Baron, M. Intégration d'un modèle de tâche dans une démarche sûre de construction d'interface. In *Proceedings of 14° Conférence Francophone sur l'Interaction Homme-Machine (IHM'2002)* (27-29 november, 2002, Poitiers), ACM Press, 2002, pp. 73-80

Intégration d'un modèle de tâche dans une démarche sûre de construction d'interface

Mickaël Baron

Laboratoire d'Informatique Scientifique et Industrielle
ENSMA, BP 40109, 86961 Futuroscope Cedex, France
http://www.lisi.ensma.fr/ihm
baron@ensma.fr

RESUME
La démarche présentée dans cet article propose de faire collaborer les approches « constructeur d'interface » (GUI-Builder) et « systèmes basés sur modèles » (Model Based System) afin de définir une méthode et un environnement de construction d'application interactives sûres. Nous nous concentrons tout particulièrement sur l'intégration d'un modèle de tâche dans notre démarche et nous décrivons les outils qui permettent de le manipuler.

MOTS CLES : Systèmes basés sur modèles, programmation visuelle, modèle de tâche, méthodes formelles.

ABSTRACT
The approach presented in this contribution is the collaboration of interface builders and model based systems in order to define a method and an environment for building safe interactive applications. We concentrate particularly on the integration of a task model, and we describe the interactive tools that allow editing and evaluating this model.

KEYWORDS : Model-Based Systems, Visual Programming, Task Model, Formal Methods.

INTRODUCTION

La prise en compte de l'analyse de tâche dans le développement des applications interactives, est, dans la pratique industrielle, peu utilisée. Les systèmes basés sur modèles tirent parti des spécifications de modèles, dont les modèles de tâche, pour construire et générer des applications interactives, comme le système Mobi-D [1] par exemple. Cependant, la généralisation de l'utilisation de ces systèmes est freinée par leur difficulté d'utilisation. En effet, cette dernière est réservée à des spécialistes en informatique, et nécessite généralement l'apprentissage de langages spécifiques. Les constructeurs d'interfaces, quant à eux, privilégient l'aspect présentation et permettent de réaliser le noyau fonctionnel sous forme de « callbacks » en code textuel au fur et à mesure de la construction de l'interface. Ils sont plus abordables que les systèmes précédents, car essentiellement basés sur la programmation visuelle. Mais l'inconvénient majeur des constructeurs d'interfaces est de se concentrer sur la programmation de surface de l'interface au détriment de la conception globale du système. Enfin, aucun modèle de tâche ne permet de raisonner sur l'interface construite.

Notre but consiste, à partir de ces deux approches, à définir une méthode et un environnement de construction d'application interactives sûres. D'une part, nous nous appuyons sur des modèles liés dont les sémantiques peuvent être clairement définies. D'autre part, nous fournissons au développeur un outil de développement interactif qui permet la manipulation des modèles dans le respect de leur sémantique.

Un premier travail a abouti au système GenBUILD et [2]. Cet outil génère automatiquement une application interactive à partir de l'interface d'un noyau fonctionnel préalablement développé. L'originalité de la démarche réside dans le fait que cette application générée automatiquement est couplée à un constructeur d'interface qui permet de réaliser une véritable application interactive tout en conservant le lien avec le noyau fonctionnel. Les limites majeurs du système GenBUILD sont cependant doubles. D'une part, l'absence du point de vue de l'utilisateur : aucun modèle de tâche n'est pris en compte. Le système ne peut garantir que les actions de l'utilisateur sur l'application générée peuvent être atteintes. D'autre part, la description du noyau fonctionnel du système s'appuie sur une sémantique pauvre, c'est-à-dire qu'aucun raisonnement sur le noyau ne peut-être effectué.

Nous proposons une nouvelle démarche qui vise à dépasser les limites de GenBUILD. Notre environnement interactif pour utilisateur final, appelé **SUIDT** (Safe User Interface Design Tool) valide cette démarche. Il regroupe plusieurs constructeurs de modèle : un noyau fonctionnel, un modèle de tâche (modèle de tâche abstrait + modèle de tâche concret) et un modèle d'interface.

Nous nous concentrons tout particulièrement dans cet article sur l'intégration du modèle de tâche. Après avoir

IHM 2002, November 26-29, 2002, Poitiers, France

rappelé les principes de notre méthode et défini l'exemple sur lequel seront basées nos illustrations, nous explicitons le modèle de tâche que nous utilisons et décrivons les outils qui permettent de le manipuler. Enfin, nous comparons notre approche à celle suivie par différents auteurs.

UN NOYAU FONCTIONNEL FORMEL

L'absence d'une vraie description formelle empêche le raisonnement sur le noyau fonctionnel. Un des points essentiels de notre approche est de pouvoir s'appuyer sur un noyau fonctionnel sûr, développé indépendamment de toute perspective d'interaction, et sur lequel des raisonnements peuvent-être effectués. Les outils que nous avons développés peuvent ainsi s'appuyer sur la sémantique des spécifications du noyau fonctionnel pour en garantir les propriétés. Notre démarche initiale, le système GenBUILD, à base d'expressions correspondant en fait à des pré / post-conditions, nous a conduit tout naturellement à choisir comme formalisme une méthode basée sur modèle, où l'essentiel de la sémantique peut s'exprimer sous la forme d'invariants et de pré et post-conditions.

