## Titre

Tester la conformité d'une interface homme-machine à son modèle de tâches.

## Auteurs

Loé SANOU, Patrick GIRARD, Laurent GUITTET, Sybille CAFFIAU

Laboratoire d'Informatique Scientifique et Industrielle (LISI / ENSMA)
Téléport 2-1, avenue Clément Ader - BP 40109, 86961 Futuroscope, France
{sanou, girard, guittet, caffiaus} @ensma.fr
Téléphone : (+33/0) 5 49 49 80 70 fax : (+33/0) 5 49 49 80 64

## Resumé

Cet article présente une approche permettant de vérifier la conformité d'une application interactive avec le modèle de tâches correspondant. La technique consiste à relier l'IHM au modèle de tâches en insérant du code spécifique dans le code de l'IHM. Grâce à ce code, basé sur l'enregistrement des actions de l'utilisateur sur l'IHM et leur traduction en noms de tâches, il est possible de produire un document. Ce document contenant la séquence des actions faites sur l'IHM est utilisé comme scénario et rejoué dans K-MADe pour vérifier la conformité ou non de l'IHM vis-à-vis du modèle de tâches associé.

## Mots clés :

Interaction Homme-Machine, Interface Homme Machine, Test d'interface, Modèle de tâches, Programmation sur Exemple, Boîte à outils.

## Nom et version du logiciel utilisé :

Microsoft Word 2004 pour Mac Version 11.0 (040910)

## Forme de participation :

Articles longs

## Thèmes de la soumission :

Ingénierie de l'IHM, outils de conception et réalisation : programmation sur exemple, évaluation de l'IHM, analyse et modèle de tâches

# Tester la conformité d'une interface homme-machine à son modèle de tâches

*Loé Sanou – Patrick Girard – Laurent Guittet – Sybille Caffiau*

Laboratoire d'Informatique Scientifique et Industrielle (LISI / ENSMA)
Téléport 2 – 1, avenue Clément Ader BP 40109
86961, Futuroscope Chasseneuil, France
{sanou, girard, guittet, caffiaus}@ensma.fr

**RESUME**
Cet article présente une approche permettant de vérifier la conformité d'une application interactive avec le modèle de tâches correspondant. La technique consiste à relier l'IHM au modèle de tâches en insérant du code spécifique dans le code de l'IHM. Grâce à ce code, basé sur l'enregistrement des actions de l'utilisateur sur l'IHM et leur traduction en noms de tâches, il est possible de produire un document. Ce document contenant la séquence des actions faites sur l'IHM est utilisé comme scénario et rejoué dans K-MADe pour vérifier la conformité ou non de l'IHM vis-à-vis du modèle de tâches associé.

**MOTS CLES :** Interaction Homme-Machine, Interface Homme Machine, Test d'interface, Modèle de tâches, Programmation sur Exemple, Boîte à outils.

**ABSTRACT**
This article presents an approach to verify the conformity of an interactive application developed with the task model would work. The technique involves linking a HCI to a task model by inserting specific code in the HCI code. With this code, based on a toolkit allowing the registration of user actions on the HCI and their translation into tasks names, it is possible to produce a document. This document containing the sequence of actions made on the HCI is used as a script and replayed in K-MADe to verify compliance or not of HCI regard to the task model associated.

**CATEGORIES AND SUBJECT DESCRIPTORS:** D.2.3 [Software Engineering]: Coding Tools and Techniques; D.2.5 [Testing and Debugging] : Testing tools; H.5.2 [User Interface] : Evaluation/Methodology

**GENERAL TERMS:** Experimentation, Verification,

---



**KEYWORDS:** Human Computer Interaction, Human Computer Interface, Interface testing, Task model, Programming by Demonstration, Toolkit.

