiant en Master IHM. Dans le cadre d'un projet de cours, il doit prototyper un éditeur de dessin prenant en charge les opérations de déplacement et de changement d'échelle (*pan* et *zoom*). Il n'a pas encore fait le choix des techniques de navigation qu'il désire rendre disponibles

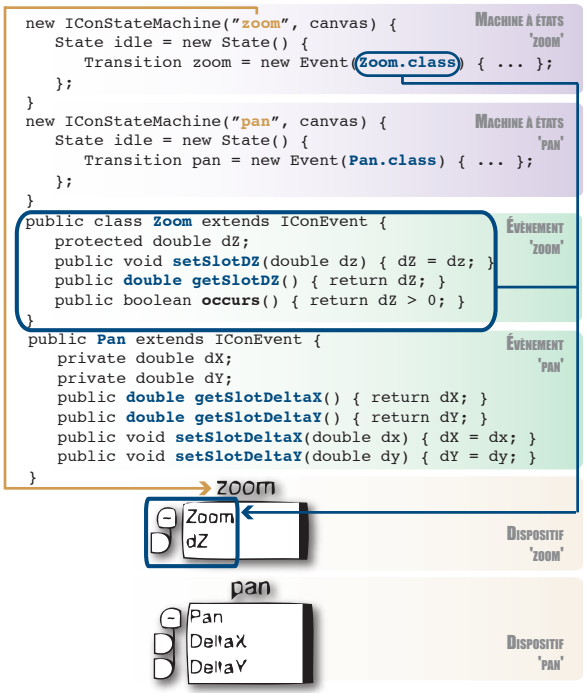

FIGURE *4* : Le programme SWINGSTATES (partie supérieure) permettant de générer les périphériques logiques ICON (partie inférieure)

pour l'utilisateur final. Il commence donc par programmer des interactions avec SWINGSTATES en utilisant des événements virtuels de *Pan* et de *Zoom* (figure 4).

Bob instancie d'abord deux machines à états de type IConStateMachine, une pour le pan et une pour le zoom. Il déclare ensuite les classes d'événements Pan et Zoom de type IConEvent (qui étend la classe VirtualEvent de SWINGSTATES). Dans chacune de ces deux classes, il déclare les données qui composent l'événement en ajoutant des méthodes spécifiques pour y accéder et les modifier. Par exemple, un événement *Zoom* comporte un réel dZ qui correspond au changement d'échelle à appliquer. Dans la classe Zoom, Bob déclare donc un champ dZ et les méthodes getSlotDZ et setSlotDZ.

FLOWSTATES transforme automatiquement chaque machine à états en dispositif ICON selon le mécanisme suivant : à l'instanciation d'une machine, les transitions sont explorées afin de découvrir les événements de type IConEvent. Pour chaque type d'événement, FLOWSTATES crée un groupe de slots d'entrée contenant un slot par couple (getSlot ?,setSlot ?) déclaré dans l'événement. Par exemple, un événement Pan est représenté par un groupe de slots nommé *Pan* et composé de deux slots réels DeltaX et DeltaY (figure 4). Au moment de l'exécution, à chaque fois qu'un slot d'entrée recevra un nouveau signal au niveau d'ICON, un événement sera crée et envoyé à la machine à états. La génération dynamique des événements se fait par des mécanismes d'introspection, qui instancient et initialisent les objets événement via leurs accesseurs.

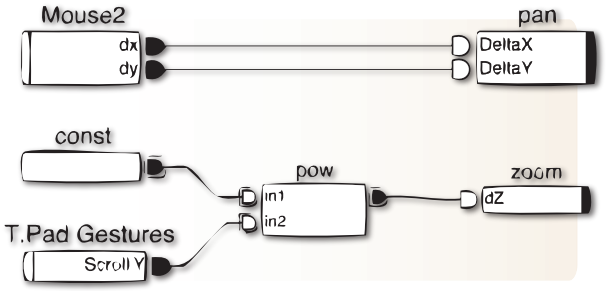

FIGURE *5* : Navigation multi-échelle bimanuelle. Le *Pan* est contrôlé par une souris dédiée, et le *Zoom* est contrôlé par des gestes verticaux sur un Trackpad.