Des méthodes du génie logiciel, et plus précisément les travaux sur les langages formels, explorent ces problèmes. Nous avons donc recherché parmi ces derniers, la méthode la plus adaptée à nos besoins. Parmi les approches basées sur modèles, VDM, Z et B sont les formalismes les plus largement diffusés. VDM [3] est le plus ancien ; c'est une notation mais aussi une méthode de développement formel. Cependant VDM ne contient pas de mécanismes permettant de décomposer/composer des spécifications et des raffinements de façon aisée. Z [4] est avant tout une notation pour faire des spécifications. Z n'est pas une méthode de développement et son axiomatique n'est pas clairement exprimée. Notamment la notion de pré-condition n'est pas explicitement définie. Enfin B [5] est la plus récente des trois notations. B est aussi une méthode de développement qui autorise la conception de l'application depuis la phase de spécification jusqu'à la phase d'implémentation. Elle est basée tout comme VDM et Z sur la théorie des ensembles, mais elle contient en revanche un mécanisme de structuration (la machine abstraite), la notion de raffinement et atteint l'implémentation.

Nous avons choisi la méthode B car c'est une méthode couvrant tout le spectre depuis la spécification jusqu'au code. De plus la pertinence de la méthode B est d'assurer le respect des propriétés, exprimées au moment des spécifications, tout au long du processus de développement. Enfin, le langage B est parfaitement instrumenté par l'environnement de développement « Atelier B » [6]. Nous ne nous étendrons cependant pas sur la liaison entre la méthode B et notre approche, qui est décrite dans [7]. Nous allons à présent étudier un noyau fonctionnel formel en B de l'application convertisseur Francs/Euros qui nous servira d'illustration.

LE CONVERTISSEUR FRANC/EURO

Le convertisseur est un logiciel utilitaire permettant de réaliser des conversions d'une somme en Francs vers son équivalent en Euros et vice versa. L'utilisateur entre la valeur qu'il désire convertir, choisit le sens de la conversion, déclenche la conversion, puis lit le résultat dans la monnaie souhaitée. Il y a de nombreuses possibilités d'interface pour cette application, des interfaces de types calculatrice ou des interfaces très simples, voir Figure 1.

Figure 1 : Interfaces du convertisseur Francs/Euros (a) et (b) .

NOYAU FONCTIONNEL DU CONVERTISSEUR FRANCS/EUROS

Le noyau fonctionnel du convertisseur Francs/Euros est très simple mais il respecte les services préconisés par [8]. Une première primitive initialise les variables de l'application *nf_start*. Deux primitives de modification (*conversion_euro_franc* et *conversion_franc_euro*) modifient le sens de conversion. Deux autres méthodes réalisent la conversion (*entrer_somme* et *convertir*). Les deux dernières primitives retourne le résultat de la conversion (*afficherfe* et *afficheref*). Il y a d'autres façon de concevoir le noyau fonctionnel. Mais l'important est de souligner qu'avec ces fonctions, il est possible de concevoir les deux interfaces de la Figure 1 (a et b).

La séparation stricte entre noyau fonctionnel formel et interface nous conduit à distinguer deux profils très différents pour la réalisation d'applications interactives à l'aide de SUIDT. La conception du noyau fonctionnel formel ne peut être effectuée que par un spécialiste de ces méthodes, maîtrisant également les aspects classiques de la programmation (pour la phase de raffinement). A l'inverse, la construction de l'interface utilisateur n'impose aucun pré-requis en matière de connaissances informatiques. A condition de lui fournir les outils adéquats, un utilisateur final (end-user) au sens de [9] est à même de construire une application interactive en totale cohérence avec les contraintes du noyau fonctionnel formel, ce que nous qualifions une application interactive sûre. Nous appellerons par la suite « concepteur d'application interactive » ce profil particulier. Nous décrivons dans les sections suivantes les différents outils

qui permettent d'atteindre ce résultat, et en particulier par l'utilisation du modèle de tâches utilisateur.

UN GENERATEUR D'INTERFACE ET UN ANIMATEUR DE NOYAU FONCTIONNEL

La méthode utilisée dans GenBUILD s'adapte directement à un noyau fonctionnel exprimé par la méthode B. L'analyse des spécifications du noyau fonctionnel permet de générer automatiquement une interface de manipulation de ce même modèle. L'interface affiche les fonctions du noyau fonctionnel formel et permet de contrôler s'il fonctionne ou pas. Il assure également visuellement que les invariants sont établis.

Pour générer cette interface de développement, l'outil récupère toutes les informations des opérations des machines B, c'est-à-dire le nom de chaque opération et les conditions des paramètres d'entrées-sorties. Ces conditions donnent des renseignements sur les prédicats de typage et les prédicats de propriétés. Ainsi l'animateur de noyau fonctionnel affiche chacune des opérations en les associant à des objets graphiques. Des boutons sont utilisés pour lancer une opération alors que des zones de textes sont employées pour afficher les informations et l'état des opérations du noyau fonctionnel. Cette phase est entièrement automatique, à partir des spécifications exprimées en B.

