Le modèle d'architecture logicielle H^4 : Principes, usages, outils et retours d'expérience dans les applications de conception technique

The H^4 Software Architecture Model: Principles, Applications, Tools and Experience feedbacks with Computer-Aided Design Software

DEPAULIS Fabrice[1,2], JAMBON Francis[1,3], GIRARD Patrick[1] et GUITTET Laurent[1]

(1) LISI/ENSMA, 1 av. C. Ader, BP 40109, 86961 Futuroscope cedex, France
(2) LIPSI/ESTIA, Technopole Izarbel, 64210 Bidart, France
(3) CLIPS-IMAG/MultiCom, BP 53, 38041 Grenoble cedex 9, France

Résumé. Les différents modèles d'architecture logicielle pour les applications interactives, proposés depuis deux décennies, peuvent être classés en trois grands groupes. Les modèles globaux (e.g. Seeheim, Arch) ont permis de donner un premier cadre de référence universellement admis, mais demeurent d'un niveau d'abstraction trop élevé pour être utilisés seuls efficacement. Les modèles à agents (e.g. PAC, MVC, …) intègrent la dimension objet, très largement utilisée aujourd'hui, et répondent à l'objection précédente ; cependant, ils présentent de nombreuses difficultés d'utilisation dans les applications réelles. Les modèles hybrides (e.g. PAC-Amodeus) ont été conçus pour profiter du meilleur de ces deux catégories. Nous présentons dans cet article un modèle de ce type, dénommé H^4, ainsi appelé pour ses quatre hiérarchies. Nous décrivons la sémantique de H^4, et montrons, à travers plusieurs outils et applications reposant sur ce modèle, comment il peut être exploité tant dans le domaine de l'ingénierie de l'interaction homme-machine que dans celui de la validation des systèmes interactifs.

Mots-clés. Architecture logicielle, dialogue structuré, modélisation du dialogue, validation.

Abstract. The various software architecture models for interactive applications, proposed for two decades, could be classified in three groups. The global models (e.g. Seeheim, Arch) proposed a first framework universally accepted nowadays as a reference, but these models remain at a too high level of abstraction to be used alone effectively. The agents models (e.g. PAC, MVC, ...) integrate the object dimension. They are very largely used today, and answer to the preceding objection. However, they are difficult of use in real-world applications. The hybrid models (e.g. PAC-Amodeus) were designed to benefit from the best of the two former categories. We present in this article a model from this type, called H^4 (four hierarchies). We describe the semantics of H^4, and show, through several tools and applications based on this model, how it can be used both for man-machine interaction design and interactive systems validation.

Keywords. Software architecture, structured dialogue, dialogue modeling, validation.

Introduction

L'étude des architectures logicielles est un domaine relativement récent, datant de la fin des années 80 et du début des années 90. Dans le domaine de l'interaction homme-machine (IHM), il a donné lieu à des travaux importants, motivés par la nécessité de prendre en compte dans le processus de développement logiciel les particularités de l'interaction entre l'utilisateur et le système.

La simple utilisation de techniques nouvelles, comme les techniques graphiques, a motivé tout naturellement l'émergence de modèles permettant d'identifier l'interface homme-machine comme une entité externe au reste de l'application, dénommé « Noyau Fonctionnel » ou encore « Noyau Sémantique ». C'est ainsi que le premier modèle d'architecture consacré à l'IHM, et universellement reconnu aujourd'hui, le modèle de Seeheim (Pfaff, 1985), considère l'interface homme-machine comme une entité totalement externe au noyau fonctionnel, décomposée en trois modules (Présentation, Contrôleur de Dialogue, et Interface avec l'Application) selon une métaphore très langagière. En effet, la Présentation correspond à la couche lexicale, le Contrôleur de Dialogue à la couche syntaxique et l'Interface avec l'Application à la couche sémantique du langage d'interaction entre l'utilisateur et le système. Ce modèle, fondateur d'une terminologie employée universellement aujourd'hui (noyau fonctionnel, contrôleur de dialogue, présentation), a été revisité au début des années 90 pour donner lieu au modèle « Arch » (Uims, 1992). Celui-ci adresse cette fois l'ensemble de l'application interactive (on intègre le noyau fonctionnel dans l'architecture), et prend en compte l'arrivée des nouveaux outils que sont les « Toolkits », véritables boîtes à outils de l'interaction, développant le concept de « Widgets », objets élémentaires d'interaction. Ces architectures logicielles, qualifiées souvent de globales, ne permettent cependant d'obtenir qu'une macro-décomposition de l'application, très utile pour une première approche, mais qui se révèle insuffisante lorsque l'application croît en taille. De plus, elles n'intègrent pas les approches objet, qui sont aujourd'hui omniprésentes.

En parallèle avec ces modèles, se sont développées des approches multi-agents, qui ont proposé de modéliser l'application au moyen d'agents réactifs, souvent appelés interacteurs depuis. Les modèles à agents sont nombreux (Coutaz et Nigay, 2001) et diffèrent essentiellement sur la répartition des rôles et des responsabilités entre agents. Nous n'illustrerons ici ces propos que par deux exemples, PAC (Coutaz, 1990) et MVC (Krasner et Pope, 1988). Les agents se composent traditionnellement de facettes, permettant de décomposer au niveau même de l'agent les services rendus ; on retrouve ainsi des services sémantiques (*Abstraction* pour PAC, *Model* pour MVC), des services d'échange avec l'utilisateur (*Présentation* pour PAC, décomposé en deux facettes, *View* et *Controller*, pour MVC), et des services de coordination entre facettes et autres agents (*Contrôle* pour PAC, *Model* et *Controller* pour MVC[1]). Ces modèles peuvent également, comme dans le cas de PAC, préconiser une macro-décomposition de l'application fondée sur différents rôles des agents, permettant par exemple de prendre en compte une décomposition basée sur les modèles de tâches. En effet, l'un des plus gros problèmes des

[1] Dans le cas de MVC, le *Controller* pilote la vue et peut avoir des interactions avec d'autres *Controllers* dans le cadre strict du dialogue interactif, alors que le *Model* pilote indirectement les ensembles *Control/View* à travers un mécanisme d'alerte (appelé originellement « dependence », et qui correspond en fait à la mise en œuvre du pattern *Observer/Observable*) et dialogue éventuellement avec d'autres *Models*.

architectures consiste à déterminer précisément les agents nécessaires à la réalisation de l'application.

Les architectures globales ou multi-agents ont une caractéristique commune, celle d'être de style homogène (Coutaz et Nigay, 2001), ce qui leur confère une relative simplicité, mais qui, à l'inverse, ne simplifie pas leur utilisation. De ce constat, sont nées des approches qualifiées d'hybrides, permettant d'allier les bénéfices des deux classes précédentes, dont l'exemple le plus illustratif est certainement PAC-Amodeus (Nigay, 1994). Hybride des modèles Arch et PAC, PAC-Amodeus propose de considérer le contrôleur de dialogue du modèle Arch comme une hiérarchie d'agents PAC. Cette dernière peut ainsi aisément refléter la hiérarchie des tâches de l'utilisateur, et sa connexion avec les deux modules adjacents se fait naturellement au travers des facettes correspondantes (*Abstraction* pour l'adaptateur de noyau fonctionnel, et *Présentation* pour la présentation abstraite). Nous verrons par la suite que le modèle H⁴, objet de cet article et contemporain de PAC-Amodeus, cherche comme lui à résoudre des problèmes d'expressivité et de complétude laissés en suspens dans les approches globales et multi-agents. La motivation première de la conception de ce modèle réside dans le besoin de concevoir un modèle d'architecture adapté à une classe d'applications à la problématique particulière, celle des applications de conception technique. En effet, la multitude d'objets et de relations entre objets, mais également la nature des tâches concernées par ces applications (cf. section 1.1) rendent les modèles décrits ci-dessus inopérants (cas des modèles à agents) ou insuffisants (cas des modèles globaux). Dans le premier cas, ils entraînent une explosion des échanges entre agents, alors que dans le second, ils délèguent trop de notions à l'intérieur des macro-composants.

Durant cette courte présentation des modèles d'architecture (on se réfèrera à (Coutaz et Nigay, 2001) pour une analyse plus fine de ces modèles), nous n'avons parlé que d'architectures conceptuelles pour les applications interactives. Cela ne signifie pas que ces architectures ne sont pas applicables au niveau implémentationnel. En effet, elles sont utilisées de façon plus ou moins généralisées dans de nombreux contextes. Ainsi, MVC est-il aujourd'hui considéré comme un motif de conception (« Design Pattern ») (Gamma *et al.*, 1994) et utilisé comme tel, même si son application peut parfois amener à des distorsions du modèle, comme c'est le cas par exemple dans Swing, la boîte à outils graphique de Java. Cependant, rares sont les travaux qui ont formalisé le passage des architectures conceptuelles aux architectures implémentationnelles. Encore plus rares sont les outils gérant cette transformation. AMF (Tarpin-Bernard et David, 1999), système multi-agent basé sur un nombre non fixe de facettes, en est ainsi un exemple. La seconde motivation pour la création du modèle H⁴ consiste en la définition de déclinaisons implémentationnelles, accompagnées si possible d'outils adaptés. Cette dimension a donné naissance à des approches comme les interacteurs hiérarchisés (Girard *et al.*, 1995), ou les *Diagets* (présentés ci-après).

Afin de limiter la taille de cet article, nous ne parlerons pas ici des travaux sur l'utilisation des méthodes formelles dans les IHM, qui ont, de façon conjointe avec les travaux sur les architectures, cherché à définir formellement la notion d'interacteur, puis à proposer des méthodes pour modéliser formellement le dialogue des applications interactives (Jambon, 2002). Pourtant, ces approches s'avèrent très utiles, voire indispensables, pour atteindre le deuxième objectif que nous nous étions fixé. C'est pourquoi l'étude des possibilités de description formelle du dialogue des applications interactives a-t-elle été intégrée à l'ensemble de la réflexion menée autour du modèle H⁴.

Cet article est structuré en trois parties. Dans la première, nous présentons en détail le contexte initial, celui des applications de conception technique, point de départ de notre travail. Ceci nous amènera à identifier les points de complexité des applications interactives. Dans la deuxième partie, nous présentons en détail le modèle H4, et l'illustrons au travers de son premier modèle d'implémentation, les interacteurs hiérarchisés. Dans la troisième partie, nous donnons quelques exemples de l'utilisation de ce modèle, qui dépassent le cadre du domaine du contexte de départ. Enfin, quelques perspectives sont ensuite évoquées.

1 Contexte

Les Applications Graphiques Interactives de Conception Technique (AGICT) proposent de mettre au service d'un expert d'un domaine les outils nécessaires pour que celui-ci soit en mesure de réaliser, de la manière la plus précise possible, toutes les opérations désirées pour atteindre un objectif donné. Laurent Guittet (Guittet, 1995) définit une AGICT en ces termes : « une AGICT est une application permettant de créer une représentation informatique d'un objet technique susceptible d'être manipulée par des fonctions programmées de simulation ou de production » (tout au long de cet article, nous illustrerons nos propos avec des systèmes de CAO, qui constituent les AGICT les plus connues).

Dans la majorité des applications « grand public », comme Paint™ ou MacDraw™, la construction se fait par des techniques approximatives de manipulation directe ; typiquement, on créera un cercle à l'aide d'une technique de « drag », puis on le positionnera en le déplaçant à l'aide de la souris (drag&drop). Au mieux pourra-t-on bénéficier d'une aide au positionnement sous la forme d'une matérialisation de relations d'alignements, comme dans Keynote™ d'Apple. Au contraire, une AGICT impose de manipuler des contraintes précises, spécifiques au domaine et indispensables au concepteur. Par exemple, un système de CAO doit pouvoir proposer la possibilité de construire un cercle en spécifiant trois droites qui lui sont tangentes, ce qui ne peut se faire en aucun cas à l'aide des techniques décrites ci-dessus.

Pour comprendre les spécificités du dialogue en conception technique ainsi que les raisons qui nous ont poussés à créer une architecture logicielle dédiée, dans un premier temps, aux AGICT, nous revenons dans cette partie sur les caractéristiques des applications interactives, à partir de la classification de (Pierra, 1995).

1.1 Classification des applications interactives

La taxonomie des systèmes interactifs établie dans (Pierra, 1995) met en évidence les points particuliers d'un domaine d'application. Dans ce récapitulatif, nous utilisons le terme tâche en qualifiant ainsi un but utilisateur associé à un ensemble de fonctions du système[2].

Arité des tâches

On dit d'une application qu'elle est mono-objet si les tâches qu'elle utilise ne requièrent qu'un seul objet du modèle pour s'exécuter. Par exemple, les applications Paint™ ou MacDraw™ sont mono-objet : toutes les actions possibles portent sur un seul objet du modèle (dessiner un cercle, changer la couleur d'un rectangle, etc.). Ces applications permettent également d'effectuer des actions sur des groupes

[2] Pour aller plus loin dans la notion de tâche, le lecteur pourra se référer à (Scapin et Bastien, 2001) et (Diaper et Stanton, 2003).

d'objets, mais dans ce cas, l'action sur le groupe revient sémantiquement à effectuer une même action mono-objet successivement sur chacun des objets du groupe. Les applications mono-objet s'opposent naturellement aux applications dites multi-objets. Celles-ci comportent des tâches qui peuvent utiliser plusieurs objets du système pour s'exécuter. Les applications de CAO sont multi-objets. Par exemple, « construire un cercle tangent à trois droites » est une tâche multi-objets puisqu'elle met en jeu trois objets du modèle. Comme l'indique (Martin, 1995), les utilisateurs de systèmes de CAO sont habitués à utiliser des relations sous forme de contraintes ou d'opérateurs géométriques.

Structure des tâches

Une tâche est dite atomique si elle est indépendante de l'exécution de toute autre tâche. Le résultat est enregistré par l'objet du domaine. Par opposition, une tâche structurée peut utiliser le résultat d'une autre tâche comme paramètre. Le résultat de son exécution n'est connu que lorsque l'utilisateur a donné la hiérarchie complète tâches/sous-tâches qui la compose. Cette décomposition hiérarchique correspond à l'analyse donnée par Norman (Norman, 1986), expliquant qu'un utilisateur est plus apte à résoudre un problème quand il l'a dissocié en sous-buts plus accessibles.

Une AGICT est caractérisée par le fait qu'elle est capable de supporter des tâches structurées. Les différents éléments du modèle sont en effet liés par plusieurs types de relations : des relations numériques (le diamètre d'un cercle vaut deux fois la longueur de ce segment), géométriques (le centre de ce cercle est situé à l'intersection de ces deux segments) ou grapho-numériques (le diamètre du cercle à créer vaut deux fois la distance entre ces deux points). Celles-ci sont connues de l'utilisateur et elles doivent être exploitables durant la phase de conception.