Par défaut, un événement est créé si au moins l'un des slots qui le composent reçoit un signal et si tous les slots ont été initialisés. Des conditions supplémentaires peuvent être spécifiées en surchargeant la méthode occurs de la classe d'événement correspondante. Par exemple, il est possible d'imposer qu'un événement Zoom ne soit généré que lorsque la valeur du champ dZ est strictement positive. Pour construire des conditions plus complexes, FLOWSTATES fournit aussi des mécanismes permettant d'effectuer des tests sur les slots associés, pour déterminer notamment quels slots ont été mis à jour.

Lorsque Bob lance son programme, il voit dans l'éditeur graphique d'ICON ses deux machines à états sous la forme de deux dispositifs (figure 4). Il peut alors brancher ses machines à états sur les entrées physiques qu'il désire, sachant qu'il n'est pas contraint par le système et peut directement commencer à explorer des techniques non-standard. Inspiré par l'article qu'il a étudié en cours sur l'efficacité d'une configuration bimanuelle pour la navigation multi-échelle [7] et disposant d'un ordinateur portable équipé d'un trackpad ainsi que de deux souris, Bob décide alors de tester une configuration à deux mains : le trackpad servira au *zoom*, une des deux souris sera dédiée au *pan*, et l'autre souris contrôlera le pointeur système. Il couple donc le changement de *Zoom* aux gestes verticaux sur son trackpad, et le changement de *Pan* aux déplacements d'une des deux souris (Figure 5). Sur cette figure, le dispositif utilitaire d'ICON "*pow*" (qui réalise l'opération $in1^{in2}$) est utilisé pour transformer des déplacements en changements d'échelle relatifs selon la formule $dZ = 1.1^{scrollY}$.

Nous avons vu dans ce premier exemple les principes de base du contrôle d'une machine à états SWINGSTATES avec un dispositif ICON et l'approche déclarative qui en résulte : le programmeur déclare les événements de haut niveau dont il a besoin pour contrôler les interactions plutôt que de programmer l'interaction autour des événements qui lui sont imposés par le modèle et la boîte à outils (en général très fortement liés aux périphériques d'entrée standard). Il peut ensuite spécifier comment sont émis ces événements par des configurations d'entrée ICON.

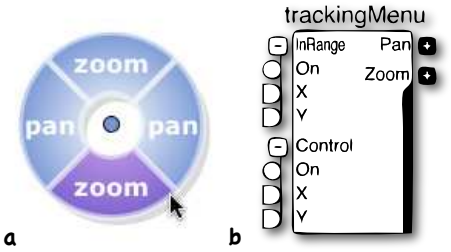

FIGURE *6* **:** Un tracking menu pour les changements d'échelle et les déplacements

Une machine plus complète : le Tracking Menu

Bob est maintenant satisfait de sa configuration bimanuelle, mais réalise que tout le monde ne partage pas sa configuration matérielle. Il décide donc de programmer un *tracking menu* [16] au cas où seulement un seul dispositif de pointage est présent. Un tracking menu est un menu circulaire initialement destiné à être utilisé avec un stylet. Une fois activé, il reste toujours visible et suit le curseur tant que le stylet reste proche de la surface de la tablette (mode 'tracking'). Une commande est sélectionnée au moment où le stylet touche la surface, et un paramètre peut être ensuite ajusté à la manière d'un *control menu* [31]. La figure 6a montre le rendu graphique du menu tandis que la partie droite de la figure 7, extraite de l'article de Fitzmaurice et al. [16], décrit le comportement de ce menu. Le rendu graphique a été programmé avec le graphe de scène de SWINGSTATES et ne sera pas détaillé ici.

La partie gauche de la figure 7 montre le code que Bob a écrit pour reproduire la machine à états de droite. Comme pour les machines habituellement manipulées dans SWINGSTATES, les conditions de déclenchement des transitions sont réparties entre le type de transition, les arguments passés à cette transition et, le cas échéant, la garde.

Il y a deux grandes familles de transitions disponibles dans FLOWSTATES, *Switch* et *Event*, qui permettent de décrire respectivement les *changements de mode* et les *entrées continues*. Jusqu'ici nous n'avons utilisé que des transitions de type *Event*. Un exemple de transition de type *Switch* est donné Fig.7, ligne 11 : la transition stopTracking est déclenchée à chaque fois que le stylet s'éloigne de la surface de la tablette, ce qui correspond à la réception d'un événement InRange dont le booléen on est passé à faux. De manière générale, une transition *Switch* attend en constructeur une classe d'événements qui hérite de SwitchEvent et contient un champ on. En fonction du deuxième argument passé au constructeur (SWITCH_ON ou SWITCH_OFF), la transition sera déclenchée à chaque fois que le champ on de l'événement passe à vrai ou à faux. Comme pour n'importe quel champ d'événement, la sémantique de on est indéfinie : elle dépendra de ce qui lui sera connecté dans ICON.