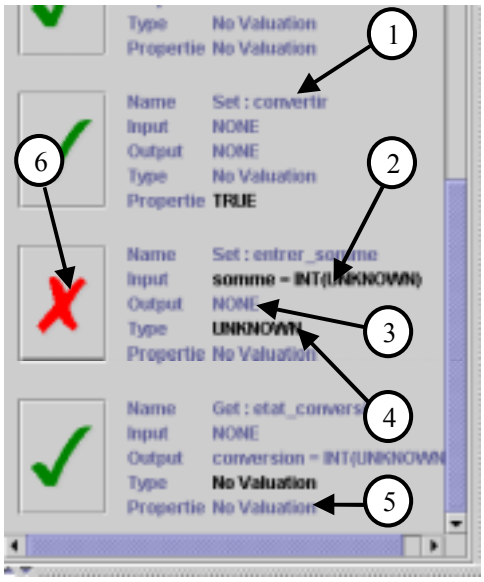

Figure 2 : Animateur de noyau fonctionnel

La Figure 2 représente l'outil qui anime le noyau fonctionnel. Toutes les informations des fonctions sont affichées. Le champ n°1 symbolise le nom de la fonction. Le champ n°2 affiche les informations des paramètres d'entrée (le nom du paramètre, son type et sa valeur courante). De même le champ n°3 retourne les informations des paramètres de sortie. Les champs n°4 et n°5 donnent les informations des prédicats de typage et de propriétés. L'animateur évalue à chaque nouvelle modification de l'état du noyau fonctionnel le prédicat du champ n°4 et n°5 ce qui autorise alors l'activation du champ n°6 associé à un bouton. C'est le concepteur d'application interactive qui interagit avec l'animateur en pressant les boutons associés aux fonctions du noyau fonctionnel indiquant ainsi qu'elles doivent être déclenchées tout en respectant les pré/post conditions.

L'avantage de cette approche est de disposer d'un noyau fonctionnel que l'on peut tester et utiliser pendant le développement de l'application. Cependant, dans le cas d'une véritable application, l'appel des fonctions du noyau doit suivre un (ou des) scénario-(i) précis. C'est la raison pour laquelle nous avons inclus dans notre approche un modèle de tâche.

LE MODELE DE TACHE DANS SUIDT

De nombreuses méthodes de développement d'applications interactives préconisent d'effectuer au préalable une analyse de l'activité [10] pour construire un modèle de tâche. Au premier abord, cette analyse doit être indépendante de toute idée d'interface et elle ne doit pas être composée d'insinuations sur les systèmes d'interaction à implémenter. C'est une vue abstraite du fonctionnement de l'application sans élément de l'interface. Il a semblé tout à fait naturel d'introduire la notion de modèle de tâche dans notre approche, afin de permettre de décrire au mieux l'application du point de vue de l'utilisateur. C'est à partir de ce modèle de tâche, et dans le respect des contraintes du noyau fonctionnel, que le concepteur d'application interactive sera à même de construire son application.

Les travaux dans le domaine des modèles de tâche sont nombreux, et parfaitement adaptés à cette phase de l'analyse. Dans le même esprit que celui dans lequel nous avons recherché une méthode formelle utilisable pour nos besoins, nous avons recherché une méthode de description de tâches qui nous permette d'exprimer cette analyse de tâche au sein de notre outil. Une notation graphique, à sémantique parfaitement définie, et si possible disposant d'outils d'analyse connectables à notre environnement de développement conviendra à notre besoin.

GOMS (Goals Operators Methods Selection rules) [11], UAN (User Action Notation) [12], MAD (Méthode analytique de description des tâches) [13] et CTT (ConcurTaskTrees) [14] sont des formalismes de modélisation de tâche qui semblent à première vue de bons candidats. Ils ont cependant des syntaxes (textuelles ou graphiques), des niveaux de formalité et des opérateurs de décomposition de tâches différents. GOMS, le plus ancien, se limite à la description des opérateurs temporels, prenant en compte uniquement les tâches séquentielles. UAN en revanche, est une notation pour la description d'interaction basée sur une syntaxe textuelle, parfaitement conçue pour décrire des interactions de bas niveau

(comme le clic souris, par exemple). Sa sémantique est cependant mal définie, et elle ne semble pas adaptée pour traiter des applications volumineuses. MAD et CTT sont des formalismes à syntaxe graphique et comportant des opérateurs temporels riches. Au regard de nos besoins, ils s'avèrent très proches. Notre choix s'est porté sur CTT, et a été motivé par le fait que CTT est supporté par l'outil CTTE [14] qui s'avère relativement ouvert sur l'extérieur (possibilité d'exporter et d'importer des modèles en XML).

Dans la suite de l'article, nous décrirons rapidement le formalisme CTT et nous présenterons notre utilisation de ce formalisme.

Le formalisme CTT

Selon ses auteurs, CTT est une notation de spécifications de modèle de tâche définie pour surmonter les limitations des notations préalablement utilisées pour concevoir des applications interactives. Sa principale caractéristique est d'être une notation facile à utiliser pour la conception de réelles applications industrielles. Le modèle de tâche de CTT est basé sur trois points.