## INTRODUCTION

La validation et la vérification des systèmes informatiques constituent un aspect important du cycle de développement logiciel. La validation des systèmes a connu de grandes avancées par l'utilisation de méthodes formelles, telles que les méthodes à base de preuves (méthode B par exemple) ou les méthodes appliquant la théorie du model-checking [1]. Malgré la réussite certaine de ces approches, ces techniques n'ont pas rendu obsolète la notion de test. En l'absence d'un cycle complet de développement formel, le test demeure la seule solution permettant de s'assurer que le système effectivement réalisé, éventuellement validé par les modèles développés pour sa conception, effectue bien le travail pour lequel il a été conçu. De nombreux travaux ont ainsi été menés sur la notion de test, comme en témoigne l'abondante littérature sur la question publiée dans les revues et conférences de génie logiciel. De nombreuses méthodes, et encore plus d'outils, adressent le problème de l'automatisation du test, afin de permettre l'instrumentation des tests de non-régression, censés permettre d'éviter une régression des fonctionnalités du logiciel au fil de ses évolutions. Le test a pris une importance primordiale dans les méthodes prônant « l'eXtreme Programming » [2], ce qui a donné lieu à des outils particulièrement efficaces comme ceux de la série xUNIT [3]. Cependant, les outils proposés actuellement s'intéressent principalement, voire exclusivement, à l'aspect strictement logiciel du développement, en se concentrant sur les fonctions calculables.

La validation des systèmes interactifs est loin de ne concerner que ce strict aspect logiciel. La fiabilité du calcul réalisé ne saurait en aucun cas être une mesure suffisante de la qualité du système. Les spécificités de la participation humaine ont été prises en compte dans des travaux qui ont conduit à l'établissement de critères ergonomiques ([4], [5], [6]), susceptibles de représenter un aspect non fonctionnel important des systèmes interactifs. Plus généralement, la définition de la notion de conception centrée utilisateur a permis d'intégrer l'activité humaine au centre du processus de conception des ap-

plications, dans le but d'obtenir une meilleure utilisabilité. Ceci a conduit la communauté de l'interaction homme-machine (IHM) à définir des modèles spécifiques, dont les plus importants sont certainement les modèles de tâches, qui s'appuient sur la théorie de l'action de Norman [7] pour modéliser l'activité de l'utilisateur.

Tout naturellement, de nombreux travaux ont cherché à transposer les notions de validation et de vérification dans le domaine de l'interaction homme-machine. Comme le prouve le nom même de la principale conférence dédiée à ce problème, DSVIS pour Design, Specification and Validation of Interactive Systems, le domaine de la validation a été le plus intensément exploré, les différents auteurs cherchant à adapter les méthodes aux spécifités de l'IHM. Comme pour les travaux en génie logiciel, les méthodes à base de preuve et les méthodes à base de model-checking ont été largement utilisées, parfois de façon complémentaire [8]. L'approche des systèmes à base de modèles (Model-Based Systems) très en vogue aujourd'hui avec le développement de l'ingénierie dirigée par les modèles (Model-Driven Enginenering, [9]), intègre une bonne partie de ces résultats, et a aboutit à des outils comme PetShop [10] ou TERESA [11] qui permettent respectivement d'animer un système interactif dans le respect de son modèle sous-jacent, lui-même basé sur des réseaux de Petri, et de générer une application interactive à partir de son modèle (de tâches en l'occurrence).

Cependant, comme pour le développement logiciel à dominante fonctionnelle, l'utilisation de ces méthodes ne saurait éviter le besoin de recourir à des méthodes de test. Mais en la matière, très peu de travaux ont été consacrés à la définition de méthodes ou d'outils de tests d'IHM. Certes, une catégorie d'outils, à base d'espionnage de l'interaction entre l'utilisateur et le système, a vu le jour, tels que [12]. Ces outils permettent de traduire une série d'interactions de l'utilisateur en logs plus ou moins évolués, susceptibles d'êtres ré-exécutés sur le système. En dehors du fait que ces outils sont très sensibles à la partie lexicale de l'interaction homme-machine (tout changement d'un widget de l'interface entraîne l'obsolescence du test), le principal reproche que l'on puisse faire à ces méthodes est le très faible niveau sémantique des données qu'ils manipulent. A. Memon et son équipe [13] ont développé de nombreux travaux spécifiquement dédiés au test des interfaces utilisateurs graphiques. Fortement inscrits dans les travaux autour des tests de non-régression, ils proposent des méthodes statistiques pour générer des jeux de tests présentant des taux de couverture importants. Cependant, cette approche est essentiellement centrée sur le système ; elle se base sur une représentation des fonctionnalités du système, et ne prend pas en compte l'activité de l'utilisateur. Elle s'avère donc incapable de vérifier que le système répond à ses spécifications utilisateur. Les seules approches qui envisagent un lien direct entre l'application et l'activité sont à rechercher dans les travaux de Bourquin-Tarby chercher dans les travaux de Bourquin-Tarby [14] et Balbo [15], que nous explorerons plus en détail dans la suite de ce papier.