Nous avons déjà vu précédemment que des événements pouvaient être déclenchés sur des formes graphiques particulières (en utilisant par exemple la classe ClickOnShape). Ceci est également vrai pour les Switch. De manière générale, chaque type de transition possède quatre variantes : une variante *absolue*, et trois variantes *positionnelles* suffixées par OnPosition, OnShape et OnTag. Les transitions positionnelles attendent en argument une classe d'événements qui contient au minimum les champs x et y, interprétés comme une position dans la scène graphique. Ceci permet notamment au développeur de s'affranchir de la gestion du *picking* graphique.

Dans notre exemple, le picking graphique est requis parce que les comportements diffèrent selon l'endroit où les événements surviennent. Par exemple, le menu doit passer en mode *tracking* lorsque le stylet s'approche de la tablette (événement InRange), et il doit être repositionné si le stylet sort du menu. Ce comportement est spécifié Fig.7, lignes 5-6 : la transition startTrackingInMenu ne sera déclenchée que si l'événement survient au-dessus du menu (toutes les formes du tracking menu portent un tag MenuItem), alors que la transition startTrackingOutMenu sera déclenchée quelle que soit sa position. Des conditions plus fines peuvent également être décrites avec des gardes. Par exemple, une garde permet de repositionner le menu lorsque le pointeur se trouve à l'extérieur de celui-ci dans l'état *tracking* (Fig.7, lignes 17-21).

Notons pour finir que notre automate utilise seulement deux classes d'événements : InRange pour le passage de l'état *outOfRange* vers l'état *tracking*, et Control pour le passage de l'état *tracking* vers l'état *touching*. Lorsque la même classe d'événements apparaît dans plusieurs transitions, FLOWSTATES ne crée qu'un seul groupe de slots sur le dispositif ICON (Figure 8). Par exemple, le type d'événements Control, utilisé par plusieurs transitions pour modéliser l'entrée dans le mode *control* et l'interprétation des événements positionnels jusqu'à la sortie du mode (Fig.7, lignes 12-13, 25-26 et 27), ne se retrouve qu'une fois dans le dispositif ICON. Cette convention de nommage permet de spécifier que des transitions sont liées entre elles en termes de logique d'interaction.

Connexion de deux machines

Bob ayant écrit la machine à états du tracking menu, il lui reste à la connecter avec les machines de navigation pan et zoom décrites précédemment. Pour ce faire, il peut utiliser les mécanismes déjà présents dans SWINGSTATES en faisant émettre des événements Pan et Zoom par la machine du menu, qui seront écoutés par les machines de navigation (une machine peut émettre des événements en utilisant sa méthode fireEvent). L'extrait de code suivant abonne les machines de navigation à la machine du tracking menu :

```
1 smTrackingMenu.addStateMachineListener(smPan);
2 smTrackingMenu.addStateMachineListener(smZoom);
```

```
1 IConStateMachine smTrackingMenu = new IConStateMachine("trackingMenu", canvas) {
2    MenuItem currentItem;
3
4    State outOfRange = new State() {
5       Transition startTrackingInMenu = new SwitchOnTag(MenuItem.class, InRange.class, SWITCH_ON, ">> tracking") { };
6       Transition startTrackingOutMenu = new SwitchOnPosition(InRange.class, SWITCH_ON, ">> tracking") {
7         public void action() { menu.showMenu(getPoint()); }
8       };
9    };
10   State tracking = new State() {
11      Transition stopTracking = new Switch(InRange.class, SWITCH_OFF, ">> outOfRange") { };
12      Transition startControl =
13      new SwitchOnTag(MenuItem.class, Control.class, SWITCH_ON, ">> touching") {
14        public void action() { currentItem = (MenuItem)getTag(); }
15      };
16      Transition moveOutMenu = new EventOnPosition(InRange.class) {
17        public boolean guard () {
18          double d = menu.getCurrentPosition().distance(getPoint());
19          return d > menu.getRadius();
20        }
21        public void action() { /* compute dx, dy */ menu.moveBy(dx, dy); }
22      };
23    };
24    State touching = new State() {
25      Transition stopControl =
26      new SwitchOnPosition(Control.class, SWITCH_OFF, ">> tracking") { };
27      Transition control = new EventOnPosition(Control.class) {
28        public void action() { /* perform action associated with current menu item */ }
29    };
30  };
31 };

// classes d'événements
public class InRange extends IConPositionSwitchEvent { }
public class Control extends IConPositionSwitchEvent { }
```