De ce fait, les tâches doivent être structurées et une tâche doit pouvoir utiliser le résultat d'une autre tâche comme paramètre (par exemple, la création d'un segment doit pouvoir utiliser le résultat du calcul de la distance entre le centre de deux cercles). Comme nous l'écrivons dans l'introduction de cette section, dans une AGICT, les approximations ne sont pas satisfaisantes : on veut être capable de faire appel à des contraintes de construction précises (tel segment est *tangent* à tel cercle, par exemple), tout en conservant un mode d'expression simple.

Structure des objets

Les objets du domaine de conception peuvent être considérés comme structurés lorsque plusieurs niveaux sont accessibles à l'utilisateur. Au contraire, les objets sont dits simples s'ils ne peuvent pas être constitués d'autres objets du modèle. Dans le second cas, la désignation et son écho peuvent être réalisés par la couche de « présentation », chargée des interactions directes avec l'utilisateur. En revanche, dans le cas d'objets structurés, cette interprétation ne peut se faire que par le composant « domaine », porteur de la sémantique de l'application. La délégation sémantique de cette interprétation dans un moteur 3D permet de résoudre partiellement ce problème, mais ne dispense pas d'un fort couplage entre ce dernier et les objets du noyau fonctionnel.

Dans les AGICT, les objets sont fortement structurés. Ainsi, pour un système de CAO, un cube est-il constitué de faces, elles-mêmes organisées en arêtes, etc.

Relations entre objets

Les objets du domaine sont dits relationnels si la représentation d'un objet dépend d'autres objets du domaine (par opposition à « autonomes »). Dans ce cas, la modification de l'un peut entraîner la modification des autres. À l'inverse, si un

objet est autonome, il peut être mis en bijection avec un objet unique d'interaction chargé entre autre de sa représentation (Duke et Harrison, 1993).

D'après cette définition, les entités manipulées dans une AGICT comme un système de CAO sont relationnelles. Par exemple, un objet mécanique correspond le plus souvent à un assemblage de sous-objets, reliés entre eux par des liaisons éventuellement mobiles, ce qui définit des contraintes précises.

Conclusion sur la classification

D'après cette classification, une application de CAO est multi-objets, possède des tâches structurées et des objets composés et relationnels. Forts de cette constatation, nous analysons dans la section suivante le dialogue mis en place dans des applications de ce type.

Stratégies de dialogue en conception technique

La nature structurée des tâches implique un mode de dialogue particulier. On distingue deux types de langage, correspondant à deux méthodes différentes pour décrire une tâche.

Dans un système utilisant un dialogue basé sur un langage post-fixé, l'utilisateur doit d'abord fournir l'objet sur lequel porte la tâche avant de désigner l'action elle-même. Par exemple, il va désigner un objet puis activer une commande de changement de couleur qui va agir sur cet objet. Ce type de dialogue, utilisé notamment en manipulation directe, se prête uniquement aux systèmes supportant des tâches atomiques mono-objet. En effet, une fois l'objet de l'action sélectionné, il n'est pas concevable d'en sélectionner un deuxième dans le but de lui fournir le résultat d'une opération ayant eu lieu sur le premier. Un tel dialogue ne répond donc pas aux besoins d'un système devant supporter des tâches structurées. Cette stratégie de dialogue a néanmoins été explorée dans des travaux relatifs à H[4] (Patry, 1999) que nous ne détaillerons pas ici.

A l'inverse, les langages préfixés reposent sur une logique « commande - opérande ». Dans un système basé sur un mécanisme de ce type, l'utilisateur précise d'abord l'opération qu'il veut effectuer (par exemple en sélectionnant un item de menu, ou en appuyant sur le bouton associé à l'opération) puis seulement les paramètres de la tâche. Comme nous l'avons montré (Guittet, 1995), il s'agit là du seul type de dialogue utilisable lorsque les tâches d'un système sont structurées. En effet, dans ce mécanisme, un opérande peut très bien être remplacé par une tâche capable de produire l'information attendue.

1.2 Dialogue structuré

Décomposition des tâches selon Norman

Le dialogue structuré se base sur les résultats de Norman : "Breaking down goals into sub-goals is the natural way for users to solve problems. This decomposition is done recursively until actions of the system may be reached." (Norman, 1986).

Il met en pratique la décomposition des buts de l'utilisateur en un ensemble hiérarchique de buts/sous-buts, correspondant directement à l'arbre des tâches nécessaire pour atteindre un but. Ainsi, mettre en place un tel dialogue revient à fournir à l'utilisateur le moyen de calquer son arbre des tâches sur celui des buts/sous-buts. Les paramètres des tâches sont exprimés au moyen d'autres tâches et une tâche peut être exécutée dans le contexte d'une autre tâche.

A titre d'exemple la tâche « créer un segment dont une extrémité est le centre d'un cercle et l'autre l'intersection de deux segments » peut être décomposée en

« récupérer le centre d'un cercle », « récupérer l'intersection de deux segments » et « créer un cercle par deux positions ».

La référence aux définitions de Norman peut sembler curieuse dans ce contexte. En effet, on a plutôt l'habitude de réserver l'analyse de Norman aux tâches de l'utilisateur, souvent abstraites, et de décrire au niveau du système des « commandes » ou des « actions ». Il nous semble que, au moins dans le cas des AGICT, on peut faire redescendre l'analyse de Norman au niveau du système, et ainsi analyser de façon continue d'activité de l'utilisateur. Dans la suite de cette section, nous utiliserons le terme de « tâche » pour parler de l'ensemble direct action/réaction de l'utilisateur sur le système. De plus nous nous intéresserons exclusivement à l'impact de la description de ces tâches sur le dialogue de l'application.

Mise en œuvre du dialogue structuré

Mettre en oeuvre la décomposition des tâches selon Norman à l'intérieur d'une architecture logicielle nécessite la prise en compte de ses caractéristiques particulières.

La première notion qui découle naturellement de cette méthode est celle, couplée, de production/consommation et de hiérarchisation des tâches. Certaines tâches produisent des informations qui sont récupérées et utilisées comme paramètres par d'autres (dans l'exemple du paragraphe précédent, les tâches « centre d'un cercle » et « intersection de deux segments » fournissent des informations qui peuvent être utilisées par « créer un segment »). Cela induit que le contrôleur de dialogue de l'application doit être capable de différencier les tâches de production, qui fournissent des informations, des tâches de consommation, qui utilisent des informations[3]. La distinction présentée plus haut entre tâches atomiques et tâches structurées n'est pas suffisante ici. D'une part, nous appelons tâches terminales les tâches identifiées comme pouvant être associées à des objectifs de haut niveau (au sens de Norman) de l'utilisateur. Celles-ci modifient l'état du Noyau Fonctionnel mais ne produisent pas de données dans le cadre du dialogue. Ces tâches sont généralement structurées. D'autre part, nous appelons sous-tâches de production les tâches calculant et fournissant un résultat à d'autres tâches. Ces tâches peuvent être atomiques, ou elles-mêmes structurées.

Une conséquence de cette distinction entre tâches terminales et tâches de production est que les différentes tâches d'un système peuvent être regroupées en différentes catégories, suivant les services qu'elles fournissent à l'application. Dans la plupart des systèmes, on en distingue au moins trois :

- les tâches de création, qui modifient l'état du système en créant de nouveaux objets ;
- les tâches de calcul, qui sont chargées de récupérer la valeur de l'attribut d'un objet ou d'effectuer un certain nombre de calculs sur ces attributs ;

[3] Précisons que cette notion de production/consommation fait exclusivement référence au dialogue homme-machine. Il est certain qu'une tâche permettant la création d'une entité géométrique produit quelque chose pour l'application. Mais sa « production » est destinée au noyau fonctionnel. En revanche, une tâche qui permet de désigner un point existant dans le noyau fonctionnel pour qu'il puisse servir de base à la construction d'une nouvelle entité ne « produira » rien pour ce même noyau fonctionnel, alors qu'elle sera considérée comme une tâche de production pour le dialogue (dans la suite du dialogue, on se servira de cet objet). On peut assimiler ces tâches de production à des fonctions sans effet de bord, permettant de construire des expressions, alors que les tâches de consommation doivent plutôt être vues comme des sous-programme avec ou sans effet de bord.

- les tâches de sélection, qui permettent de fournir une entité du modèle (sélectionnée par exemple avec le curseur de la souris) aux autres tâches.

Le dernier élément primordial dans la constitution d'un dialogue structuré concerne l'information qui transite de tâche productrice en tâche consommatrice. Cette information est identifiée par son type et est indépendante de son mode de production. Comme l'analyse Guillaume Texier (Texier, 2000), l'expression d'un dialogue structuré nécessite donc :

- l'identification des unités d'information indépendamment de leur mécanisme de production. De telles unités d'information sont appelées jetons ; on peut voir ici une analogie avec les jetons des réseaux de Petri. Cependant, ces jetons contiennent des valeurs, et s'assimilent plutôt à des jetons colorés, ou même des jetons de réseaux de Petri à objets.
- la description des tâches en termes de production/consommation et la détermination de leur catégorie, puis l'organisation de ces tâches entre elles, ce qui définit la syntaxe du langage de dialogue reconnu par le contrôleur de dialogue.

Conclusion sur le dialogue structuré

Nous avons récapitulé ici les différents éléments qui constituent un dialogue structuré. Comme cela est déjà analysé dans (Guittet, 1995), (Gardan *et al.*, 1988), (Gardan *et al.*, 1993) et (Martin, 1995), la mise en oeuvre de cette méthode nécessite une architecture particulière. Il faut ainsi que le contrôleur de dialogue d'une application devant supporter le dialogue structuré :

- soit doté d'une unité d'information qui transite d'une tâche à l'autre ;
- identifie, du point de vue du dialogue de l'application, les tâches de production et les tâches terminales ;
- permette le regroupement des tâches pour favoriser la mise en œuvre de la stratégie de l'utilisateur.

Tous ces éléments ont servi de base à la constitution du modèle d'architecture H4. Dans la partie suivante, nous étudions ce modèle en décrivant ses composants principaux et en détaillant le fonctionnement de son contrôleur de dialogue.

1.3 AGICT et couches logicielles

Une dernière particularité des AGICT concerne leur réalisation dans un environnement logiciel composé d'outils variés. Alors que la majorité des applications WIMP[4] actuelles s'appuient de façon exclusive sur une boîte à outils standard (Java Swing, Qt, MFC, etc.), le domaine des AGICT foisonne d'outils plus ou moins complexes permettant de faciliter le développement essentiellement graphique. Ainsi, les différentes bibliothèques (2D, 3D) et les modules spécifiques, partiellement ou totalement optimisés par rapport aux cartes graphiques des ordinateurs (moteurs de rendu réaliste par exemple) doivent-elles s'insérer harmonieusement dans l'architecture logicielle. La délégation sémantique est ainsi souvent nécessaire pour tirer parti de ces spécificités et optimisations. Ces points doivent être considérés dans la définition de l'organisation des couches logicielles.

2 Le Modèle H4

H4 est un modèle d'architecture logicielle qui a été développé au laboratoire LISI/ENSMA, notamment par (Guittet et Pierra, 1993), (Girard *et al.*, 1995) et (Guittet, 1995). Sa conception a été motivée par le besoin des AGICT, et

[4] Window, Icons, Menus and Pointers

notamment des systèmes de CAO, de disposer d'un dialogue structuré. Ce modèle est basé sur ARCH (Uims, 1992) et distingue comme lui cinq composants logiciels. Mais, à la différence de ce dernier, H⁴ décrit de manière précise tous les modules, et préconise la structuration de quatre d'entre eux en hiérarchies d'agents.

2.1 Macro Composants

De la même manière que ARCH, le modèle H⁴ est composé de cinq composants, dont les dénominations diffèrent légèrement (Figure 1).

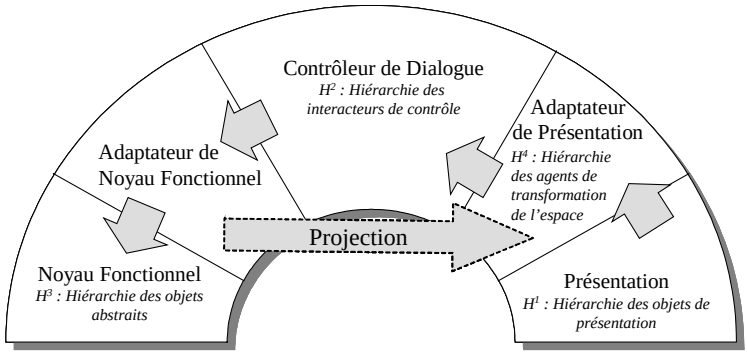


Figure 1. *Les différentes couches du modèle H⁴.*

Le noyau fonctionnel représente les objets du domaine ainsi que toutes les fonctionnalités qui y sont associées. Il représente la véritable sémantique de l'application. Il supporte la hiérarchie des objets de l'application.

L'adaptateur de noyau fonctionnel est une surcouche logique du noyau fonctionnel. Il s'agit d'une interface entre le contrôleur de dialogue et le noyau fonctionnel qui rend ces deux éléments indépendants entre eux. Il est donc chargé de traduire les informations du noyau fonctionnel en données compréhensibles par le contrôleur de dialogue et inversement. D'autre part, il est capable de communiquer avec l'adaptateur de présentation pour afficher l'état du noyau fonctionnel, quand le passage par le contrôleur de dialogue n'a pas de réelle signification.

L'adaptateur de présentation est chargé de rendre l'application indépendante de l'implémentation de l'interface graphique en mettant à la disposition du contrôleur de dialogue une surcouche des objets et des primitives de la présentation (en fait, la boîte à outils de programmation d'interfaces). Il fournit donc à l'application les outils permettant de réaliser l'affichage de l'état du noyau fonctionnel. On peut y trouver une hiérarchie d'agents de transformation d'espace, par exemple dans le cas des applications 3D ; c'est ici également que l'on trouvera une partie importante de la délégation sémantique, ainsi que des techniques plus particulières comme le rendu réaliste par exemple.

La présentation gère les entrées et les sorties avec l'utilisateur. Par l'intermédiaire de composants d'interaction (les *widgets* d'une boîte à outils), naturellement organisés sous forme de hiérarchie, cette couche logicielle récupère les informations fournies par l'utilisateur et affiche l'état du noyau fonctionnel. Ce composant correspond au composant Interaction de ARCH.

2.2 Contrôleur de dialogue

Contrairement à d'autres modèles d'architecture, la spécification de H⁴ définit précisément le rôle, la composition et le fonctionnement du contrôleur de dialogue.

Pour cela, nous passons du niveau de modèle conceptuel à celui de modèle implémentationnel (Coutaz et Nigay, 2001).

Dans ce paragraphe, nous décrivons les différents composants qui le constituent avant de détailler son fonctionnement. Ces composants découlent tout naturellement des caractéristiques analysées en partie 1 de cet article, et jugées nécessaires à l'instauration d'un dialogue structuré au sein d'une application.