- Elle est orientée action utilisateur, et propose une structure hiérarchique des tâches (une structure en forme d'arbre) ;
- La sémantique de CTT repose sur un formalisme temporel bien connu, Lotos. Les relations temporelles entre tâches sont ainsi parfaitement exprimées ;
- Les tâches peuvent manipuler des objets qui peuvent communiquer entre eux.

Le formalisme CTT inclut quatre catégories de tâche (Tableau 1).

	Les **tâches utilisateurs** réalisées entièrement par l'utilisateur.
	Les **tâches applications** effectuées complètement par le système.
	Les **tâches interactions** réalisées par les interactions de l'utilisateur avec le système.
	Les **tâches abstraites** raffinées par les trois catégories précédentes.

Tableau 1: Catégories du formalisme CTT

Un modèle abstrait et un modèle concret

Le formalisme que nous avons choisi est adapté pour la description de tâches afin d'exprimer les besoins des utilisateurs, souvent des non-informaticiens. Il s'agit de définir les tâches sans entrer dans les détails de la réalisation de l'interface. Dans notre exemple de convertisseur Francs/Euros, on peut ainsi distinguer la tâche, que nous qualifierons **d'abstraite** (ex : *choix du sens de conversion)*, de la tâche **concrète** permettant de la réaliser sur l'interface, (par exemple le clic sur un bouton ou la sélection d'une case dans un bouton radio).

Cette distinction nous a amené à définir deux niveaux distincts de modèles de tâche, que nous avons appelés respectivement « modèle de tâche abstrait » et « modèle de tâche concret ». Dans le modèle de tâche abstrait, aucun choix de technique d'interaction ne sera autorisé. A l'inverse, le modèle de tâche concret est destiné uniquement à concrétiser en terme d'interactions les feuilles de l'arbre de tâches abstrait. De ce fait, il s'appuie complètement sur ce dernier, et consiste en un raffinement de l'arbre abstrait.

Nous verrons par la suite de cet exposé comment cette distinction permet de séparer les activités de conception et surtout de validation.

Le modèle de tâche abstrait basé sur la sémantique de CTT

Afin de définir notre modèle de tâche abstrait, nous avons utilisé CTT en le restreignant. Cependant, nous avons systématiquement évité de modifier la sémantique de CTT, afin de conserver ses propriétés formelles. Dans le formalisme CTT initial, chaque tâche est associée à un élément de l'application (éléments du noyau fonctionnel, éléments de l'interface…) qui est appelé pendant l'activation de la tâche. De façon plus restrictive, notre modèle de tâche abstrait n'autorise que la manipulation d'objets venant du noyau fonctionnel. En particulier, le formalisme CTT initial autorise l'association d'objets à tous les niveaux du modèle de tâche, ce que nous n'autorisons pas dans le modèle de tâche abstrait. Seules les tâches de plus bas niveau (les feuilles, en considérant le modèle de tâche abstrait défini par un arbre) sont associées à des fonctions du noyau fonctionnel. Ainsi, au niveau du modèle de tâche abstrait est défini un ordre d'appel aux fonctions du noyau fonctionnel par l'intermédiaire d'un modèle appelé **modèle de fonctions**. Celui-ci définit hiérarchiquement l'appel des fonctions. Chaque tâche de plus bas niveau du modèle de tâche abstrait est associée à un modèle de fonctions. La sémantique de ce modèle est aussi basée sur celle de CTT. Il s'agit d'un arbre à un seul niveau où chaque nœud est associé à une fonction du noyau fonctionnel. Chaque nœud est séparé par une relation temporelle.

L'EDITEUR DE MODELE DE TACHE ABSTRAIT

Nous avons réalisé un outil permettant de construire le modèle de tâche abstrait de l'application. Dans notre démarche, le concepteur du modèle abstrait est le « concepteur d'application interactive », que nous avons défini précédemment. Nous avons donc utilisé de nombreuses techniques pour utilisateur final, et plus particulièrement la programmation visuelle (« visual programming ») et la programmation sur exemple (« programming by demonstration »). La conception du modèle se fait dans une zone de construction appelée « Vue du modèle de tâche abstrait », *repère 2* de Figure 3. Le modèle de tâche abstrait est représenté par un arbre

où les nœuds représentent soit des tâches, soit des relations temporelles.

La vérification de ce modèle de tâches peut s'effectuer en utilisant l'outil CTTE. Nous avons ainsi prévu une communication avec cet outil ; cependant, dans l'état actuel de son développement, CTTE permet d'exporter un modèle CTT en XML, mais pas d'en importer. Cette méthode sera néanmoins toujours limitée car l'outil CTTE ne peut être connecté au noyau fonctionnel. Afin d'aller plus loin dans la vérification du modèle de tâche le plut tôt possible SUIDT offre deux approches de test qui restent complémentaires de CTTE.