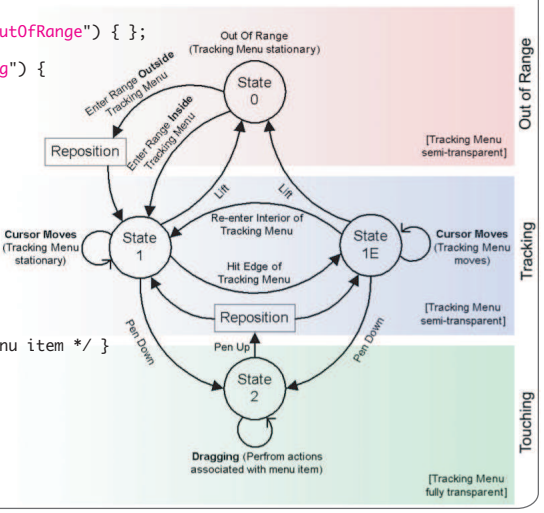

FIGURE **7 :** L'automate décrivant le comportement d'un tracking menu (extrait de [16]), et le code SWINGSTATES correspondant.

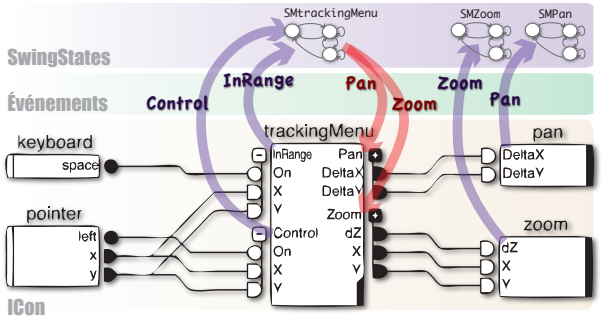

FIGURE ***8*** **:** Connecter les machines via leur représentation en périphériques logiques. Bien que le tracking menu soit prévu pour être utilisé avec une tablette, les entrées utilisées ici sont la souris et la touche espace du clavier pour simuler la proximité du stylet.

Mais FLOWSTATES permet aussi de transformer le menu en un dispositif utilitaire ICON qui pourra être intégré dans le flot de données via le configurateur graphique (Figure 8). Pour cela, Bob surcharge dans un premier temps la méthode getOutputTypes de la machine afin qu'elle retourne le tableau des types d'événements générés par la machine (lignes 2-4 dans le code qui suit). Ainsi, FLOWSTATES peut créer les groupes de slots de sortie adéquats sur le dispositif ICON de la machine (figure 8).

Ensuite, Bob doit écrire le code pour que la machine à états émette des événements de type Pan ou Zoom au cours de l'interaction, c'est-à-dire lors des transitions *control* de l'état *touching* pour notre exemple du tracking menu. Il va utiliser pour cela la méthode fireEvent qui va permettre à FLOWSTATES de mettre à jour les slots de sortie correspondants du dispositif (lignes 9-21 dans le code qui suit). Il est alors possible de connecter la sortie du tracking menu aux interactions Pan et Zoom décrites précédemment, comme l'illustre la figure 8.

```
1 smTrackingMenu = new IConStateMachine([...]) {
2   public Class[] getOutputTypes() {
3     return new Class[] {Pan.class, Zoom.class};
4   }
5   State touching = new State() {
6     [...]
7     Transition control = new EventOnPosition(
8                                  Control.class) {
9       public void action() {
10        if (currentItem.getName().equals("zoom")) {
11          Zoom event = new Zoom();
12          [...]
13          fireEvent(event);
14        } else {
15          if(currentItem.getName().equals("pan")) {
16            Pan event = new Pan();
17            [...]
18            fireEvent(event);
19          }
20        }
21      }
22    };
23  };
24 };
```

Réalisation d'une technique d'interaction générique

Bob est satisfait de son tracking menu et pense que d'autres utilisateurs d'ICON pourraient en profiter. Cependant, sa technique est peu réutilisable dans l'état actuel : les sorties ont été spécialisées pour être dirigées vers les techniques Pan et Zoom. En effet, les valeurs d'entrées sont adaptées dans le code de la machine à états pour produire les bonnes valeur de sorties (conversion des déplacements et calcul du facteur de zoom aux lignes 12 et 17 du code précédent qui ont été masquées pour plus de