Jetons

L'unité d'information est le jeton. On distingue les jetons de type « paramètre » des jetons de type « commande ». Les premiers représentent une abstraction des données du noyau fonctionnel et de la présentation. Un jeton paramètre transporte ainsi des valeurs (par exemple, une position graphique donnée par l'utilisateur sera transportée par un jeton « position » stockant une abscisse et une ordonnée). Les jetons commandes représentent quant à eux l'intention de l'utilisateur. Ils transportent l'identifiant d'une tâche à activer.

Tâches

Dans H4, les tâches du système (par opposition aux tâches de l'utilisateur) sont représentées par des signatures de fonctions appelées questionnaires. Un questionnaire est défini par son nom, ses paramètres d'entrée et, éventuellement, un paramètre de sortie. L'existence ou non d'un paramètre de sortie définit la catégorie de la tâche : une tâche de production (au sens du dialogue, toujours) comportera un paramètre de sortie, alors qu'une tâche terminale n'en possèdera pas. Les tâches du système sont implémentées dans l'adaptateur de noyau fonctionnel et constituent le lien entre le contrôleur de dialogue et les primitives de l'application.

Interacteurs de dialogue

L'essentiel de l'originalité de H4 réside dans la définition d'un composant propre au dialogue, qui nous avons appelé « Interacteur de dialogue ». Nous avons vu précédemment que le dialogue structuré avait pour caractéristique principale de permettre de coller parfaitement à l'arbre de tâches de l'utilisateur. Le rôle des interacteurs de dialogue sera de permettre une implémentation directe de ce dialogue structuré, et donc de l'arbre de tâches de l'utilisateur.

Les interacteurs de dialogue de H4 sont concrètement chargés de gérer les appels des primitives du système, en fonction des entrées de l'utilisateur. En cela, ils diffèrent totalement des interacteurs d'entrée/sortie de (Harrison et Duce, 1994), (Paternò et Faconti, 1994) ou (Myers *et al.*, 1990), chargés eux des échanges entre le système et l'utilisateur, qui correspondent plus généralement à la notion de widget.

La première déclinaison du modèle H4 a été appelée « interacteurs hiérarchisés », en référence à l'organisation hiérarchique des interacteurs de dialogue. Lorsque l'on décompose les tâches effectuées par un utilisateur d'AGICT, on est en mesure de caractériser aisément des niveaux d'abstraction distincts, s'appuyant en particulier sur les catégories de tâches (création, calcul, sélection) mentionnées plus haut. Les tâches de création, qui correspondent au but final de l'utilisateur, sont du niveau le plus élevé. Les tâches de sélection, qui ne servent qu'à déterminer sur quels objets vont porter les actions, sont du niveau le plus bas. Les tâches de calcul, permettant de transformer des attributs pour qu'ils soient utilisés dans le cours de la tâche, appartiennent à un ou plusieurs niveaux intermédiaires. Ces observations ont conduit (Guittet et Pierra, 1993), (Girard *et al.*, 1995) et (Guittet, 1995) à définir un interacteur de dialogue comme devant représenter un niveau d'abstraction. Il réunit ainsi un certain nombre de questionnaires logiquement équivalents et indépendants, correspondant chacun à une tâche du

système. Les interacteurs sont ensuite organisés de façon hiérarchique, d'où le nom qui leur a été donné dans la première déclinaison de H⁴.

La gestion des appels des primitives systèmes (à travers les questionnaires) à partir des entrées de l'utilisateur (représentées par les jetons) nécessite la description d'une certaine dynamique. Les différentes implémentations de H⁴ réalisées jusqu'à présent s'appuient pour cela sur des réseaux de transitions augmenté (ATN pour Augmented Transition Network (Woods, 1970)). Ce choix d'implémentation n'est nullement imposé par le modèle H⁴. Cependant, afin d'illustrer de façon plus concrète la suite de notre discours, nous nous fonderons sur cette modélisation de la dynamique du dialogue.

Un ATN peut-être vu comme un automate à états fini comprenant une variable d'état, appelée registre. Une particularité de ces ATN réside dans le fait qu'ils reviennent toujours dans l'état initial, qui correspond à l'attente d'un début de commande. Dans cet automate, les transitions s'effectuent sur des jetons : quand l'interacteur de dialogue reçoit un jeton d'un type donné, il « regarde » si son état courant propose une transition vers un autre état par ce type de jeton. Si c'est le cas, il franchit la transition et stocke la valeur du jeton accepté dans son registre, qui peut ainsi contenir plusieurs valeurs. Lorsqu'il reçoit un jeton qui le renvoie dans son état initial, il appelle le questionnaire correspondant, lui transmet toutes les valeurs stockées et vide son registre (Figure 2).

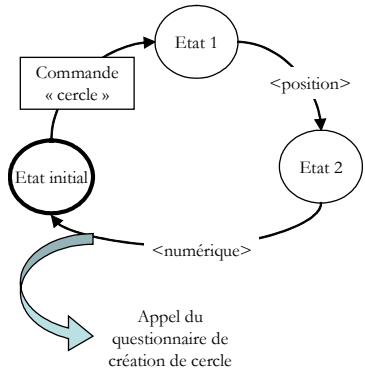


Figure 2. *Automate représentant un questionnaire de création de cercle par centre et rayon.*

Dans le cadre d'une AGICT utilisant un dialogue préfixé, l'état initial de l'interacteur de dialogue ne comporte que des transitions franchissables par un jeton de type commande. Cela correspond à contraindre l'utilisateur à fournir d'abord le nom de l'opération à effectuer avant d'en donner les opérandes.

Moniteur

L'organisation hiérarchique des interacteurs de dialogue (donc des questionnaires et plus généralement des tâches) est à la charge d'un composant logiciel appelé moniteur. Les interacteurs de dialogue sont classés de bas en haut en suivant leur niveau d'abstraction (les interacteurs de plus bas niveau étant placés sous ceux de niveau plus élevé). En dehors de cette hiérarchisation, le rôle du moniteur est de récupérer les jetons de la couche de présentation et de les transmettre successivement à tous les interacteurs de dialogue, en commençant par celui de plus bas niveau, jusqu'à celui de plus haut niveau, selon un algorithme décrit dans le paragraphe ci-après.

Fonctionnement

Lorsque l'utilisateur effectue une interaction, l'adaptateur de présentation transforme la donnée reçue de la couche présentation en un jeton d'un type correspondant contenant une valeur. Par exemple, un clic de souris peut être transformé en un jeton de type « position » contenant les coordonnées du point de l'écran sur lequel l'utilisateur a cliqué. Le jeton ainsi créé est transmis au moniteur qui le présente au premier interacteur de sa hiérarchie.

Si l'état courant de l'automate qui représente cet interacteur possède une transition sur ce type de jeton, alors il « accepte » ce jeton (on dit aussi qu'il le consomme). Dans ce cas, l'automate change d'état et stocke la valeur du jeton. D'autre part, si la transition replace l'automate dans son état initial, l'interacteur appelle le questionnaire correspondant en lui transmettant toutes les valeurs enregistrées. Le questionnaire est ensuite chargé de faire le lien avec le noyau fonctionnel en transformant les jetons en données du domaine et en appelant la primitive associée. Dans le cas où cette primitive produit un résultat (cas d'une tâche de production), la valeur calculée est à son tour transformée en jeton puis rendue au moniteur qui se charge de la transmettre aux interacteurs de plus haut niveau dans la hiérarchie (Figure 3).

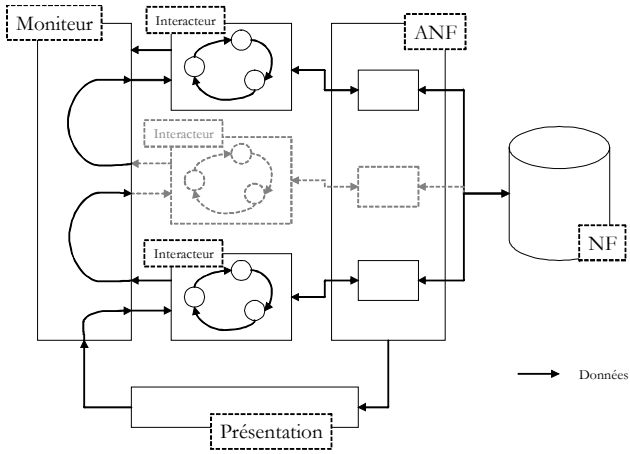


Figure 3. *Fonctionnement du contrôleur de dialogue de H[4].*

Si l'automate de l'interacteur ne se trouve pas dans un état propre à activer une transition par le jeton proposé, il le rend au moniteur qui le propage à l'interacteur de plus haut niveau dans la hiérarchie, et ainsi de suite jusqu'à ce qu'il soit consommé ou qu'il n'y ait plus d'interacteur.

Cette organisation et ce fonctionnement assurent l'indépendance des tâches entre elles. Aucun interacteur ne connaît ni la source, ni la destination des jetons qu'il reçoit et qu'il émet. Ceci assure une parfaite modularité et permet à tout interacteur d'intervenir dans la construction d'une phrase du dialogue, sans que cela ait été prévu initialement par le concepteur de l'application. Cette caractéristique est très intéressante dans la mesure où elle permet de faire abstraction de l'utilisation ultérieure des tâches que l'on définit. Ainsi, une tâche calculant l'intersection de deux segments pourra intervenir dans une tâche de création de cercle par son centre et son rayon, sans que cela n'ait été préalablement établi.

Cependant, cette indépendance syntaxique ne dispense pas d'une bonne structuration logique. Il faut être vigilant dans le placement des interacteurs à

l'intérieur de la hiérarchie. Si I_1 et I_2 sont deux interacteurs, rangés dans cet ordre, acceptant tout deux une entrée de type T, un jeton de type T accepté par I_1 ne sera jamais proposé à I_2.

Considérons par exemple une application mettant en scène une calculette grapho-numérique. L'utilisateur désire calculer le résultat de la multiplication d'un nombre n par la distance séparant deux pointés graphiques. Pour réaliser le second pointé graphique, il doit faire appel à une fonction de Zoom. L'ambiguïté va porter sur les paramètres d'entrée des interacteurs chargés du Zoom (deux coordonnées spatiales) et du calcul d'une distance (également deux coordonnées spatiales). Si on exécute la série d'instructions suivante :

<Calcul> n * <Distance> ↗ <Zoom> ↗ ↗ ↗

<>	*représente une commande*
n	*représente un entier*
*	*représente l'opérateur de multiplication*
↗	*représente un pointé graphique*

La question est de savoir comment la séquence va être interprétée. Le système doit-t-il considérer les deux pointés graphiques suivant la commande <Zoom> comme les deux coordonnées spatiales nécessaires à l'activation du zoom, ou bien doit-il utiliser le deuxième pointé pour activer la fonction de calcul de distance à laquelle on a déjà fourni un paramètre ? Ceci est entièrement déterminé par la hiérarchie dans le moniteur : si l'interacteur permettant le calcul de la distance est placé avant celui chargé du zoom, le calcul du produit aura lieu avant le zoom. Cette particularité entraîne la nécessité de disposer d'une méthodologie stricte de positionnement des interacteurs les uns par rapport aux autres. Nous verrons dans la partie outils qu'il est possible de construire un outil de vérification de la logique de la hiérarchisation des interacteurs de dialogue.

Conclusion sur le modèle H⁴

H⁴ ne prétend pas posséder un caractère universel permettant la modélisation de toute forme de dialogue. En contrepartie, sa spécialisation lui donne la possibilité de décrire de manière précise un contrôleur de dialogue autorisant une modélisation simple d'un dialogue structuré. Cette particularité en fait un modèle d'architecture logicielle privilégié pour les AGICT. Comme nous l'avons dit précédemment, les applications et les outils développés jusqu'à aujourd'hui utilisent, pour la modélisation de la dynamique des interacteurs de dialogue, des automates. Ceci n'est pas un pré-requis dans H⁴, qui n'impose pas ce point ; ainsi, l'utilisation de réseaux de Petri, par exemple, serait tout aussi pertinente à ce niveau.

2.3 Exemple

Cette partie illustre le fonctionnement du contrôleur de dialogue de H⁴ sur un exemple détaillé.

Étude de cas

Considérons la tâche permettant de « créer un segment par ses deux extrémités dont l'une est le centre d'un cercle, et l'autre l'intersection de deux segments ». La figure 4 illustre l'analyse de cette tâche comme pourrait le faire un ergonome. Les rectangles symbolisent les commandes de l'utilisateur tandis que les textes en italique sont des désignations directes d'objets graphiques. L'opérateur temporel AND précise que les deux sous-tâches peuvent être exécutées dans n'importe quel ordre.

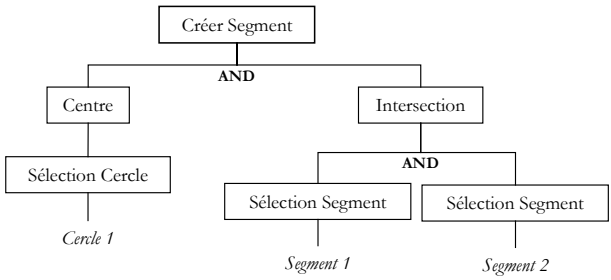
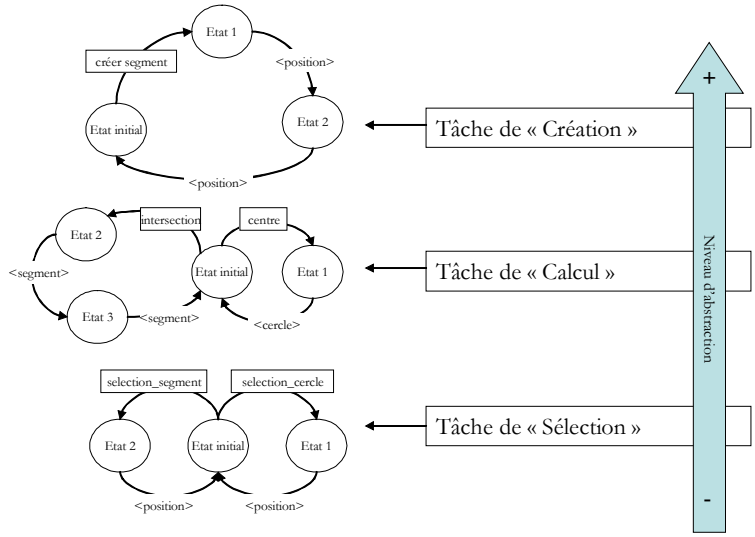


Figure 4. *Tâche de création d'un segment.*

Structure des questionnaires et interacteurs de dialogue

Dans H4, chaque sous-tâche est représentée par un questionnaire. Ces questionnaires sont ensuite regroupés selon leur niveau d'abstraction à l'intérieur d'interacteurs, comme l'illustre la figure 5.