Dans le présent travail, nous explorons les liens possibles entre un modèle de tâches et l'application finale censée le représenter. Tournant le dos aux approches dirigées par les modèles où l'application serait dérivée des modèles, et donc en partie du modèle de tâches, nous considérons le cas où le développeur d'application souhaite donner la possibilité de vérifier que l'exécution de tests sur l'interface est bien conforme au modèle de tâches prescrit. Pour cela, nous proposons une méthode d'instrumentation du code développé, qui s'avère très légère pour le développeur. Couplée à un outil de simulation de modèles de taches, K-MADe, cette méthode permet facilement de construire des séries de tests qui peuvent être confrontés directement au modèle de tâches.

Dans cet article, nous présentons dans un premier temps, l'interface utilisateur et le modèle de tâches en soulignant le rôle du dernier dans la construction de l'interface utilisateur. Dans une deuxième partie, nous montrons les liens que l'on peut définir entre les actions élémentaires de l'interface utilisateur et les tâches élémentaires du modèle de tâches. Enfin, la troisième partie présente la technique d'implémentation par la génération de scénarii à partir de l'interface utilisateur en exécution. Nous terminons par une analyse critique de la solution proposée ainsi que les perspectives de notre travail.

## INTERFACE UTILISATEUR ET MODELE DE TACHES

Les modèles de tâches ont été conçus dans le but de modéliser l'activité humaine. Cependant, de nombreux auteurs ont cherché à utiliser ces modèles dans la construction des interfaces, et ces derniers ont ainsi largement évolué vers la prise en compte de spécificités d'interaction.

### Interface utilisateur d'une application

La plupart des systèmes informatiques possèdent une interface homme machine encore appelée interface utilisateur. Ces interfaces constituent en général le seul point d'entrée de l'utilisateur dans ces systèmes [16]. Elles ont pour but de donner accès aux fonctionnalités du système à l'utilisateur. À ce titre, elles doivent satisfaire à des critères de qualité spécifiques, découlant de la notion d'utilisabilité. L'un des points les plus importants est la nécessité d'une parfaite adéquation entre l'interface utilisateur et les besoins de l'utilisateur [17] [18]. Un deuxième facteur nous semble important à prendre en compte : au cours des vingt dernières années, les interfaces utilisateurs sont devenues de plus en plus complexes [19]. L'apparition de nouvelles techniques d'interaction, qualifiées généralement du vocable post-WIMP, ont remis sur le devant de la scène le rôle important du développeur, et plus particulièrement du spécialiste d'interface.

La conséquence principale de ceci est que les outils développés autour des concepts de génération d'interfaces, qui avaient éventuellement permis de prendre en compte des aspects liés à l'utilisabilité, deviennent inadaptés dès que l'on cherche à utiliser les nouvelles techniques d'interaction.

### Modèle de tâches

Pour exprimer les besoins des utilisateurs, souvent non-informaticiens, de nombreuses notations de description ont été proposés dans le domaine des IHM. Ces notations, couramment appelées modèles de tâches sont, pour la plupart, centrées utilisateurs et sont souvent éloignées des implantations informatiques. Les modèles de tâche possèdent des qualités et des finalités très différentes. Différentes classifications ont été proposées pour résoudre ce problème [20] [21] [22] [23]. Un effort salutaire de simplification a été proposé au travers de leur regroupement en deux catégories dans [24] : les modèles d'analyse de tâches et les modèles de description de l'interface homme machine.