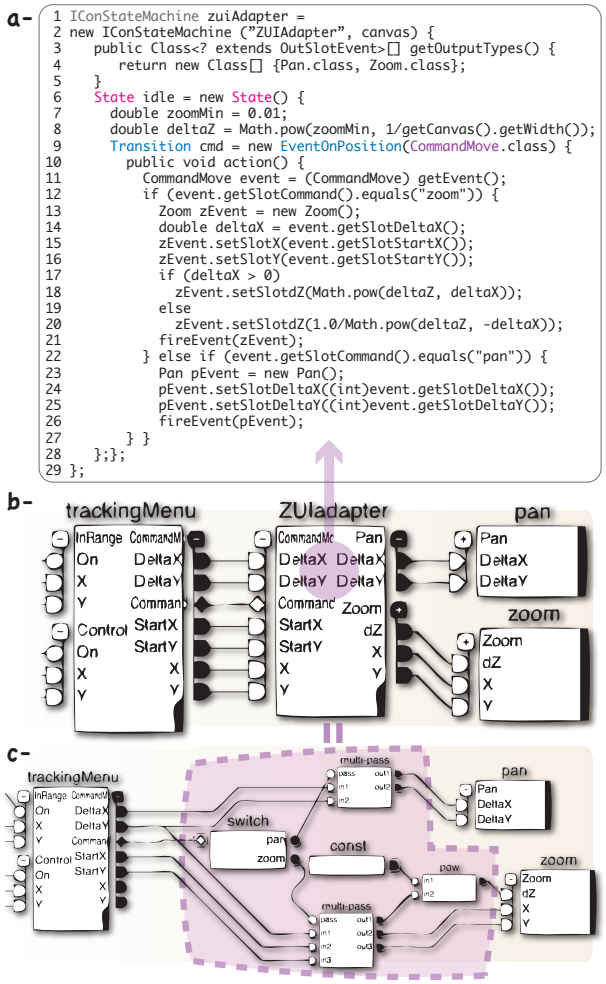

FIGURE ***9*** **:** Les machines à états comme adaptateurs ICon

lisibilité). Ce mécanisme peut cependant être généralisé afin de proposer un instrument "tracking menu générique" qui pourra être branché via ICON à des objets du domaine proposant d'autres opérations que les déplacements et changements d'échelle selon les principes de l'interaction instrumentale [5].

Bob modifie alors légèrement le code de la machine du tracking menu afin que cette dernière n'envoie plus que des événements de la classe générique CommandMove pendant la phase de contrôle de la commande. Un événement CommandMove est composé des champs génériques suivants : DeltaX et DeltaY (déplacement), Command (l'intitulé de la commande), StartX et StartY (la position au moment de la sélection de la commande) et X et Y (la position courante). Le dispositif aura maintenant les slots de sortie correspondants, comme illustré dans la figure 9-b, et pourra être utilisé pour d'autres applications.

Réalisation d'un adaptateur

Maintenant que Bob a réalisé une technique d'interaction générique, il ne lui reste plus qu'à adapter ses sorties aux entrées spécialisées des techniques Pan et Zoom. Il peut

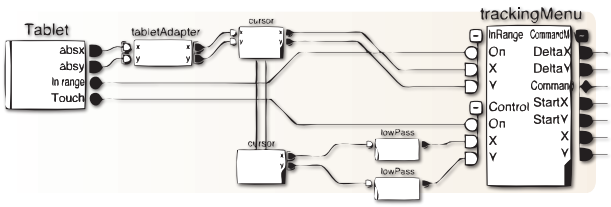

FIGURE ***10*** **:** Ajout d'un comportement de type "trailing widget" avec ICon.

le faire directement au niveau d'ICON, en utilisant des dispositifs utilitaires (Figure 9-c). Cependant, l'approche en flot de données peut s'avérer complexe pour décrire des aiguillages et des contrôles comme requis dans ce cas. Bob préfère décrire ce comportement avec des machines à états, et voudrait également pouvoir l'encapsuler pour pouvoir le distribuer avec sa technique. Il choisit donc de développer un dispositif *ZUIadapter* à partir d'une machine a états (Figures 9-a et 9-b). Le rôle de cette machine est très proche de celui des *transducteurs d'événements* de Accot et al. [2] : il traduit des événements d'un niveau d'abstraction à l'autre. Dans notre exemple, il transforme les événements génériques CommandMove en événements spécifiques Pan et Zoom.