Etat 1
créer segment
<position>
Etat initial
Etat 2
<position>
Tâche de « Création »
intersection
centre
Etat 2

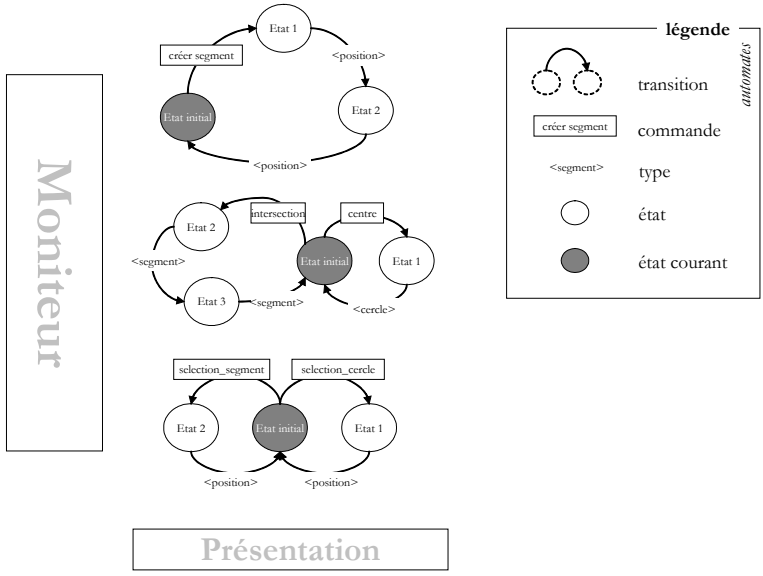
Etat initial
Etat 1
Etat 3

<cercle>
Tâche de « Calcul »
selection_segment
selection_cercle
Etat 2
Etat initial
Etat 1
<position>
<position>
Tâche de « Sélection »
+
Niveau d'abstraction
-

Figure 5. *Trois automates représentant trois interacteurs de dialogue de niveau* d'abstraction *différent et regroupant les tâches « créer segment », « récupérer le centre d'un cercle », « calculer l'intersection de deux segments », « sélectionner un segment » et « sélectionner un cercle ».*

Dynamique du dialogue

Dans la description qui suit, on suppose que le dialogue se trouve dans son état initial, l'utilisateur n'ayant pas encore interagi avec l'application, c'est-à-dire que tous les automates (représentant chacun un interacteur du contrôleur de dialogue) sont dans leur état initial (Figure 6).


Moniteur
Etat 1
créer segment
<position>
Etat initial
Etat 2
<position>
Etat 2
intersection
centre

Etat initial
Etat 1
Etat 3

<cercle>
selection_segment
selection_cercle
Etat 2
Etat initial
Etat 1
<position>
<position>
Présentation
légende
automates
transition
créer segment
commande

type
état
état courant

Figure 6. *Le contrôleur de dialogue dans son état initial.*

Pour initier la phase de construction du segment, l'utilisateur doit informer le système en sélectionnant la commande correspondante (par ex. à partir d'une icône ou d'une entrée de menu de l'IHM). À partir de cette information de bas niveau (clic sur un bouton), l'adaptateur de présentation crée un jeton de type commande, portant l'information « création de segment », qu'il transmet au moniteur. Le moniteur transmet ce jeton au premier interacteur de sa hiérarchie (étape 1, Figure 7).

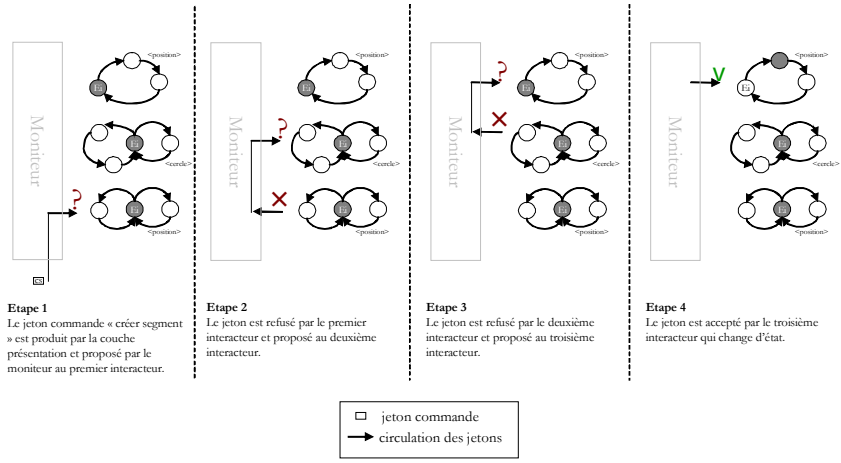
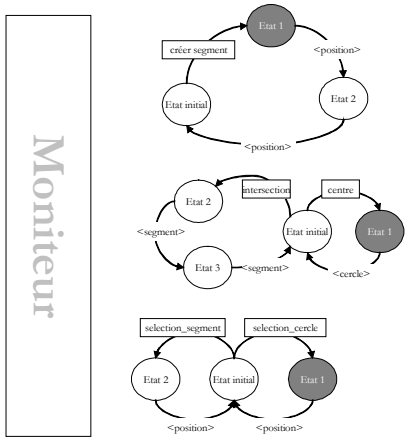


Figure 7. *Circulation d'un jeton commande « créer segment » dans la hiérarchie des interacteurs.*

Celui-ci ne possède aucune transition sur un jeton de type « commande » portant la valeur « créer segment ». Il rend donc le jeton au moniteur qui le transmet à l'interacteur de plus haut niveau (étape 2, Figure 7). Pour les mêmes raisons, le

jeton est également refusé par celui-ci puis finalement proposé au dernier interacteur (étape 3, Figure 7) qui se trouve dans son état initial et possède une transition sur un jeton commande de valeur « créer segment ». L'interacteur accepte donc le jeton : il stocke sa valeur dans son registre et change l'état de son automate (étape 4, Figure 7).

L'utilisateur continue à interagir avec le système. Il doit maintenant fournir au système le premier opérande de l'opération de création de segment par deux extrémités. Pour cela, il doit ou bien fournir une position, ou bien fournir une commande dont le résultat produira une position. Il choisit cette dernière alternative et sélectionne successivement les commandes « obtenir le centre d'un cercle » et « sélectionner un cercle ». De la même manière que précédemment, un jeton commande est créé à chaque fois, et proposé successivement aux interacteurs. Le premier jeton est cette fois consommé par le deuxième interacteur dans l'ordre de la hiérarchie, et le second jeton par le premier interacteur. À l'issue de ces manipulations, le contrôleur de dialogue se trouve donc dans l'état décrit à la figure 8).


Moniteur
Etat 1
créer segment
<position>
Etat initial
Etat 2
<position>
Etat 2
intersection
centre

Etat initial
Etat 1
Etat 3

<cercle>
selection_segment
selection_cercle
Etat 2
Etat initial
Etat 1
<position>
<position>

Figure 8. *Etat du contrôleur de dialogue après l'activation successive des commandes « créer segment », « centre d'un cercle », « sélectionner un cercle ».*

L'utilisateur doit ensuite fournir un pointé graphique correspondant à la sélection du cercle désiré. Lorsqu'il clique sur l'écran avec la souris, l'adaptateur de présentation crée un jeton de type « position », portant comme valeur la position du curseur de la souris. Celui-ci est transmis au moniteur, qui le présente au premier interacteur de sa hiérarchie, lequel est dans un état propre à l'accepter puisqu'il a préalablement reçu le jeton commande « selection_cercle » (étape 1 de la Figure 9). L'interacteur, qui revient de ce fait dans son état initial, active du même coup le questionnaire correspondant. Celui-ci transforme la position du curseur transportée par le jeton en une donnée compréhensible par le noyau fonctionnel, et appelle la primitive de recherche d'un cercle à partir d'un point donné (Figure 10). La valeur calculée est à son tour transformée en jeton (de type « cercle ») et rendue au moniteur (étape 2 de la Figure 9).

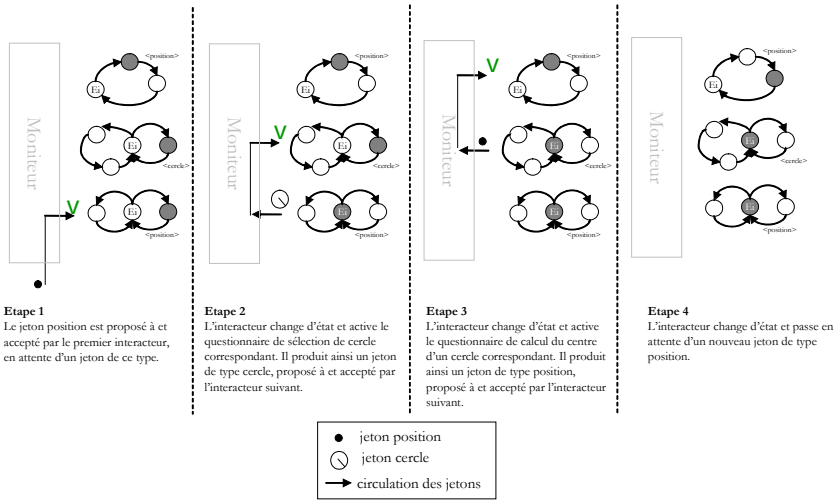


Figure 9. *Evolution de l'état des interacteurs lors d'un clic souris, lorsque les tâches de création de segment, de sélection d'un cercle et de récupération du centre d'un cercle sont en cours.*

Le jeton de type cercle ainsi généré est transmis à l'interacteur suivant dans la hiérarchie. Or celui-ci est justement en attente d'un jeton de ce type. Il l'accepte et, selon le même procédé, active le questionnaire correspondant à la récupération du centre d'un cercle (remarque : comme ce questionnaire utilise le jeton qu'il vient de recevoir comme paramètre, c'est bien le centre du cercle sélectionné par l'utilisateur que l'on va calculer). En outre, ce questionnaire crée un nouveau jeton de type « position » et le rend au moniteur, lequel le propose à l'interacteur de niveau immédiatement supérieur, qui attend un jeton de ce type (étape 3 de la Figure 9). Une nouvelle fois l'interacteur change d'état (étape 4 de la Figure 9), sans toutefois qu'un questionnaire soit appelé puisque le franchissement de cette transition ne l'a pas ramené dans son état initial.

Une fois cette séquence terminée (Figure 9), l'utilisateur a fourni à la tâche de création de segment une de ses deux extrémités. Pour que le questionnaire correspondant soit activé il faut lui fournir le second paramètre nécessaire à son exécution. Un processus identique aux exemples précédents permet à l'utilisateur de fournir cette position par l'intermédiaire d'un questionnaire produisant un jeton de ce type, en calculant l'intersection de deux segments comme prévu par le modèle des tâches (Figure 4). Notons que l'utilisateur peut également utiliser un pointé graphique pour fournir cette position. Cette alternative est disponible bien qu'elle n'ait pas été explicitement prévue initialement du fait de l'indépendance des interacteurs. Au final, un segment est créé dans le noyau fonctionnel du système par l'intermédiaire du questionnaire adéquat (Figure 10).

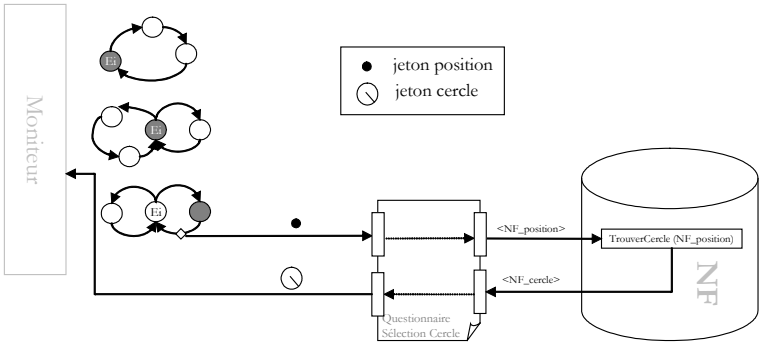


Figure 10. *Lors de la réception d'un jeton « position », l'automate représentant la tâche « sélectionner un cercle » revient dans son état initial en appelant le questionnaire correspondant. Celui-ci transforme la donnée reçue (le jeton position) en donnée du noyau fonctionnel (NF_position) et appelle la primitive de recherche de cercle. Le résultat produit est à son tour transformé en jeton et rendu au moniteur.*

2.4 H⁴ et les autres modèles hybrides

Comme nous l'avons dit en introduction, le modèle H⁴ doit être considéré comme un modèle hybride, au même titre que PAC-Amodeus. Comme ce dernier, il est fondé sur le modèle Arch, dont il reprend la structure globale. La seule entorse à ce modèle réside dans la possibilité de court-circuiter la transmission d'information en retour du noyau fonctionnel par le contrôleur de dialogue. En effet, dans le modèle Arch, les composants ne sont censés communiquer qu'avec leurs voisins immédiats ; ainsi, l'Adaptateur de Noyau Fonctionnel ne peut échanger qu'avec le Noyau Fonctionnel et le Contrôleur de Dialogue. Dans H⁴, l'ANF peut effectuer une projection directe dans la présentation, lorsqu'il s'agit uniquement de visualisation. Cette liberté prise avec Arch se justifie dans le cadre des AGICT, mais ne constitue pas un élément essentiel du modèle. On notera que cette particularité réhabilite un point particulier du modèle de Seeheim (la petite « boîte » au rôle indéterminé dans l'article original), en lui donnant une sémantique : la possibilité de dissocier la présentation d'informations de la hiérarchie des tâches, point qui se justifie dans certains contextes (ce n'est pas le cas général).

En première approche, on peut considérer que le modèle PAC-Amodeus peut être inclus dans la définition de H⁴, comme une spécialisation de ce dernier. En effet, H⁴ préconise de modéliser une hiérarchie (optionnelle) dans quatre composants de l'architecture Arch. Or, PAC-Amodeus préconise de représenter une telle hiérarchie (d'objets PAC) dans le contrôleur de dialogue. Les autres composants n'étant pas particulièrement détaillés par rapport au modèle Arch original, il n'est pas exclu de modéliser de façon hiérarchique le noyau fonctionnel, et l'utilisation des boîtes à outils d'interaction impose généralement une hiérarchie propre au composant d'interaction. Notons d'ailleurs que la décomposition hiérarchique du contrôleur de dialogue poursuit dans les deux cas le même objectif : représenter au mieux le modèle de tâches du système, du point de vue de l'utilisateur. La différence de modélisation dans le cas de l'implémentation des interacteurs hiérarchisés se justifie par le domaine particulier des AGICT, comme nous l'avons exposé dans la section sur le contexte.

Tout comme H⁴, PAC-Amodeus a été implémenté dans un contexte particulier, celui de la multimodalité. Ceci a conduit à proposer une architecture implémentationnelle adaptée. À l'inverse de H⁴, aucun outil n'a été proposé pour

faciliter, voire automatiser, l'implémentation d'applications. En revanche, des travaux importants ont été menés pour concevoir des heuristiques et des motifs architecturaux pour la conception de la hiérarchie des agents PAC (i.e. PAC-Expert (Nigay, 1994)), qui pourraient assurément être adaptés dans d'autres contextes.