La première consiste à utiliser un outil intégré, le **simulateur**. Ce dernier analyse le modèle de tâche abstrait pour connaître l'ensemble des tâches actives à un moment précis. Le concepteur d'application interactive peut alors choisir parmi cet ensemble une tâche active qui pourra être évaluée. Si les fonctions activées possèdent des paramètres, un ensemble de boîtes de dialogue est utilisé pour qu'il puisse saisir les paramètres. De même les fonctions avec des valeurs de retour affichent leurs résultats. Enfin le simulateur évalue à chaque modification le modèle de tâche jusqu'à ce qu'il n'y ait plus de tâches actives. Il est possible d'appeler ou sauvegarder des scénarios de fonctionnement.

Une seconde approche consiste à tester le modèle de tâche abstrait pendant son édition : le concepteur d'application interactive peut éditer et simuler le modèle de tâche abstrait tout en conservant le contexte d'exécution du noyau fonctionnel.

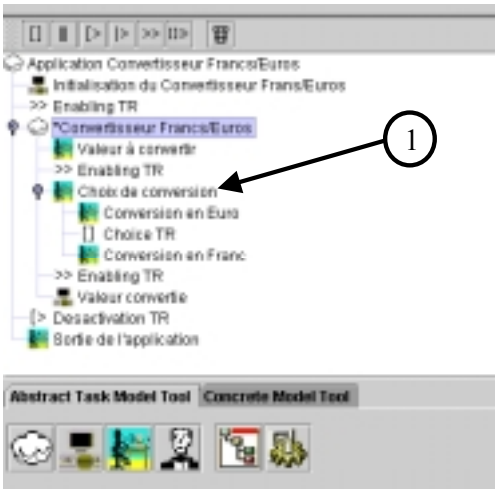

Figure 3 : Edition d'un modèle de tâche abstrait: (le convertisseur Francs/Euros)

A ce stade, on peut faire une rapide comparaison avec le simulateur de l'outil CTTE : ce dernier vérifie uniquement les incohérences (boucle sans fin, tâche inatteignable) du modèle de tâche et les spécifications du cahier des charges. Il n'exécute pas les actions, et se contente d'afficher les tâches actives. SUIDT est relié aux attributs des références des objets du noyau fonctionnel qui seront modifiées pendant la simulation. Ainsi le simulateur de SUIDT permet d'exécuter réellement les actions, et de vérifier que les appels du noyau fonctionnel sont cohérents.

LE MODELE DE TACHE ABSTRAIT DU CONVERTISSEUR FRANCS/EUROS

Illustrons le modèle de tâche abstrait sur l'exemple du convertisseur Francs/Euros. L'outil que nous avons développé diffère notablement des outils graphiques utilisés pour la modélisation des tâches. En effet, à l'inverse de CTTE par exemple, qui représente ses modèles sous formes d'arbres complètement graphiques, nous avons choisi une représentation plus schématique, basée sur les widgets « arbres » désormais classiques, qui permettent de s'affranchir de nombreux problèmes d'édition purement graphiques. Il n'est cependant pas certain que cette approche soit pertinente, et des tests d'utilisabilité devront être conduits pour aboutir à une version industrialisée de nos outils.

La tâche de plus haut niveau est *Application Convertisseur Francs/Euros*. C'est une tâche abstraite décomposée en 3 sous tâches. La tâche application *Initialisation du Convertisseur Francs/Euros* associée à un modèle de fonctions comportant la fonction *nf_start* initialise l'application à son lancement. La tâche Convertisseur Francs/Euros est itérative (symbolisé par le caractère *), elle représente l'action principale de l'application. L'utilisateur saisit la valeur à convertir (*entrer_somme*). Il choisit ensuite le sens de conversion en Euros (tâche utilisateur *Conversion en Euros)* **ou** en Francs (tâche utilisateur *Conversion en Francs)* associées aux fonctions *conversion_euro_franc ou conversion_franc_euro.* Finalement la tâche application *valeur_convertie* calcule la valeur et affiche le résultat par l'intermédiaire des fonctions *convertir* et *afficherfe* et *afficheref.* Tant que l'utilisateur ne choisit pas la tâche utilisateur *Sortie de l'application* la tâche abstraite itérative *Convertisseur Francs/Euros* se répète. Tout au long de la simulation l'utilisateur final vérifie les appels aux fonctions du noyau fonctionnel. Il choisit les tâches à activer et les valeurs des paramètres des fonctions à saisir. Ce faisant, il procède déjà à une phase de test.

Nous avons défini un modèle de tâche abstrait qui nous permet de modéliser la logique de spécification du noyau fonctionnel. Nous ne l'avons pas montré mais, avec ce même noyau fonctionnel, il est possible d'obtenir différents modèles de tâche abstrait. Cependant, quand le concepteur d'application interactive conçoit le modèle de tâche abstrait, il a déjà une vision globale de la logique de l'interface de l'application. Dans la phase suivante, il va pouvoir concrétiser son interface, tout en restant dans la logique de son modèle de tâche abstrait

LE MODELE CONCRET

Dans cette partie, nous détaillons ce que nous appelons le modèle concret. Il s'agit d'un modèle de tâche, tout comme le modèle de tâche abstrait, mais qui associe les objets des tâches aux éléments de l'interface (interaction de l'utilisateur sur l'application, retour d'informations…). Le formalisme CTT ne permet pas de décrire explicitement le type d'interaction que l'utilisateur emploie à tout moment sur l'application. Cette information n'est obtenue que par la combinaison d'une catégorie *tâche interaction* et de ses objets contenus. Notre objectif est de pouvoir à la manière de UAN décrire une décomposition de bas niveau des interactions ou des modifications de l'interface, ceci dans le but de connaître l'ensemble des possibilités d'interaction que l'utilisateur effectue sur l'interface. Chaque *tâche interaction* de bas niveau du modèle de tâche est une tâche qui doit être décomposée en sous-tâche pour décrire l'interaction. Cette décomposition en sous-tâche est ce que nous appelons le modèle concret. C'est un raffinement du modèle de tâche abstrait du point de vue des tâches d'interaction et d'application. Ce modèle implémente les actions et les interactions de l'application.