Discussion

Les différents exemples ci-dessus illustrent les possibilités multiples qu'offre FLOWSTATES pour la réalisation de dispositifs ICON à l'aide de machines à états, que ce soit des dispositifs d'interaction spécialisés ou génériques (les deux versions du tracking menu) ou des dispositifs utilitaires (l'adaptateur ZUIadapter). Ils démontrent la flexibilité de FLOWSTATES entre les deux modèles machine à états/flot de données : le programmeur peut choisir où se situe la frontière entre les deux approches en fonction du problème qu'il veut résoudre et de ses besoins.

Pour illustrer à nouveau ce point, Bob ou d'autres utilisateurs de sa technique peuvent facilement doter le tracking menu d'un comportement à la "trailing widget" [17] en insérant un filtre passe-bas entre le périphérique de pointage et le tracking menu dans le flot de données ICON (Figure 10). Le menu va alors "suivre" le pointeur avec un retard qui permet d'interagir avec le fond de l'application (pour dessiner par exemple). Cette fois, le flot de données est plus adapté car implémenter ce comportement avec SWINGSTATES aurait demandé de faire le changement de coordonnées dans chaque transition.

COMPARAISON AVEC L'ETAT DE L'ART

Peu de travaux à notre connaissance ont traité de la combinaison de l'approche à flots de données et de celle dirigée par les états pour la spécification de systèmes interactifs, approches que nous qualifions d'*hybrides*. Le langage Lustre [18] est probablement le plus ancien. Il propose des constructions pour décrire des flots de données synchrones, qui peuvent être mélangées avec des structures

de contrôle classiques mais qui ne sont pas explicitement décrites par des machines à états comme avec FLOWSTATES. Intuikit [9] est un second exemple qui intègre flot de données et contrôle. Dans Intuikit, une interface est un ensemble de composants non seulement capables de produire des événements et d'en consommer (notamment grâce à des machines à états) mais aussi capables d'écouter des changements de propriétés et de changer des propriétés. Flot de données et machines à états sont donc fortement intégrés au sein d'un même composant alors que FlowStates propose une approche plus structurée. Les machines à états peuvent en effet aussi être utilisées pour le contrôle au sein d'un device ICON mais FlowStates permet également au programmeur de s'abstraire de la logique flot de données lors de la programmation du comportement avec une machine à états, l'externalisation vers le flot de données étant faite automatiquement à partir de la machine. Par ailleurs, Lustre comme Intuikit reposent sur des langages purement textuels.

Les travaux les plus proches de notre approche sont le modèle PMIW [22], l'architecture VRMeer [13] et l'environnement qui intègre PetShop et ICON [28].

Dans PMIW / VRMeer, l'interaction est spécifiée en deux parties : la logique de l'interaction est représentée par un formalisme dirigé par les états alors que chaque état peut contenir un ensemble de relations continues entre variables. VRMeer et PMIW couvrent cependant un ensemble restreint d'entrées physiques. Par exemple, PMIW se limite aux événements clavier et souris couverts par le système X Window et les événements d'un Polhemus et du dispositif de suivi de regard ISCAN. PMIW est d'ailleurs plus une preuve de concept qu'un réel outil de développement et de prototypage.

Dans le couple PetShop/ICON, la logique de l'interaction est spécifiée par des réseaux de Petri dans l'environnement graphique de PetShop, et la gestion des dispositifs d'entrée est déléguée à ICON. FLOWSTATES et PetShop visent cependant des objectifs différents : PetShop est principalement dédié à la spécification et à la vérification formelle, alors que FLOWSTATES privilégie la simplicité, parfois au détriment d'une approche plus formelle. En conséquence, nous avons choisi les machines à états qui présentent l'avantage d'être plus faciles à manipuler. En outre, cette logique est programmée en Java dans SWINGSTATES alors qu'elle est spécifiée visuellement dans PetShop. Bien que de manière générale les langages visuels se prêtent bien au prototypage, ils introduisent des couches de traduction qui peuvent rendre difficile l'identification de l'origine d'une erreur dans le processus de prototypage. En effet, PetShop comme PMIW requièrent de spécifier visuellement la logique de l'interaction, puis d'écrire des morceaux de programme pour la lier au noyau fonctionnel. Dans FLOWSTATES, cette couche de traduction est absente. La seule couche de traduction se situe entre le langage visuel d'ICON et la syntaxe de SWINGSTATES. Par contre, nous pensons que le recours à un langage visuel pour le traitement des entrées est pleinement justifié, car comparé à la logique de l'interaction, c'est un aspect qui est très changeant et volatile : les périphériques d'entrée peuvent être très variables d'une machine à l'autre ou nécessiter des reconfigurations au cours de l'exécution d'une application. De plus, cette partie peut être totalement découplée du code du noyau applicatif. Nous avons vu que dans FLOWSTATES, ces deux aspects peuvent être clairement séparés, voire spécifiés par des concepteurs différents.