Pour terminer sur les architectures, il est intéressant de considérer une dernière architecture, l'architecture multi-couche (Fekete, 1996). Elle s'adresse à une classe d'applications particulière, celle des « éditeurs ». Cette classe présente beaucoup de caractéristiques de la classe des AGICT, en particulier celles concernant les propriétés des tâches. Le modèle multi-couche est décomposé en un nombre variable de couches, gérant chacune une partie de l'interaction. À ce titre, il peut être considéré comme un modèle hybride. Cependant, à l'inverse de PAC-Amodeus et H⁴, il n'est pas constitué d'un nombre fixe de composants à partir de Arch.

On peut considérer que, pour une part de l'architecture, celle du contrôleur de dialogue, le modèle multi-couche est proche de l'implémentation des interacteurs hiérarchisés, avec laquelle il partage certains objectifs. La principale différence réside dans le fait que les différentes couches se connaissent entre elles, alors que les interacteurs de dialogue demeurent totalement indépendants les uns des autres.

3 Usages et pratiques de H⁴ : retours d'expériences

Depuis les premiers résultats sur H⁴, il y a maintenant plus de 10 ans, de nombreux travaux sont venus alimenter l'ensemble des outils de mise en œuvre de l'architecture et d'exploitation de ses propriétés. Cette partie propose la description de quelques-uns des plus remarquables d'entre eux : une boîte à outils et un éditeur graphique de programmation du contrôleur de dialogue, ainsi qu'une méthode de vérification de propriétés du dialogue d'une application reposant sur H⁴.

3.1 La Boîte à Outils du dialogue

L'idée principale de la boîte à outils du dialogue repose sur l'analogie que l'on peut faire entre la conception d'une interface graphique et celle du dialogue d'une application : dans les deux cas il s'agit de réifier un certain nombre de composants bien identifiés. La boîte à outils du dialogue fournit donc un ensemble de classes représentant des éléments de contrôle du dialogue décrits dans le modèle d'architecture H⁴. Elle permet de construire le contrôleur de dialogue d'une application interactive de la même façon qu'un concepteur crée la couche de présentation avec une boîte à outils de programmation d'interfaces graphiques.

Dans cette partie, nous analysons les problèmes que peut poser la mise en œuvre pratique de H⁴, et nous présentons une boîte à outils permettant de simplifier le travail du concepteur.

Principe de base

Tout comme pour PAC-Amodeus (d'où les travaux de L. Nigay sur les heuristiques et les motifs de conception), une des difficultés de la mise en œuvre des principes décrits dans H⁴ est la définition de la structuration, et donc ici des interacteurs. En effet, la structure sous-jacente d'un interacteur est un réseau de transitions augmenté. Or, la création de tels automates, si elle est assez simple, reste longue et fastidieuse pour le concepteur. Pour un système de CAO standard, possédant une cinquantaine d'actions constructives (donc du même niveau d'abstraction) avec en moyenne deux paramètres chacune, l'automate résultant posséderait une centaine d'état et quelques milliers de transitions.

Nous sommes partis du principe que les interacteurs avaient la charge des appels des questionnaires. Un appel est réalisé lorsque l'interacteur a reçu les

données nécessaires, c'est-à-dire lorsqu'il a reçu le nom de la commande correspondant au questionnaire (sous la forme d'un jeton commande) ainsi que la suite des paramètres utilisés par le questionnaire (sous la forme de jetons paramètres). L'automate est utilisé afin de contrôler que les jetons reçus sont bien ceux qui manquaient pour l'appel d'un questionnaire. On s'aperçoit donc que si l'on spécifie un questionnaire en termes de commande d'activation et de nature de jetons paramètres, la suite de transitions permettant son appel est strictement déterminée par la suite de ses paramètres.

Si on considère un automate responsable de l'appel d'un questionnaire de création d'un cercle par son centre et son rayon, la valeur du centre est contenue dans un jeton de type position et le rayon dans un jeton de type nombre. La première transition se fait sur un jeton commande représentant la commande cercle permettant à l'utilisateur de spécifier au système qu'il veut créer un cercle. L'enchaînement des transitions amenant à la création du cercle est : <Commande cercle>, <position>, <nombre>. Cela correspond exactement à la spécification du questionnaire. Il est donc possible de générer automatiquement l'automate à partir de cette spécification : la construction du contrôleur de dialogue est réalisée à partir d'objets représentant la signature des fonctions qui constituent les tâches du système.

Composition de la boîte à outils du dialogue

De la même façon que pour les boîtes à outils de programmation d'interfaces, la boîte à outils du dialogue est basée sur un ensemble de classes, réification des éléments du contrôleur de dialogue de H4.

Jetons. La boîte à outils du dialogue offre un certain nombre de classes de jetons, et permet au concepteur d'en définir de nouvelles. On distingue deux grandes classes : la classe *Commande* qui représente l'intention de l'utilisateur, et la classe *Paramètre*, qui est une classe abstraite et permet au concepteur de créer une hiérarchie de classes de paramètres. Pour créer une sous-classe de paramètres, le concepteur doit fournir le type de données référencées et la classe dont elle hérite. La hiérarchie des jetons paramètres permet de factoriser l'écriture de la spécification des questionnaires, en définissant par exemple un seul questionnaire basé sur des jetons de classe *Objet*, plutôt que deux questionnaires basés sur des jetons *Droite* et *Cercle*, héritant de la classe *Objet*.

Questionnaires. La classe *Questionnaire* permet de décrire la spécification des tâches de l'application. Les automates contenus dans les diagets (éléments de dialogue implémentant les interacteurs de dialogue) sont générés à partir de ces spécifications. Le formalisme permet de décrire aussi bien des tâches simples que des tâches possédant une itération sur un des paramètres d'entrée.

Diagets. La classe *Diaget* (pour « dialog + gadgets », par similitude avec les widgets « windows + gadgets » des boîtes à outils de présentation) réifie les interacteurs de dialogue de H4. Ils permettent l'organisation hiérarchique des tâches du système décrites sous la forme de questionnaires, et sont chargés de conserver et d'organiser les jetons qu'ils reçoivent afin de les transmettre aux questionnaires. Ils possèdent à ce titre la liste des questionnaires qu'ils permettent d'appeler.

La structure sous-jacente des diagets est un réseau de transitions augmenté. Chaque diaget possède un nombre fini d'états reliés entre eux par des transitions. Une transition est définie par un état de départ, un état d'arrivée, et est étiquetée par la nature du jeton qui la valide.

D'autre part, comme le rôle des diagets est de contrôler les appels des questionnaires, chacun conserve les jetons qu'il consomme dans son registre jusqu'à ce qu'un questionnaire puisse être appelé (lorsque tous les paramètres du

questionnaire ont été reçus par le diaget). Alors, le diaget réalise cet appel, en fournissant au questionnaire les jetons conservés dans son registre.

Les Diagets constituent une implémentation objet de la notion d'interacteur de dialogue telle qu'elle a été décrite dans la section 2. Ils se distinguent des interacteurs hiérarchisés principalement dans leur dynamicité : en tant que réels objets, ils peuvent être programmés complètement dynamiquement ; ils constituent ainsi véritablement une brique élémentaire de dialogue.

Moniteur. La classe *Moniteur* est chargée de l'organisation hiérarchique des diagets comme cela est décrit dans le modèle H4, et possède donc à ce titre une liste de diagets classés hiérarchiquement. Le moniteur récupère les jetons venant de la couche d'entrée-sortie et les transmet de diaget en diaget jusqu'au dernier, ou jusqu'à ce qu'il n'y ait plus de production. Il est important de noter qu'il n'y a qu'un seul moniteur par application.

Génération de l'automate

L'une des difficultés de la mise en œuvre du modèle d'architecture H4 est la définition manuelle des automates contenus dans les interacteurs de dialogue. La boîte à outils du dialogue propose de réduire le temps et les erreurs de conception en générant automatiquement les automates à partir des spécifications des questionnaires qu'ils permettent d'appeler.

La boîte à outils du dialogue utilise le jeton commande défini à partir du nom du questionnaire pour créer une transition à partir de l'état initial, puis génère les transitions et les états permettant de récupérer l'ensemble des paramètres nécessaires à l'appel des questionnaires. Enfin, il ajoute aux transitions sur le dernier paramètre l'appel du questionnaire. L'état d'arrivée du questionnaire est remplacé par l'état initial. Cela signifie qu'à chaque fois qu'un diaget appelle un questionnaire, il revient dans l'état initial.

La boîte à outils utilise un algorithme par construction (algorithme incrémental à partir des différents questionnaires) pour générer les automates. Cet algorithme est moins coûteux en temps de calcul qu'un algorithme d'optimisation (Autebert, 1994). Notons qu'il permet également de définir des diagets récursifs, correspondant à des tâches récursives (comme la construction d'une expression parenthésée). Pour plus de détails sur ce sujet, on se référera à la thèse de G. Texier (Texier, 2000).

Contrôle de l'application

La boîte à outils du dialogue offre la possibilité au concepteur de créer le contrôleur de dialogue de l'application à partir de la simple description des tâches réalisées par le système. Le rôle principal du contrôleur de dialogue est de réaliser l'appel des actions du noyau fonctionnel en fonction des entrées de l'utilisateur. En fait, la boîte à outils du dialogue offre un certain nombre de services complémentaires au concepteur. Elle propose, entre autres, des mécanismes de prévention et de correction d'erreur au sens de (Lewis et Norman, 1986) et (Van Der Schaaf, 1997).

Prévention des erreurs. Dans le cadre des applications pilotées par les données, où le système change d'état en fonction des types de données fournies par l'utilisateur, il est nécessaire que ce dernier puisse fournir tous les types de données attendus par le système à un certain moment, et qu'il lui soit impossible de fournir des types de données non attendus. Pour cela, il faut que le système soit en mesure de contrôler la couche de présentation, afin d'activer les widgets permettant de fournir des données attendues, et de désactiver les widgets qui fournissent des données non désirées. Par exemple, si le système attend uniquement des positions (lorsque l'utilisateur crée une droite par deux points), l'utilisateur ne doit pas être en

mesure de lui fournir de valeurs numériques. Cependant, toutes les commandes susceptibles d'appeler une action qui produit des points doivent tout de même être accessibles à l'utilisateur.

La boîte à outils du dialogue offre un mécanisme permettant de réaliser le contrôle des entrées. Le moniteur possède une méthode d'analyse qui permet à chaque instant de connaître, d'une part, l'ensemble des types de jetons paramètres attendus par le contrôleur de dialogue pour réaliser une action, et d'autre part, l'ensemble des commandes permettant de produire l'un des jetons attendus. Cela permet au moniteur d'autoriser les jetons (commandes ou paramètres) susceptibles d'être utilisés par le système.

Correction des erreurs. En général, on considère qu'il existe deux grandes méthodes pour la correction d'erreurs (Dix *et al.*, 1993) : la correction en arrière, dans le cas où le système revient dans l'état où il se trouvait précédemment, et la correction en avant, dans le cas où l'utilisateur réalise une tâche spécifique pour replacer le système dans l'état désiré. Ce second cas relève plus de la planification hiérarchique des tâches que d'un problème propre au dialogue. En revanche, la correction arrière présente deux fonctions directement liées au dialogue. Ces fonctions se distinguent par leur niveau de granularité : la fonction annuler (une commande en cours) et la fonction défaire (la dernière interaction).

Pour l'annulation, la boîte à outils dispose d'un jeton commande spécifique « annuler » capable de replacer toutes les instances de diagets dans leur état initial. Cependant, ce simple mécanisme ne suffit pas pour effacer toutes les conséquences déjà acquises de la tâche initiée. Lorsque l'utilisateur annule une phrase en cours, il est probable que le contrôleur de dialogue ait appelé un ou plusieurs questionnaires et/ou réalisé un ou plusieurs échos. Le système doit alors annuler toutes les actions réalisées lors de ces appels et/ou effacer les échos. Pour que cela soit possible, les diagets peuvent exécuter un questionnaire particulier qui possède en entrée le nom du questionnaire dont on veut annuler l'exécution. Ce questionnaire est chargé d'appeler une méthode d'annulation spécifique du noyau fonctionnel. Cette méthode fait partie des trois services indispensables que doit fournir le noyau fonctionnel d'une application graphique interactive (Fekete, 1996). D'autre part, il est important de noter que l'ordre d'annulation des questionnaires est important et doit correspondre à l'ordre inverse de leur appel.

La fonction « défaire » permet également de gérer le retour en arrière du noyau fonctionnel et de l'interaction, mais avec une granularité limitée à une seule interaction. Dans ce cas, l'utilisateur n'annule qu'une partie de l'expression en cours. D'un point de vue technique, le « défaire » est donc réalisé par le mécanisme suivant : le moniteur conserve tous les jetons qui lui arrivent dans une liste. Lorsqu'il reçoit la commande « défaire » (correspondant à un jeton commande particulier), il envoie au diaget la commande d'annulation (de manière identique à l'annulation). Tous les diagets reviennent alors dans leur état initial, et toutes les actions, réalisées au cours de l'expression du but en cours, sont annulées. Ensuite, le moniteur retransmet dans l'ordre et un à un tous les jetons mémorisés sauf le dernier. Ainsi, l'application retrouve l'état qui était le sien juste avant d'avoir reçu le dernier jeton. Bien entendu, si l'utilisateur effectue un ensemble de fonctions « défaire », cela revient à effectuer une fonction « annuler ». La fonction « refaire » n'a pas été implantée.

Conclusion sur la boîte à outils du dialogue

Dans cette section, nous avons décrit la boîte à outils du dialogue. Elle est composée de plusieurs classes qui sont des réifications des éléments décrits dans l'architecture H4. Des choix ont été faits dans la définition des différentes classes,

afin de permettre l'automatisation de certaines tâches, comme la création automatique de l'automate interne à partir des signatures des questionnaires. De même, la structure et le comportement du moniteur sont essentiellement adaptés au domaine des AGICT : cette boîte à outils permet de modéliser aisément tous les types de tâches rencontrés dans les systèmes supportant des tâches structurées et multi-objets.

3.2 DTS Edit

Même en utilisant la boîte à outils du dialogue, la création du contrôleur de dialogue d'une application modélisée selon l'architecture H⁴ est un travail lourd et fastidieux de programmation (à l'instar de la programmation d'une interface graphique avec une boîte à outils). Le caractère répétitif des instructions mène souvent le concepteur à des erreurs dues par exemple aux nombreux copier/coller utilisés. Pourtant, l'organisation des interacteurs de dialogue entre eux propose des propriétés permettant théoriquement de vérifier la validité et la complétude des tâches modélisées. Malheureusement, un tel travail n'est pas envisageable « manuellement ».

Pour ces raisons, nous avons mis au point un outil interactif qui répond aux besoins du concepteur : il génère le code du contrôleur de dialogue en permettant une définition aisée des différents composants. D'autre part, il est capable de vérifier différentes propriétés de complétude des tâches en s'appuyant sur les propriétés et l'organisation des interacteurs de dialogue.