Le modèle concret basé sur le formalisme CTT

Comme pour le modèle de tâche abstrait, nous avons utilisé ici le formalisme CTT. Les catégories existantes du formalisme CTT ne permettent pas de décrire précisément la décomposition, aussi proposons-nous plusieurs nouvelles catégories qui dépendent soit des tâches interactions, soit des tâches applications. Nous ajoutons aux **tâches interactions** plusieurs catégories qui symbolisent l'ensemble des interactions sur les widgets proposés pour concevoir une interface graphique. Par exemple dans le cas de notre application, il y a une interaction de type « un clic sur le bouton gauche de la souris » abonné à un bouton. Cette interaction et le bouton sont considérés comme une nouvelle catégorie. Ensuite il faut associer aux catégories interactions les références aux objets du noyau fonctionnel. Les objets du noyau fonctionnel sont les fonctions disponibles uniquement dans le modèle de fonctions de la tâche abstraite raffinée. Ce raisonnement s'applique aussi aux **tâches applications** où uniquement deux autres tâches sont ajoutées, les **tâches systèmes** et les **tâches feedback**. Les tâches systèmes appellent des objets du système (une impression par exemple) alors que les tâches feedback servent à modifier l'affichage de l'interface après une interaction. Notons que ces ajouts ne modifient en rien la sémantique de CTT, puisqu'il ne s'agit nullement de toucher aux opérateurs de synchronisation. La seule entorse au formalisme CTT réside dans le fait que, alors qu'en CTT « pur », une tâche ne peut-être décomposée en une seule sous-tâche (deux sous-tâches sont nécessaires au minimum), le passage du modèle abstrait au modèle concret peut se faire par une décomposition en un seul fils (si l'interface ne propose pas deux manières d'exécuter une tâche).

Le modèle de tâche concret du convertisseur Francs/Euros

Le repère n°1 de la ***Figure 4*** montre la tâche Conversion en Euro raffinée par deux sous-tâches interaction : une première tâche symbolise un simple clic sur un bouton de l'interface et une deuxième tâche symbolise un raccourci clavier. Le rôle du concepteur de l'application interactive est de lier chaque modèle. Il y a le noyau fonctionnel par l'intermédiaire de la fonction *conversion_franc_euro*, cette fonction a été associée pendant l'édition du modèle de tâche à la tâche Conversion en Euro. Ensuite le concepteur d'application interactive précise deux éléments : l'interaction click gauche de la souris sur un bouton et un raccourci clavier qui sont des catégories cataloguées par la boîte à outils de l'outil GUI-Builder. Cet outil permet aussi de construire l'interface de l'application.

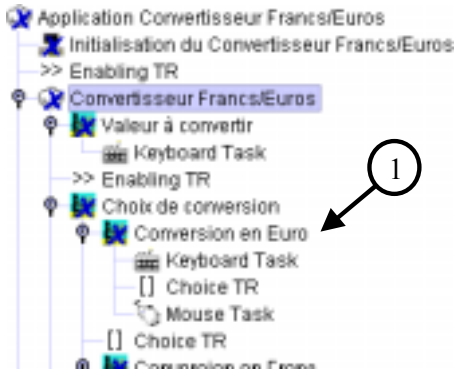

Figure 4: Extrait du modèle de tâche du convertisseur

La simulation du modèle de tâche concret se déroule de manière identique à la simulation du modèle de tâche abstrait. La différence avec la simulation du modèle de tâche abstrait est que l'interface est prise en compte : au fur et à mesure de l'évaluation du modèle concret l'état de l'interface sera modifié. Cette liaison avec le constructeur d'interface est actuellement en cours de réalisation.

AUTRES APPROCHES

Comme nous l'avons mentionné en introduction, les outils qui construisent des applications graphiques interactives en prenant en compte un modèle de tâche sont aujourd'hui peu nombreux. En parcourant les offres logicielles de conception d'application, deux approches de conception logicielle avec intégration de modèle de tâche se distinguent. Tout d'abord il y a une approche à deux niveaux : le modèle de tâche est édité et validé par un outil et ensuite l'application est construite par un constructeur d'interface. L'intérêt de cette approche est l'utilisation de techniques de programmation visuelle qui facilitent l'utilisation des outils. Mais en revanche la difficulté est d'appliquer les besoins exprimés du modèle de tâche à la construction de l'application. Une deuxième approche qui se situe plus du côté des systèmes basés sur modèles, est une intégration dans l'outil de conception

d'un ensemble de modèles qui sont liés les uns aux autres. Cette approche a l'inconvénient de ne pas disposer d'outils pour l'utilisateur final mais offre l'avantage de générer une application qui correspond aux spécifications du modèle de tâche.