Pour finir, l'intégration entre SWINGSTATES et ICON va plus loin que l'intégration entre PetShop et ICON. D'une part, FLOWSTATES exploite le graphe de scène de SWINGSTATES et offre donc une prise en charge implicite des mécanismes graphiques. Dans PetShop/ICON, ceux-ci doivent être spécifiés à la main, nécessitant alors la programmation et l'ajout de dispositifs *de service* ICON. D'autre part, les automates de FLOWSTATES peuvent fournir des sorties vers ICON, ce qui permet au programmeur de basculer d'un modèle à l'autre selon sa convenance. Ce dialogue à "double-sens" et cette flexibilité entre les modèles permettent notamment de décrire avec des automates des techniques d'interaction qui sont indépendantes du noyau applicatif, et qui peuvent ensuite être connectées en cascade avec ICON. Cette approche inspirée de l'interaction instrumentale [5] et des architectures à composants offre plus de flexibilité que le modèle à couches classique de Seeheim ou de Arch [1] pour la spécification d'interactions avancées.

CONCLUSION

Les passerelles entre flots de données et machines à états développées dans FLOWSTATES offrent un grand pouvoir d'expression et une grande flexibilité pour le prototypage de l'interaction. SWINGSTATES, basée sur les machines à états, est particulièrement appropriée à la spécification de la logique de l'interaction tandis qu'ICON, basée sur les flots de données et la programmation visuelle, se prête tout à fait à la spécification de relations continues et dynamiques. L'intégration des machines à états dans des flots de données permet non seulement de faciliter la gestion du contrôle et de l'aiguillage mais aussi d'offrir un vocabulaire d'entrée très riche aux machines à états (des périphériques d'entrée standard comme la souris aux périphériques les plus "exotiques" comme ceux de la Nintendo wii en passant par des commandes vocales).

FLOWSTATES (http ://www.lri.fr/∼appert/FlowStates) fait le pont entre deux boîtes à outils sans modifier les paradigmes de programmation sous-jacents de chacune des boîtes à outils dans le souci de favoriser leur adoption par les développeurs qui sont familiers avec l'une et/ou l'autre. Nos pistes futures s'inscrivent dans cette lignée avec l'intégration de la boîte à outils ZVTM [29] pour la programmation du rendu graphique. En effet, si SWINGSTATES propose un graphe de scène adapté à la pro-

grammation de scènes vectorielles, elle n'offre pas les mécanismes fins de rendu au niveau pixel de ZVTM pour la programmation de techniques composant plusieurs vues comme les Sigma Lenses [30]. Nous prévoyons également de tester la couverture et l'utilisabilité de FLOWSTATES grâce à un benchmark incluant la réalisation de techniques d'interaction avancées. Enfin, si le but premier de FLOWSTATES est le prototypage, des langages de spécification en Java comme Biscotti [10] permettraient d'instrumenter les machines à états pour faire de la vérification formelle.

REMERCIEMENTS

Ce travail a bénéficié du soutien de l'ANR - projet iStar.