Présentation générale

La fenêtre principale de DTS Edit (Depaulis *et al.*, 2002) pour **D**ialog **T**ool**S**et **Edit**or se décompose en deux parties (Figure 11) :

- La partie gauche affiche toutes les informations sur les jetons. Elle permet notamment de séparer les jetons « commandes », des jetons paramètres. Ceux-ci sont présentés sous forme d'arbre, correspondant à la hiérarchie choisie par le concepteur.
- La partie droite est quant à elle chargée de représenter les interacteurs de dialogue et la signature des questionnaires qui les composent. L'ordre de présentation (haut vers bas) a une importance sémantique puisqu'elle correspond à la hiérarchie du moniteur.

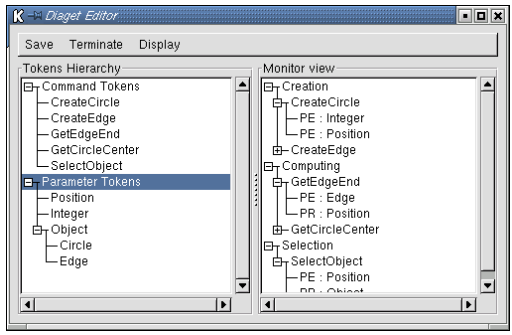


Figure 11. *Fenêtre principale de DTS Edit.*

Cet éditeur utilise largement la manipulation directe et autorise notamment des opérations telles que le « drag'n drop » d'un panel à l'autre. L'utilisateur a également

la possibilité de sauvegarder et de charger une architecture, que ce soit dans le format de l'éditeur ou dans celui du code généré[5].

Voyons maintenant comment un concepteur peut, à partir de cet éditeur, réaliser toutes les opérations (création de jetons, d'interacteurs, de questionnaires, etc.) permettant la description du contrôleur de dialogue de son application et effectuer les différentes vérifications présentées précédemment.

Jetons

La partie gauche de l'éditeur présente la hiérarchie des jetons de l'application. Cette hiérarchie est séparée en deux classifications distinctes.

D'un côté, on trouve les jetons « commande ». Ils transportent le nom des questionnaires et sont, dans l'implémentation initiale du Contrôleur de Dialogue de H[4], les seuls jetons capables d'initier l'automate d'un interacteur : à partir du moment où l'interacteur reçoit une commande correspondant à un de ses questionnaires, celui-ci est alors en mesure de recevoir les autres jetons de sa séquence d'entrée. Dans la plupart des applications, ils sont fournis par l'utilisateur de DTS-Edit au Contrôleur de Dialogue par l'intermédiaire des menus ou des boutons de l'interface graphique.

Dans une nouvelle hiérarchie, l'interface de l'éditeur présente les jetons « paramètre ». Ils s'agit de jetons transportant les valeurs des objets du noyau fonctionnel (par exemple des cercles et des segments dans une application de type MacDraw™) et de la couche de présentation (par exemple des pointés graphiques). Ces jetons sont organisés de manière hiérarchique, exactement comme le sont les objets du Noyau Fonctionnel de l'application. Par exemple, dans une application de type MacDraw™, l'utilisateur peut créer des cercles et des segments dont on peut imaginer qu'ils héritent chacun d'un objet de plus haut niveau d'abstraction, « Objet Graphique ». L'organisation des objets du Noyau Fonctionnel (la hiérarchie de classes dans un langage à objets) doit être transmise aux jetons afin que le dialogue puisse profiter des mêmes avantages que le modèle. Ainsi, des questionnaires peuvent être regroupés entre eux. Dans l'exemple de l'application de type MacDraw™, il n'est pas nécessaire de définir plusieurs questionnaires de sélection : un seul peut regrouper la sélection de tous les objets graphiques, qu'ils soient des cercles ou des segments. Grâce à la relation d'héritage, le questionnaire peut se contenter de porter sur un jeton correspondant aux « Objets Graphiques » et être utilisé aussi bien pour les cercles que pour les segments.

Dans DTS Edit, les deux classifications de jetons sont affichées à l'écran. Les commandes sont rangées dans un arbre à un seul niveau de profondeur (il n'y a pas d'héritage entre les jetons « commande »). Ces commandes sont créées de manière automatique. En effet, chaque commande correspond à l'activation d'un questionnaire particulier. De ce fait, à chaque nouvelle création de l'un d'eux, il suffit de créer le jeton « commande » associé, en lui donnant exactement le nom du questionnaire.

Les jetons « paramètre » sont pour leur part affichés sous la forme d'un arbre représentant les relations d'héritage. Chaque élément d'un sous-arbre hérite des propriétés de l'élément racine dont le sous-arbre est issu. Contrairement aux jetons « commande », les paramètres ne sont pas créés automatiquement. Pour définir un nouveau jeton de ce type, l'utilisateur doit définir son nom et spécifier s'il existe une relation d'héritage avec un type de jeton existant. Le nouveau jeton est alors ajouté dans la hiérarchie (Figure 12). Il est important de noter que les jetons paramètres

[5] DTS-Edit et les diagets ont été implémentés en C++.

doivent d'abord être définis dans cette partie pour pouvoir être utilisés lors de la spécification des questionnaires.

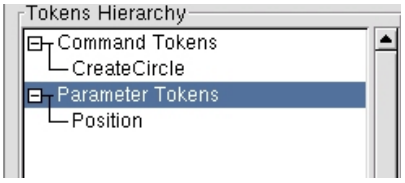


Figure 12. *Création d'un jeton « paramètre » de type position.*

Moniteur

La partie droite de la fenêtre principale représente le moniteur du contrôleur de dialogue. Sur ce panneau, le concepteur peut créer les interacteurs et les questionnaires qui les composent, ainsi que modifier l'organisation générale du moniteur.

Pour créer un interacteur, l'utilisateur donne simplement une chaîne de caractères correspondant à sa désignation (Figure 13). Les interacteurs sont affichés selon la hiérarchie du moniteur : les plus hauts dans la hiérarchie apparaissent au somment du panel.

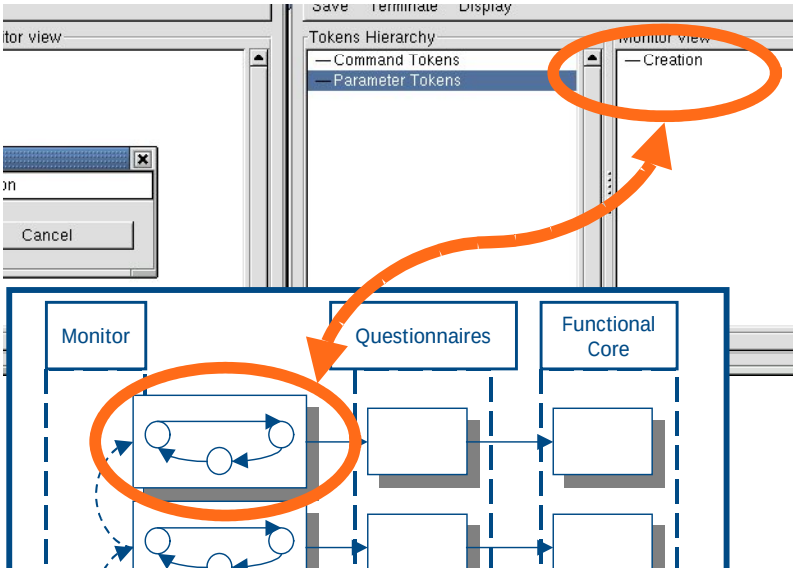


Figure 13. *Création d'un interacteur et conséquences sur le contrôleur de dialogue.*

Le concepteur de l'application peut alors ajouter un questionnaire à l'interacteur ainsi défini. Pour cela, il spécifie à nouveau une chaîne de caractères correspondant à sa désignation. Cette action a pour effet de générer automatiquement un jeton « commande », qui apparaît simultanément dans la liste des commandes de la partie gauche de la fenêtre.

L'étape suivante de la définition du contrôleur de dialogue consiste à spécifier la signature du questionnaire. Un menu permet au concepteur d'ajouter des paramètres d'entrée et éventuellement un paramètre de sortie aux questionnaires

existants. Les types de jeton utilisables pour cette action sont évidemment ceux qui ont été créés et qui apparaissent dans la partie gauche de la fenêtre.

DTS Edit n'est pas capable de générer seul le code des questionnaires. La spécification des signatures permet au logiciel de créer les automates représentant les différents interacteurs, en fonction de ces questionnaires. Les questionnaires eux-mêmes, qui font la liaison entre le contrôleur de dialogue et le noyau fonctionnel, doivent être programmés « à la main » par le concepteur. L'unique soutien qu'apporte DTS Edit à ce niveau de conception est un éditeur de texte permettant de composer les fichiers des questionnaires.

La dernière aide interactive qu'apporte l'éditeur est la possibilité de modifier interactivement, par « drag'n drop », la hiérarchie des interacteurs.

Vérification du dialogue

Une fois que le concepteur a terminé la description de tous ses jetons, qu'il a spécifié les interacteurs et la signature des différents questionnaires qui le composent, DTS Edit lui permet alors de générer automatiquement le code du contrôleur de dialogue de son application. À cette étape du processus, DTS Edit opère plusieurs vérifications sur les propriétés du dialogue.

1. La consistance globale du contrôleur de dialogue vis-à-vis de la production et la consommation des jetons est vérifiée en se basant sur l'organisation hiérarchique des interacteurs, sur les jetons qu'ils produisent et sur ceux qu'ils consomment. DTS Edit est ainsi capable de prévenir le concepteur de l'application dès qu'il détecte qu'un interacteur produisant un certain type de jeton T est placé au dessus de tous les interacteurs possédant une transition T, ou qu'inversement, un interacteur possédant une transition sur T est placé en dessous de tous les interacteurs produisant les jetons de type T.

2. Les risques de famine liés à la concurrence dans la consommation de jetons « commande » entre deux interacteurs de niveau différent peut aussi être vérifiée. En effet, si deux questionnaires, placés dans deux interacteurs différents, ont le même nom, l'un d'entre eux ne pourra jamais être activé. En effet, le jeton « commande » sera soumis à tous les interacteurs dans l'ordre de la hiérarchie, en commençant par celui placé le plus bas, mais il sera toujours intercepté par le même. Celui des deux interacteurs placés le plus bas dans la hiérarchie se verra toujours proposer le jeton en premier, et sera toujours en mesure de l'accepter. DTS Edit prévient l'utilisateur lorsqu'un tel cas se produit.

3. Enfin, les risques de comportement non déterministe d'un interacteur sont détectés. Ce cas se produit si, dans le même interacteur, deux questionnaires portent le même nom. Il s'agit d'une surcharge autorisée qui permet à l'utilisateur du système de commander la création d'un cercle par un unique bouton, qu'il définisse ensuite ce cercle par deux positions ou par une position et une valeur numérique. Cependant, un problème peut survenir si les paramètres d'entrée d'un des questionnaires sont une sous-liste des paramètres d'entrée de l'autre. Dans ce cas, il est en effet impossible au contrôleur de dialogue de savoir quelle attitude adopter :
   - lorsqu'il a reçu la liste de jetons du premier questionnaire, il décide de déclencher l'action correspondante (ce qui est possible puisque l'on dispose de tous les paramètres), cela signifie que, quoiqu'il arrive, le second ne se verra jamais proposer aucune séquence de jetons et il sera par conséquent virtuellement « gelé » ;

- en revanche, il attend l'arrivée de nouveaux jetons, essayant ainsi de compléter la liste du second questionnaire, le premier se trouve alors « oublié » et l'action correspondante ne sera jamais exécutée.

DTS Edit permet ainsi de vérifier certaines des propriétés du dialogue lors de la spécification de celui-ci. Ces vérifications permettent d'identifier une partie non négligeable des erreurs du concepteur. Cependant, certaines erreurs dans l'analyse des besoins ne sont pas prises en compte, c'est l'objet de la partie suivante, la validation du dialogue.

Conclusion sur DTS-Edit

La création du contrôleur de dialogue d'une application modélisée selon l'architecture H⁴ est un travail lourd et fastidieux de programmation. D'une part, le caractère répétitif des instructions mène souvent le concepteur à des erreurs dues par exemple aux nombreux copier/coller utilisés. D'autre part, l'organisation des interacteurs entre eux propose des propriétés permettant théoriquement de vérifier la validité et la complétude des tâches modélisées. Malheureusement, un tel travail n'est pas envisageable « manuellement ».

Pour ces raisons, nous avons mis au point un outil interactif qui répond aux besoins du concepteur : il génère une grande partie du code du contrôleur de dialogue en permettant une définition aisée des différents composants. D'autre part, il est capable de vérifier différentes propriétés de complétude des tâches en s'appuyant sur les propriétés et l'organisation des interacteurs.

3.3 Validation du dialogue

Objectif

Il n'est pas aujourd'hui possible de vérifier automatiquement, une fois un logiciel de type AGICT implanté, que les tâches prévues dans le cahier des charges sont bien réalisables par l'utilisateur. Ainsi, la recette d'un logiciel important peut-elle s'avérer fastidieuse. De même, il n'est pas aisé de vérifier que les modifications apportées lors d'une mise à jour n'ont pas altéré le dialogue homme-machine (e.g. tests de non-régression).

Il existe cependant des outils de test fonctionnel commerciaux comme *WinRunner* (Winrunner, 2006). Ceux-ci apportent effectivement une solution en permettant de rejouer des scripts au niveau présentation de l'interface, mais ils ne se basent pas sur une analyse de la tâche de l'utilisateur. Ainsi, ils ne valident qu'indirectement et partiellement le dialogue. On parle plutôt de vérification, car les scripts doivent être enregistrés un par un, avec une variabilité possible sur les données, mais à la charge du programmeur des tests. Ces scripts peuvent également avoir un aspect formel (Roché, 1998). D'autres approches font appel à une modélisation de la tâche de l'utilisateur mais ne valident qu'un modèle formel de l'interface, et non pas le code exécutable (Palanque et Bastide, 1995). D'autres approches encore se basent sur un modèle simple de la tâche mais ne s'intéressent qu'aux effets de bord de surface (« présentation ») de l'application (Memon, 2001).

Face à ce constat, notre objectif est de disposer d'un banc de test, qui, à partir d'un modèle de la tâche de l'utilisateur permet de valider *a posteriori* certaines propriétés du dialogue homme-machine (i.e. atteignabilité et complétude de la tâche) du code exécutable d'une AGICT implémentée selon le modèle d'architecture H⁴. Pour cela, nous avons défini précisément les propriétés vérifiées, adapté un modèle de tâches, développé un outil de génération de séquences de test, de soumission de ces séquences puis de détermination du succès d'un test

(problème de l'oracle). L'outil est décrit de manière plus détaillée dans (Jambon *et al.*, 1999). Nous illustrons ce travail par la validation du dialogue d'une étude de cas.

Propriétés vérifiées

Nous avons ainsi pour objectif de valider le comportement d'une AGICT suite aux sollicitations de l'utilisateur. Notons que nous ne nous intéressons ni aux retours d'informations vers l'utilisateur ni à la validité des fonctionnalités du système, mais seulement à son dialogue. Plus précisément, nous assurons deux des propriétés, dites externes, de l'utilisabilité des systèmes interactifs (Gram et Cockton, 1996) qui permettent de valider le dialogue homme-machine :

L'atteignabilité (« Reachability ») est la capacité du système à permettre à l'utilisateur de naviguer dans l'ensemble des états observables du système. Dans notre approche, nous vérifions que le système accepte bien la suite des interactions de l'utilisateur, reflet des tâches prévues dans le cahier des charges.