CTTE (ConcurTaskTrees Environment) est un outil d'édition et de validation de modèle de tâche basé sur la sémantique de CTT. La représentation de la hiérarchie des tâches se fait graphiquement sous forme d'un arbre. VTMB [15] est aussi un outil d'édition et de validation (Visual Task Model Builder). Cet outil représente simultanément le modèle de tâche graphiquement et textuellement, en effet à chaque modification du modèle il est capable de générer du code à partir de la représentation graphique. Ainsi la modification est possible dans les deux représentations. A la différence de MASTERMIND [16] où les spécifications des modèles se font textuellement dans un éditeur, notre outil est proche de CTTE ou VTMB car il offre aussi une édition et une visualisation graphique du modèle.

Mais ces outils ne permettent pas d'associer aux tâches des références aux objets des autres modèles de l'application (noyau fonctionnel, présentation) car il n'y a pas de liaison avec les autres modèles. SUIDT à l'inverse regroupe un noyau fonctionnel, un modèle d'interface et un modèle de tâche tout comme les systèmes basés sur modèle. L'inconvénient majeur de la non prise en compte des références des objets des autres modèles se situe au niveau de la validation du modèle de tâche. Les simulateurs de CTTE et VTMB contrôlent le modèle de tâche de toutes sortes d'incohérences (boucle sans fin, tâche inatteignable) et vérifie aussi l'exactitude avec les spécifications du cahier des charges. Notre outil SUIDT vérifie aussi les erreurs de constructions du modèle de tâche mais il assure que les appels du noyau fonctionnel et de l'interface sont cohérents. Dans CTTE, les tâches qui sont activées par l'utilisateur ne modifient pas les modèles de l'application. Ce qui n'est pas vrai dans SUIDT, où les attributs des références des objets associées aux tâches sont modifiés : l'état de l'application est ainsi modifié. Cette distinction est flagrante au niveau de la simulation du modèle de tâche abstrait de SUIDT. Toute fonction qui nécessite une valeur de paramètre interroge alors l'utilisateur. Ce dernier interagit avec tous les modèles de l'application alors que le simulateur de CTTE et de VTMB ne demande qu'une connaissance du cheminement des tâches sur l'application.

PetShop [17] est un environnement logiciel permettant la spécification comportementale, la validation, le prototypage de systèmes interactif fondé sur les réseaux de Petri de haut niveau. Même si cet outil s'intéresse à la modélisation de système et qu'il s'éloigne du domaine de SUIDT, il est intéressant de comparer ces deux approches. En effet la démarche de PetShop et celle de SUIDT ont deux grandes similitudes : la simulation du modèle et l'absence de mode d'utilisation. CTTE et VTMB ont une interface utilisateur modale. L'utilisateur alterne entre les modes d'éditions (pour modifier la structure du modèle de tâche) et les modes de simulations (pour vérifier et exécuter le modèle de tâche). Dans SUIDT et PetShop, cette séparation des modes n'existe pas. Le modèle de tâche est en permanence édité, vérifié et exécuté. Cependant SUIDT propose deux aspects de modalité (soit la séparation des modes soit la simulation du modèle de tâches pendant la construction de l'application). Il incombe au concepteur de l'application interactive de choisir sa façon de concevoir. Cette absence de modalité n'est possible que par la présence de liens entre le modèle de tâche et le noyau fonctionnel et l'interface. La différence la plus importante avec les autres approches est ainsi la possibilité d'interagir avec le noyau fonctionnel en s'appuyant sur le modèle de tâche abstrait, et d'interagir avec l'interface en s'appuyant sur le modèle concret.

Enfin, il est important de noter que notre méthode a l'avantage de donner au concepteur de l'application interactive la possibilité d'alterner la phase de conception et celle de test sans perdre le contexte d'exécution.

CONCLUSION ET PERSPECTIVES

Nous avons présenté dans ce papier l'approche SUIDT. C'est un outil de conception assisté par ordinateur d'application interactives utilisateurs qui s'appuie sur plusieurs formalismes déjà définis (noyau fonctionnel formel en B, CTT).

L'idée principale est de partir d'un noyau fonctionnel formel développé par un spécialiste en informatique. L'outil animateur de noyau fonctionnel aide alors le concepteur d'application interactive à tester le noyau sans connaissance dans la programmation classique. La logique d'appel des fonctions est établie par le modèle de tâche abstrait. Il définit une structure hiérarchique des appels des fonctions. Enfin le modèle concret raffine les tâches de plus bas niveau du modèle de tâche abstrait dans l'objectif d'introduire les états de l'interface de l'application (gestion de l'interaction, gestion de l'affichage). En d'autres termes, un modèle de tâche (composé du modèle de tâche abstrait et du modèle concret) lie les éléments de l'interface avec les fonctions du noyau fonctionnel, ce modèle de tâche s'occupe de la gestion du dialogue de l'application.