BIBLIOGRAPHIE

1. A metamodel for the runtime architecture of an interactive system : the UIMS tool developers workshop. *SIGCHI Bull.*, 24(1) :32–37, 1992.
2. J. Accot, S. Chatty, and S. Maury. Formal transducers : Models of devices and building bricks for the design of highly interactive systems. In *Proc. DSV-IS'97*, 143–160. Springer-Verlag, 1997.
3. C. Appert and M. Beaudouin-Lafon. SwingStates : Adding state machines to Java and the Swing toolkit. *Software : Practice and Experience*, 38(11) :1149 – 1182, 2008.
4. R. Ballagas, M. Ringel, M. Stone, and J. Borchers. iStuff : a physical user interface toolkit for ubiquitous computing environments. In *Proc. CHI'03*, 537–544. ACM, 2003.
5. M. Beaudouin-Lafon. Instrumental interaction : an interaction model for designing post-WIMP user interfaces. In *Proc. CHI'00*, 446–453. ACM, 2000.
6. R. Blanch and M. Beaudouin-Lafon. Programming rich interactions using the hierarchical state machine toolkit. In *Proc. AVI'06*, 51–58. ACM, 2006.
7. F. Bourgeois and Y. Guiard. Multiscale pointing : facilitating pan-zoom coordination. In *Proc. CHI EA'02*, 758–759. ACM, 2002.
8. W. Buxton. A three-state model of graphical input. In *INTERACT*, volume 90, 449–456, 1990.
9. S. Chatty, A. Lemort, and S. Vales. Multiple input support in a model-based interaction framework. In *Proc. TABLETOP '07*, 179–186, Oct. 2007.
10. C. D. T. Cicalese and S. Rotenstreich. Behavioral specification of distributed software component interfaces. *Computer*, 32(7) :46–53, 1999.
11. Cycling '74. max/msp/jitter. `http://www.cycling74.com/`.
12. Dassault Systemes. Virtools Dev. `www.virtools.com/`.
13. G. de Haan and F. H. Post. Flexible architecture for the development of realtime interaction behavior. In *Workshop VR'08*. IEEE Computer Society, 2008.
14. P. Dragicevic and J.-D. Fekete. Support for input adaptability in the ICON toolkit. In *Proc. ICMI'04*, 212–219. ACM, 2004.
15. P. Figueroa, M. Green, and H. J. Hoover. InTml : a description language for VR applications. In *Proc. Web3D '02*, 53–58. ACM, 2002.
16. G. Fitzmaurice, A. Khan, R. Pieké, B. Buxton, and G. Kurtenbach. Tracking menus. In *Proc. UIST'03*, 71–79. ACM, 2003.
17. C. Forlines, D. Vogel, and R. Balakrishnan. Hybrid-pointing : fluid switching between absolute and relative pointing with a direct input device. In *Proc. UIST'06*, 211–220. ACM, 2006.
18. N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. *Proc. of the IEEE*, 79(9) :1305–1320, 1991.
19. K. Hinckley, M. Czerwinski, and M. Sinclair. Interaction and modeling techniques for desktop two-handed input. In *Proc. UIST'98*, 49–58. ACM, 1998.
20. S. Hudson, J. Mankoff, and I. Smith. Extensible input handling in the subArctic toolkit. In *Proc. CHI'05*, 381–390. ACM, 2005.
21. S. Huot, C. Dumas, P. Dragicevic, J.-D. Fekete, and G. Hégron. The MaggLite post-WIMP toolkit : Draw it, connect it and run it. In *Proc. UIST'04*, 257–266. ACM, 2004.
22. 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.
23. W. A. König, R. Rädle, and H. Reiterer. Squidy : a zoomable design environment for natural user interfaces. In *Proc. CHI EA'09*, 4561–4566. ACM, 2009.
24. Meso Group. vvvv : a multipurpose toolkit. `http://vvvv.org/`.
25. B. Myers. Separating application code from toolkits : eliminating the spaghetti of call-backs. In *Proc. UIST'91*, 211–220. ACM, 1991.
26. B. Myers, D. Giuse, R. Dannenberg, B. Zanden, D. Kosbie, E. Pervin, A. Mickish, and P. Marchal. Garnet : Comprehensive support for graphical, highly interactiveuser interfaces. *Computer*, 23(11) :71–85, 1990.
27. B. Myers, R. McDaniel, R. Miller, A. Ferrency, A. Faulring, B. Kyle, A. Mickish, A. Klimovitski, and P. Doane. The Amulet environment : new models for effective user interfacesoftware development. *IEEE Trans. Soft. Eng.*, 23(6) :347–365, 1997.
28. D. Navarre, P. Palanque, P. Dragicevic, and R. Bastide. An approach integrating two complementary model-based environments for the construction of multimodal interactive applications. *Interact. Comput.*, 18(5) :910–941, 2006.
29. E. Pietriga. A toolkit for addressing hci issues in visual language environments. In *Proc. VL/HCC'05*, 145–152. IEEE Computer Society, 2005.
30. E. Pietriga and C. Appert. Sigma lenses : focus-context transitions combining space, time and translucence. In *Proc. CHI'08*, 1343–1352. ACM, 2008.

31. S. Pook, E. Lecolinet, G. Vaysseix, and E. Barillot. Control menus : excecution and control in a single interactor. In *Proc. CHI EA'00*, 263–264. ACM, 2000.
32. SENSE8. World Up. `http://www.sense8.com/`.