La complétude de la tâche (« Task Completeness ») est une propriété corollaire à l'atteignabilité. Nous vérifions que le système accepte non seulement les interactions prévues de l'utilisateur, mais aussi effectue les effets de bord correspondant au but de la tâche.

Indirectement nous validons également la **flexibilité** (« Flexibility ») de par la possibilité d'introduire dans le modèle des tâches plusieurs chemins d'interaction possibles pour obtenir un même but.

Modèle des tâches

Le modèle de tâches représente la spécification externe du dialogue. Ce modèle, spécifié *a priori* par des concepteurs assistés par des ergonomes, se doit d'être suffisamment abstrait pour permettre à ceux-ci d'exprimer l'activité prévue de l'utilisateur sans entrer dans des détails d'implémentation. Il doit également être suffisamment concret pour permettre la validation du dialogue en précisant l'ensemble des commandes sollicitées par l'utilisateur sur l'application.

Pour répondre à ces besoins, un modèle de tâche classique de type hiérarchique a été choisi. En effet, la décomposition hiérarchique des tâches permet le raffinement des buts de l'utilisateur jusqu'aux actions concrètes sur l'application. Ce modèle possède deux syntaxes : l'une graphique et l'autre textuelle. La possibilité de représenter les tâches sous forme graphique convient aux pratiques des ergonomes. En outre, la syntaxe textuelle formalisée par une grammaire permet l'automatisation du traitement du modèle des tâches en vue de la génération des séquences de test. La sémantique du modèle reste unique quelle que soit la syntaxe utilisée.

Le modèle de tâches que nous avons défini dans le contexte spécifique de la validation du dialogue structuré des AGICT est un sous-ensemble du modèle MAD (Scapin et Bastien, 2001). Le sous-ensemble de MAD utilisé limite les relations temporelles utilisables à la séquence (SEQ), l'alternative (OR), et l'ordre indépendant (AND) à l'exclusion de tout autre. De plus, quelques particularités propres au dialogue structuré ont été introduites. Notamment chaque but, quel que soit son niveau d'abstraction, est *a priori* associé à une commande du système (e.g. calculer une distance). Enfin, les feuilles du modèle, c'est-à-dire les actions atomiques, symbolisent des commandes de désignation d'objets du modèle ou plus généralement de fourniture de paramètres (e.g. sélectionner un point).

La figure 14 reproduit la modélisation en MAD simplifié de l'analyse de la tâche « Créer un cercle par centre et rayon », comme pourrait le faire un ergonome. Dans cet exemple, la modélisation a été volontairement limitée (une seule alternative a été spécifiée) afin de rendre celui-ci utilisable dans le cadre de cet article, l'étude

réelle comportant, elle, toutes les alternatives. Les rectangles symbolisent les commandes de l'utilisateur, tandis que les textes en italique sont des désignations directes d'objets graphiques ou des entrées de données numériques

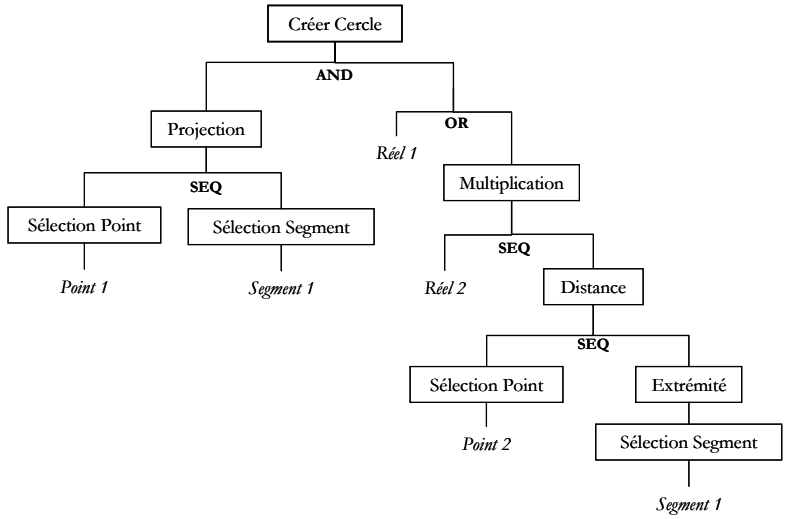


Figure 14. *Description en MAD simplifié de la tâche de création de cercle.*

Cette spécification externe du dialogue est ensuite traduite dans une forme textuelle définie par notre grammaire. Dans cette grammaire, les tâches ou sous-tâches sont numérotées par un identificateur unique. Elles commencent par les mots-clés *##tache*, respectivement *#sous-tache*, et se terminent par *##fin tache*, respectivement *#fin sous-tache*. En outre, le corps de chaque tâche contient trois champs :

Le champ **commande** indique quelle commande est activée par l'utilisateur, par exemple l'item de menu utilisé. Ce champ est utilisé par le simulateur pour initier le dialogue.

Le champ **résultat** indique le type d'objet du domaine que la tâche est supposée retourner ; dans notre exemple c'est une entité cercle. Ce champ est utilisé par le simulateur pour vérifier que la tâche a donné le résultat escompté.

Un ou plusieurs champs **paramètre** qui indiquent quels sont les objets dont a besoin la tâche pour arriver à son terme, et les moyens de les obtenir. C'est la traduction, dans un dialogue structuré, de la décomposition hiérarchique des tâches en sous-tâches. Chaque champ paramètre se compose de trois types d'information :

- le type d'objet requis, par exemple *position* ou *réel*.
- le moyen d'obtenir cet objet, ce peut être par une sous-tâche, dans ce cas elle est indiquée par son numéro entre chevrons, par exemple *<11>*, ou une tâche atomique, elle est alors directement indiquée, par exemple *une_position* ou *un_reel*. Bien entendu, le type d'objet retourné par la sous-tâche doit être identique au type d'objet requis.
- S'il y a plusieurs moyens possibles d'obtenir un paramètre, ou s'il est nécessaire d'obtenir plusieurs paramètres pour réaliser une tâche, un ou plusieurs opérateurs temporels sont insérés respectivement entre les groupes *type=moyen* (e.g. une position et un réel dans n'importe quel ordre) ou entre les champs paramètre (e.g. fournir un réel directement ou par un calcul). Ces opérateurs sont équivalents à ceux de MAD. Dans notre

grammaire, la séquence est l'opérateur par défaut, l'opérateur AND est représenté par le symbole «&» et l'opérateur OR par le symbole «/».

La figure 15 reproduit un extrait de la forme textuelle de la modélisation en MAD simplifié de la figure précédente. À l'heure actuelle, la transcription de la forme graphique à la grammaire s'effectue manuellement, mais ne présente pas de difficulté d'automatisation. À partir de cette forme textuelle sont générées les séquences de test.

```
## tache 1
    commande : creer_cercle
    resultat : cercle
    parametre : position=<11>
    & parametre : reel=un_reel / <13>

    # sous-tache 11
            commande : projection
            resultat : position
            parametre : point=<111>
            parametre : segment=<112>
    # fin sous-tache

    # sous-tache 13
            commande : multiplication
            resultat : reel
            parametre : reel=un_reel
            parametre : reel=<12>
    # fin sous-tache

# sous-tache 12
            commande : distance
            resultat : reel
            parametre : point=<121>
            parametre : position=<122>
    # fin sous-tache

    # sous-tache 122
            commande : extremite
            resultat : position
            parametre: segment=<1221>
    # fin sous-tache

…

## fin tache
```

Figure 15. *Description textuelle partielle de la tâche de création d'un cercle.*

Génération des séquences de test

De manière générale, un modèle des tâches permet de déterminer l'ensemble des scénarios d'interaction possibles pour réaliser une tâche donnée. Pour valider le dialogue, nous cherchons à générer l'ensemble des séquences valides d'actions de l'utilisateur sur l'application, lesquels doivent aboutir à la création d'un cercle. Pour cela, nous avons utilisé un parcours d'arbre classique de la description hiérarchique des tâches. Ces séquences d'actions utilisateur sur l'application ont été appelées *vecteurs de test*.

Notons que le parcours d'arbre de l'analyse de tâche nous donne tous les chemins possibles, et permet ainsi un test du dialogue avec une couverture complète. Ceci est rendu possible par l'absence de boucle dans la description des tâches. Lorsque le modèle des tâches comporte des boucles, une couverture complète est impossible. On se contente alors de générer les séquences, non plus pour tous les chemins, mais pour tous les i-chemins, i étant le nombre maximum de fois que le générateur entre dans une boucle, et est paramétrable.

Les vecteurs de test sont générés avec une syntaxe proche de celle des tâches. Chaque vecteur commence par le mot-clé *VECTEUR*, se termine par *FIN_VECTEUR*, et contient deux types d'actions : les commandes qui sont activées par l'utilisateur, par exemple des items de menu, ainsi que les paramètres fournis par celui-ci, par exemple les entrées de données ou des sélections d'objets. Un des vecteurs généré à partir du modèle des tâches précédent est reproduit en Figure 16.

```
VECTEUR
        commande : creer_cercle
        commande : projection
        commande : designer_point
        une_position
        commande : designer_segment
        une_position
        commande : multiplication
        un_reel
        commande : distance
        commande : designer_point
        une_position
        commande : extremite
        commande : designer_segment
        une_position
FIN_VECTEUR
```

Figure 16. *Un vecteur de test permettant la création d'un cercle.*

Soumission des séquences de test

L'application sous test est un prototype de logiciel de CAO, permettant la création d'objets géométriques 2D (Patry et Girard, 1999). Ce système, réalisé selon le modèle d'architecture H⁴, permet la réalisation de tâches structurées multi-objets. Ces tâches ont été spécifiées par le modèle des tâches utilisé comme étude de cas. L'objectif du test est de valider le dialogue. Pour cela, l'on se contentera de tester le module logiciel, qui, dans le cadre du modèle d'architecture H⁴, en est chargé, c'est-à-dire le contrôleur de dialogue.

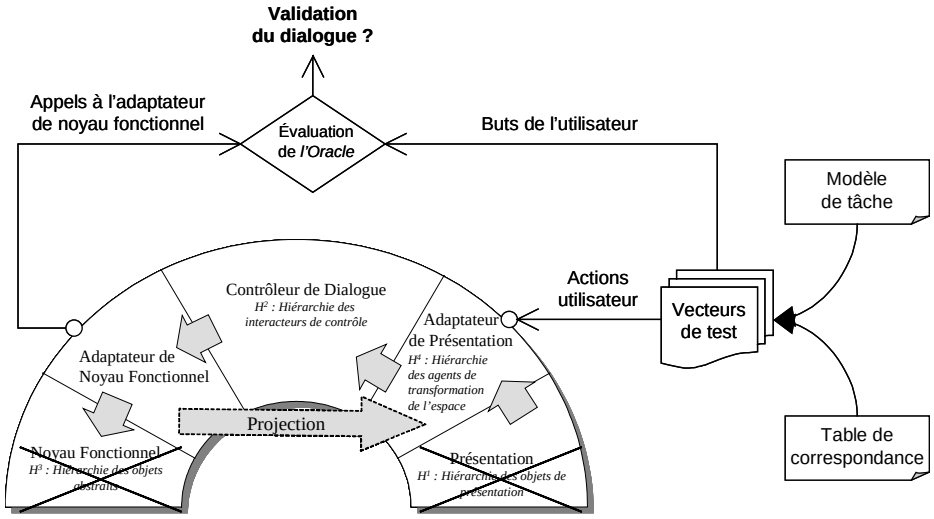


Figure 17. *Principe de fonctionnement du simulateur.*

Pour soumettre nos vecteurs de tests nous avons modifié le module présentation de l'application de manière à ce que les actions de l'utilisateur transitant

par la boîte à outils graphique soient remplacées par le contenu de nos vecteurs de tests (Cf. Figure 18). Ainsi le contrôleur de dialogue réel de l'application est sollicité comme si un utilisateur exécutait réellement ces actions. Le contrôleur a ainsi pu être testé sans aucune modification ni recompilation. Pour automatiser cette simulation, nous devons de plus traduire les actions spécifiées par l'ergonome dans le modèle des tâches par les réelles commandes logicielles de l'application. Pour cela, une table de correspondance a été établie. Cette table, reproduite en Figure 18, indique pour chaque action utilisateur (à gauche), l'entité informatique correspondante pour le contrôleur de dialogue (à droite).

```
commande : creer_cercle = COMMANDE : creation.cercle
commande : creer_rectangle = COMMANDE : creation.rectangle

un_reel : = REEL :
une_position : = POSITION :
un_segment : = SEGMENT :
un_point : = POINT :
un_cercle : = CERCLE :
un_rectangle : = RECTANGLE :
```

Figure 18. *Extrait de la table de correspondance.*

Cette table symbolise la collaboration nécessaire entre ergonomes et développeurs. Elle a une utilité multiple dans le processus de test. En effet, elle donne non seulement les équivalences entre les buts de l'utilisateur et les commandes internes de l'application, mais aussi entre les paramètres du point de vue de l'utilisateur et ceux vus par l'application. Il est ainsi possible de simuler les actions utilisateur pour la soumission des séquences de test, mais aussi de vérifier que les données fournies par l'application sont conformes aux attentes des utilisateurs, ce qui est utilisé pour déterminer l'oracle.

Problème de l'oracle

Le dépouillement des tests et la détermination de l'oracle, c'est-à-dire la référence qui permet de connaître le succès ou l'échec d'un test, s'effectue de manière symétrique à la soumission des tests. Tout d'abord, le contrôleur de dialogue, sollicité par les vecteurs de test, fait appel à un adaptateur de domaine modifié qui intercepte les appels au noyau fonctionnel. De fait, l'usage d'un véritable modèle géométrique est tout à fait inutile tant que l'adaptateur de domaine retourne le bon type d'objet au contrôleur de dialogue. Ceci ne pose pas de difficulté particulière car le type d'objet est spécifié par définition dans l'interface de programmation de ce module.

À l'issue de chaque soumission d'un vecteur de test, le banc de test vérifie que l'appel à l'adaptateur de domaine a bien pour conséquence la création d'un cercle, ce qui est l'effet de bord correspondant au but de l'utilisateur. Il utilise pour cela la table de correspondance. En pratique, nous avons identifié trois principaux types d'erreurs :

1. aucune commande de l'adaptateur de domaine n'est appelée à l'issue d'une interaction ;
2. le type d'objet retourné par l'adaptateur de domaine n'est pas celui attendu ;
3. la commande appelée n'est pas implémentée dans l'interface de programmation de l'adaptateur de domaine.