Un grand nombre de réflexions reste encore à mener. Tout d'abord la faisabilité de l'approche a été expérimentée sur un exemple limité, le convertisseur Francs/Euros. Nous envisageons d'évaluer l'approche de SUIDT sur divers domaines d'application tels que des processus de contrôle ou des applications de base de données.

Les modèles de tâches de SUIDT et CTTE sont basés sur la sémantique de CTT. Nous avons montré que SUIDT ne modifiait pas la sémantique de CTT, et même si notre approche à deux modèles de tâches (modèle abstrait et modèle concret) ne correspond pas exactement à celui utilisé par l'outil CTTE, il est envisageable d'autoriser un échange entre ces deux outils. Cet échange pourrait consister tout d'abord à éditer le modèle de tâche sous un outil et, par la suite, à charger ce modèle dans l'autre outil. Il sera intéressant d'étudier les informations qui ne pourront pas être prises en compte pendant l'échange car SUIDT est un environnement de développement alors que CTTE est avant tout un outil d'édition et de validation. Des travaux similaires ont déjà été abordés dans [18] et [19].

Enfin un dernier point intéressant est l'exploitation du modèle de tâche hormis l'utilisation pour la validation et le fonctionnement de l'application. Deux directions peuvent être explorées : la génération du contrôleur de dialogue qui autorisera alors la construction automatique d'une application finale, et la génération du manuel utilisateur.

BIBLIOGRAPHIE

1. Puerta, A. and Eisenstein, J. Interactively Mapping Task Model to Interfaces in Mobi-D, in *Proc. Eurographics Workshop on Design, Specification and Validation of Interactive Systems (DSV-IS'98)* (Abingdon, UK, 3-5 June, 1998), Proceedings, pp. 261-274.
2. Baron, M. and Girard, P. Bringing Robustness to End-User Programming, in *Proc. 2001 IEEE Symposia on Human-Centric Computing Languages and Environments* (Stresa, Italy, September 5-7 2001, 2001), Entergraphica, pp. 142-149.
3. Bjorner, D. VDM a Formal Method at Work, in *Proc. VDM Europe Symposium'87* 1987), Springer-Verlag, pp. .
4. Spivey, J.M. *The Z notation: A Reference Manual*. Prentice Hall Int., 1988.
5. Abrial, J.-R. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996, 779 p.
6. ClearSy. Atelier B - version 3.5 in 1997.
7. Baron, M. and Girard, P. Vers un développement sûr d'applications interactives, in *Proc. IHM-HCI 2001* (Lille, France, 10-14 septembre 2001, 2001), Cépaduès-Éditions, 2, pp. 155-158.
8. Fekete, J.-D. Les trois services du noyau sémantique indispensables à l'IHM, in *Proc. Journées Francophones sur l'Ingénierie de l'Interaction Homme-Machine (IHM'96)* (Grenoble, 16-18 septembre, 1996), Cépaduès, pp. 45-50.
9. Lieberman, H. *Your Wish is my command*. Morgan Kaufmann, 2001, 416 p.
10. Scapin, D. and Bastien, J.-M.C. Analyse des tâches et aide ergonomique à la conception : l'approche MAD* (chapitre 3) in *Analyse et conception de l'I.H.M. / Interaction Homme-Machine pour les S.I. vol.1, edited by C. Kolski.* Hermès Science, 2001. Vol. 1,
11. John, B.E. and Kieras, D.E. The GOMS Family of User Interface Analysis Techniques: Comparaison and Contrast. *ACM Transactions on Computer-Human Interaction*. 3, 4 (December 1996), pp. 320-351.
12. Hix, D. and Hartson, H.R. *Developping user interfaces: Ensuring usability through product & process*. John Wiley & Sons, inc., Newyork, USA, 1993.
13. Scapin, D.L. and Pierret-Golbreich, C. MAD : Une méthode analytique de description des tâches, in *Proc. Colloque sur l'Ingénierie des Interfaces Homme-Machine (IHM'89)* (Sophia-Antipolis, France, Mai, 1989), pp. 131-148.
14. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2001, 208 p.
15. Biere, M., Bomsdorf, B. and Szwillus, G. The Visual Task Model Builder, in *Proc. Third Conference on Computer-Aided Design of User Interfaces (CADUI'99)* (Louvain-la-neuve, Belgique, 21-23 October, 1999), Kluwer Academic Publishers, pp. 245-256.
16. Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. and E. Salcher. Declarative interface models for user interface construction tools : the MASTERMIND approach, in *Proc. IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)* (Grand Targhee Resort (Yellowstone Park), USA, 14-18 August, 1995), Chapman & Hall, pp. 120-150.
17. Ousmane, S. *Spécification comportementale de composants CORBA*. PhD Université de Toulouse 1, Toulouse, 2001, 201 p.
18. Navarre, D., Palanque, P., Paterno, F., Santoro, C. and Bastide, R. A Tool Suite for Integrating Task and System Models through Scenarios in *Interactive Systems Design, Specification, and Verification (DSV-IS'01), edited by C. Johnson.* Springer-Verlag, 2001. pp. 88-113.
19. Navarre, D. *Contribution à l'ingénierie en Interaction Homme-Machine*. PhD Université Toulouse 3, Toulouse, 2001, 219 p.