La première catégorie de faute est la plus répandue. Elle correspond à une erreur de programmation au niveau du contrôleur de dialogue, que nous désirons justement valider. Les deux suivantes correspondent en général à des erreurs de

programmation au niveau du modèle géométrique qui ont pour conséquence un échec du dialogue.

Les résultats obtenus à l'issue de la simulation permettent de s'assurer que tout va bien, ou bien qu'il y a au moins un problème. L'identification du problème est très liée à l'architecture H⁴, et est pour cette raison de la compétence du seul développeur, même si le problème rencontré provient de l'analyse de la tâche, domaine de l'ergonome.

Conclusion sur la validation du dialogue

Cette validation du dialogue s'effectue entre les deux extrémités du cycle de conception de l'application, avec d'un côté les spécifications de l'ergonome, et de l'autre l'application réelle. Ce processus de validation met le contrôleur de dialogue réel de l'application dans une situation où il est sollicité comme si un utilisateur exécutait réellement la séquence spécifiée et s'il faisait appel à un véritable modèle géométrique. Le type de test utilisé, se basant principalement sur des spécifications, est du type fonctionnel (boîte noire). En effet, nous n'avons aucune connaissance *a priori* du fonctionnement interne du contrôleur de dialogue. Cependant, la méthode utilisée impose un pré-requis sur l'architecture logicielle utilisée et nécessite le code source des modules présentation et adaptateur de domaine.

Deux limitations à cette approche peuvent être identifiées. La première concerne les limites du modèle de tâches utilisé. Celui-ci est très restrictif tant au niveau des opérateurs temporels (séquence, alternative et ordre indépendant) que des objets (aucune prise en compte des valeurs des paramètres). Ceci a pour conséquence de ne pouvoir valider des variations du dialogue basées sur les valeurs de ces paramètres. Pour lever ce verrou, il est nécessaire d'utiliser un modèle formel et suffisamment complet des tâches et des concepts utilisateur, comme celui proposé par (Lucquiaud, 2005). La seconde limitation concerne la nécessité de disposer d'une application utilisant le modèle d'architecture H⁴, et de disposer du code source d'une partie de ses modules. De plus, le banc de test impose une correspondance biunivoque entre les buts de l'utilisateur et l'interface de programmation de l'adaptateur de domaine.

Conclusion générale et perspectives

Tout au long de cet article, nous avons présenté le modèle H⁴ ainsi que certaines applications de ce modèle, tant dans le domaine de la conception des applications interactives que dans celui de la validation de ces mêmes applications. La description précise de la sémantique de ce modèle a en effet permis de construire plusieurs outils permettant d'exploiter plusieurs de ses particularités. Il a été utilisé tant dans des contextes de développement d'applications que dans le contexte de l'enseignement, où il a permis de faire appréhender aux étudiants les difficultés de la programmation du dialogue des AGICT.

Un des reproches que l'on peut faire à ce modèle est certainement sa trop grande orientation vers ces mêmes AGICT. En effet, s'il est aisé de construire des dialogues structurés préfixés avec cette architecture, il n'est pas facile de construire une application plus classique, de type WIMP. Il est encore moins facile de décrire des interfaces post-WIMP (Van Dam, 1997). Certains travaux, non décrits ici par manque de place ont été menés dans le but de prendre en compte la manipulation directe dans une application créée selon l'architecture H⁴, ce qui permet d'obtenir un écho sémantique proactif sans effort important de programmation (Patry, 1999) et (Depaulis, 2002). De façon plus générale, plusieurs idées mises en œuvre dans H⁴ ont été développées dans le contexte des interfaces post-WIMP. La notion de

production/consommation a ainsi été étudiée dans les « transducers[6] » (Accott *et al.*, 1997). D'autres travaux très récents préconisent l'utilisation de machines à états hiérarchiques (basées sur des *statecharts*), pour implémenter des dialogues post-WIMP (Blanch, 2002, Blanch, 2005). Il ne fait pas de doute que l'architecture H⁴ présente des points forts pour les applications post-WIMP, en particulier la possibilité de transformer les entrées en l'utilisateur en éléments de dialogue de plus haut niveau d'abstraction. Pour cela, il faudra cependant repenser très fortement le rôle du moniteur inclus dans le contrôleur de dialogue, ainsi que la nature exacte des jetons manipulés par le contrôleur de dialogue, mais provenant de la couche présentation.

Pourtant, en dehors des aspects liés aux dialogues structurés, la possibilité d'exprimer, d'implémenter, voire de valider le contrôleur de dialogue de H⁴, capable de gérer automatiquement de nombreux aspects fastidieux du dialogue à partir d'une description externe, est une piste fort intéressante pour la construction d'applications interactives plus sûres (Jambon *et al.*, 2001). Cette approche est d'ailleurs utilisée dans l'outil Petshop (Sy *et al.*, 1999), qui utilise les ICO (Interactive Cooperative Objects) pour modéliser ce même dialogue, ou l'outil SUIDT (Baron, 2004) qui permet d'assurer certaines propriétés du dialogue lors de la construction interactive de celui-ci à travers l'outil.

Parmi les axes de poursuite de travaux autour de H⁴, dans le domaine de l'ingénierie de l'interaction homme-machine, on peut citer la mise à disposition des outils réalisés, leur portage dans des langages couramment utilisés actuellement (Java, C#), ainsi que l'extension de ces outils pour leur faire prendre en compte des classes d'applications plus larges. Dans l'axe validation des applications interactives, la combinaison de ce modèle avec d'autres approches, comme celle des ICO (Palanque, 1992), mériterait d'être étudiée.

Contributions et remerciements

Ce travail n'est pas l'œuvre unique des auteurs de cet article mais l'œuvre collective de nombreux membres passés ou présents du LISI/ENSMA dont plus particulièrement Guy Pierra, Guillaume Patry, Guillaume Texier et Jean-Claude Potier. Les développements ont également fait appel à la perspicacité de nombreux stagiaires accueillis au laboratoire, dont Yohan Boisdron et Sabrina Maïano. Qu'ils en soient tous remerciés.

Bibliographie

Accott, J., Chatty, S., Maury, S., Palanque, P. (1997). Formal transducers: Models of devices and building bricks for the design of highly interactive systems. *Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'97)*, Granada, Spain, June 4-6, Springer-Verlag, 143-160.

Autebert, J. M. (1994). *Théorie des langages et des automates*. Masson.

Baron, M. (2004). *Vers une approche sûre du développement des Interfaces Homme-Machine*. Thèse de l'Université de Poitiers.

Blanch, R. (2002). Programmer l'interaction avec des machines à états hiérarchiques. *14ème conférence francophone sur l'Interaction Homme-Machine (IHM'02)*, Poitiers, France, ACM Press, 129-136.

[6] Sortes d'interacteurs incluant la notion de production et de consommation de jeton, et composables en couches

Blanch, R. (2005). *Facilitating post-WIMP Interaction Programming using the Hierarchical State Machine Toolkit*. Rapport de Recherche 1410, Laboratoire de Recherche en Informatique, Université Paris-Sud.

Coutaz, J. (1990). *Interfaces Homme-Ordinateur, Conception et Réalisation*. Dunod Informatique, Paris.

Coutaz, J., Nigay, L. (2001). Architecture logicielle conceptuelle des systèmes interactifs (chapitre 7). In Kolski, C. (Ed.), *Analyse et conception de l'I.H.M., Interaction Homme-Machine pour les S.I.*, vol. 1, Hermès Science, Paris, 207-246.

Depaulis, F. (2002). *Vers un environnement générique d'aide au développement d'applications interactives de simulations de métamorphoses*. Thèse de l'Université de Poitiers.

Depaulis, F., Maiano, S., Texier, G. (2002). DTS-Edit : an Interactive Development Environment for Structured Dialog Applications. In Kolski C., Vanderdonckt J. (Eds.), *Computer-Aided Design of User Interfaces III*, Kluwer Academics, 75-82.

Diaper, D., Stanton, N. A. (2003). *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates.

Dix, A., Finlay, J., Abowd, G., Beale, R. (1993). *Human-Computer Interaction*. Prentice Hall.

Duke, D. J., Harrison, M. D. (1993). Abstract Interaction Objects. *Computer Graphics Forum*, vol. 12, n° 3, 25-36.

Fekete, J.-D. (1996). *Un modèle multicouche pour la construction d'applications graphiques interactives*. Thèse de l'Université Paris-Sud, Orsay.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley.

Gardan, Y., Jung, J.-P., Kolopp, J.-N., Minich, C., Totino, W. (1988). Une approche nouvelle de la convivialité dans un système de CAO : les principes de dialogue dans SACADO. *MICAD*, Paris, 21-25 Mars, Hermès, 281-296.

Gardan, Y., Jung, J.-P, Martin, B. (1993). An End-User oriented approach to design man-machine interface for CAD/CAM. *IEEE International Conference on Systems, Man and Cybernetics*, Le Touquet, France, 17-20 Octobre, 525-530.

Girard, P., Pierra, G., Guittet, L. (1995). Les interacteurs hiérarchisés : une architecture orientée tâches pour la conception des dialogues. *Revue d'Automatique et de Productique Appliquée (RAPA)*, vol. 8, n° 2-3, 235-240.

Gram, C., Cockton, G. (1996). *Design Principles for Interactive Software*. Chapman & Hall.

Guittet, L. (1995). *Contribution à l'Ingénierie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO*. Thèse de l'Université de Poitiers.

Guittet, L., Pierra, G. (1993). Conception modulaire d'une application graphique interactive de conception technique : la notion d'interacteur. *Journées Francophones sur l'Ingénierie de l'Interaction Homme-Machine (IHM'93)*, Lyon, 19-20 Octobre, École Centrale de Lyon, 151-156.

Harrison, M. D., Duce, D. A. (1994). *A review of formalisms for describing interactive behaviour*. University of York.

Jambon, F. (2002). From Formal Specifications to Secure Implementations. In Kolski C., Vanderdonckt J. (Eds.), *Computer-Aided Design of User Interfaces III*, Kluwer Academics, 43-54.

Jambon, F., Brun, P., Aït-Ameur, Y. (2001). Spécifications des systèmes interactifs. In Kolski, C. (Ed.), *Analyse et conception de l'I.H.M., Interaction Homme-Machine pour les S.I.*, vol. 1, Hermès Science, Paris, 175-206.

Jambon, F., Girard, P., Boisdron, Y. (1999). Dialogue Validation from Task Analysis. *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'99)*, Universidade do Minho, Braga, Portugal, 2-4 June, Springer-Verlag, 205-224.

Krasner, G. E., Pope, S. T. (1988). A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80. *Journal of Object Oriented Programming*, vol. 1, n° 3, 26-49.

Lewis, C., Norman, D. A. (1986). Designing for Error. In Norman, D. A., Draper, S. W. (Eds.), *User Centered System Design - New Perspectives on Human-Computer Interaction*, Lawrence Erlbaum Associates, Hillsdale NJ, USA, 411-432.

Lucquiaud, V. (2005). *Sémantique et outil pour la modélisation des tâches utilisateur : N-MDA*. Thèse de l'Université de Poitiers.

Martin, B. (1995). *Contribution pour une nouvelle Approche du dialogue Homme-Machine en CFAO*. Thèse de l'Université de Metz.

Memon, A. M. (2001). *A Comprehensive Framework for Testing Graphical User Interfaces*. Ph.D. Thesis, University of Pittsburg.

Myers, B. A., Giuse, D., Dannenberg, R., Vander Zanden, B., Kosbie, D., Pervin, E., Mickish, A., Marchal, P. (1990). GARNET: Comprehensive Support for Graphical, Highly Interactive User Interfaces. *IEEE Computer*, vol. 23, 71-85.

Nigay, L. (1994). *Conception et Modélisation Logicielle des Systèmes Interactifs : Application aux Interfaces Multimodales*. Thèse de l'Université Joseph Fourier, Grenoble.

Norman, D. (1986). *User Centered System Design*. Lawrence Erlbaum Associates.

Palanque, P. (1992). *Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur*. Thèse de l'Université de Toulouse I.

Palanque, P., Bastide, R. (1995). Task Models - System Models: a Formal Bridge over the Gap. In Palanque, P., Benyon, D. (Eds.), *Critical Issues in User Interface Engineering*, Springer-Verlag, London, 65-80.

Paternò, F., Faconti, G.P. (1994). A semantics-based approach for the design and implementation of interaction objects. *Computer Graphics Forum*, vol. 13, n° 3, 195-204.

Patry, G. (1999). *Contribution à la conception du dialogue Homme Machine dans les applications graphiques interactives de conception technique : le système GIPSE*. Thèse de l'Université de Poitiers.

Patry, G. et Girard, P. (1999). GIPSE: a Model-Based System for CAD. *Third Conference on Computer-Aided Design of User Interfaces (CADUI'99)*, Louvain-la-Neuve, Belgique, 21-23 October, Kluwer Academics, 61-72.

Pfaff, G. E. (1985). User Interface Management Systems. *Workshop on User Interface Management Systems (Eurographic Seminars)*, Seeheim, November 1-3, Springer-Verlag.

Pierra, G. (1995). Towards a taxonomy for interactive graphics systems. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems*, Bonas, France, June 7-9, Springer-Verlag, 362-370.

Roché, P. (1998). *Modélisation et validation d'interface homme-machine*. Thèse de l'École Nationale Supérieure de l'Aéronautique et de l'Espace, Toulouse.

Scapin, D., Bastien, J.-M. C. (2001). Analyse des tâches et aide ergonomique à la conception : l'approche MAD*. In Kolski, C. (Ed.), *Analyse et conception de l'I.H.M., Interaction Homme-Machine pour les S.I.*, vol. 1, Hermès Science, Paris, 85-116.

Sy, O., Bastide, R., Palanque, P., Le, D.-H., Navarre, D. (1999). PetShop: a CASE Tool for the Petri Net Based Specification and Prototyping of CORBA Systems. *20th International*

Conference on Applications and Theory of Petri Nets (ICATPN'99), Williamsburg, Virginia, USA, June 21-25.

Tarpin-Bernard, F., David, B. (1999). AMF : un modèle d'architecture multi-agents multi-facettes. *Techniques et Sciences Informatiques*, vol. 18, n° 5, 555-586.

Texier, G. (2000). *Contribution à l'ingénierie des systèmes interactifs : Un environnement de conception graphique d'applications spécialisées de conception*. Thèse de l'Université de Poitiers.

UIMS (1992). The UIMS Workshop Tool Developers: A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, vol. 24, 32-37.

Van Dam, A. (1997). Post-WIMP user interfaces. *Communication of the ACM (CACM)*, vol. 40, n° 2, 63-67.

Van der Schaaf, T. W. (1997). Prevention and Recovery of Errors in System Software. *Workshop on Human Error and System Development*, Glasgow University, Scotland, 19-22 March, Glasgow Accident Analysis Group, 49-57.

WinRunner (2006). Mercury WinRunner Data Sheet (v8.2). Mercury Interactive Company (http://www.mercury.com).

Woods, W. (1970). Transition Network Grammars for Natural Language Analysis. *Communications of the ACM*, vol. 13, n° 10, 591-606.