L'analyse de tâches consiste à collecter des informations sur la façon dont les utilisateurs accomplissent une activité. L'acquisition de ces informations est obtenue par les récits des utilisateurs au moyen de lectures de rapports, d'interviews, de vidéo ou de simulations [5]. Mais cette analyse doit être indépendant de toute idée d'interface à réaliser et ne doit pas comporter de sous-entendus sur les dispositifs d'interaction. Elle doit donc rester à un haut niveau d'abstraction. Ces modèles liés à l'analyse de tâches servent essentiellement de notations support pour la collecte et l'analyse des informations. En l'espèce, ils sont rarement formalisés. Rentrent dans cette catégorie les modèles HTA (Hierarchical Task Analysis) [25] et MAD (Méthode Analytique de Description) [6] par exemple.

Les modèles de description de l'IHM définissent la vue que l'utilisateur aura du système interactif. Ces modèles s'inscrivent dans une logique de continuité par rapport à celle de l'analyse de la tâche. En plus, ils intègrent le retour d'information en provenance de l'interface. Ils améliorent la communication entre l'ergonome et le concepteur de logiciel. On peut citer ici des modèles comme UAN [26] et CTT [27]. Dans cette catégorie le dernier né, largement inspiré de MAD, est K-MAD (Kernel of Model for Activity Description) [28], que nous avons utilisé. Ce choix a été motivé par la présence de riches opérateurs qui permettent de couvrir le cadre de description d'une application interactive, et la disponibilité d'un outil, K-MADe[1] qui permet d'éditer et d'effectuer des vérifications sur les modèles K-MAD [29]. K-MADe, à l'instar de CTTE [27], permet d'éditer les modèles en respectant les contraintes du formalisme, de construire des scénarii et de simuler l'exécution de ces modèles [29] [28]. Il va plus loin que CTTE en permettant de prendre en compte les valeurs des objets concernés par les conditions définies dans le modèle. De plus, une interface d'échange, basée sur une description XML, permet d'utiliser d'autres modèles au format K-MAD, ou des scénarii, ce qui est essentiel pour nos besoins.

### Modèle de tâches et validation des IHM

Au vu de ce qui précède, il semble clair que les modèles de tâches peuvent constituer la base de départ de réalisation d'une IHM. D'après [30], le modèle de tâches issu du domaine de l'IHM est utile pour deux raisons principales. D'une part, il peut servir d'outil dans la conception et d'autre part, il s'adresse à l'utilisateur pour l'aider dans son travail via l'utilisation d'une aide sous forme de documentation, définie dans la phase de conception. Nous ajouterons à ces deux points que le modèle de tâches est un bon candidat pour servir de support à la validation du système. De par sa nature, il permet de vérifier les propriétés des tâches.

Cependant, l'utilisation des modèles dans le processus de développement n'est pas aisée. En effet, la modélisation de la tâche suit un processus informel ; elle aboutit souvent à des représentations conceptuelles erronées ou ambiguës, pour cause de manque d'outils pour vérifier la cohérence de la modélisation. Après le développement de ces représentations, comment peut-on s'assurer de la présence de toutes les tâches et de leur ordonnancement ? Comment s'assurer que l'interface utilisateur respecte le modèle de tâches ? La majorité des travaux développés jusqu'ici s'appuient sur la vérification de propriétés sur des modèles de l'application. Dans [31] et [32] la génération de l'interface a été prouvée, et cela de façon théorique avec la mise en pratique d'un test en laboratoire, mais malheureusement, la conformité de l'application aux modèles censés la représenter ne peut être prouvée. La solution que nous présentons suit une démarche différente. Elle s'appuie sur la définition d'un lien direct entre l'interface et le modèle de tâche, qui permet, par la définition de scénarii directement sur l'application finale, de vérifier que cette exécution reste conforme aux modèles. Ce travail est complémentaire à [33] et [34] dans la mesure où nous complèterons l'interface conçue afin d'assurer la vérification de la conformité. Par contre notre travail est du même aspect que [14] à la différence des principes et outils utilisés. Nous reviendrons sur cette comparaison dans l'analyse de notre solution.

## LIENS ENTRE MODELE DE TACHES ET INTERFACE UTILISATEUR

De façon intuitive, il apparaît que les modèles de tâches permettent d'exprimer une partie de l'interaction. Nous allons voir dans cette section comment établir formellement des liens entre modèles de tâches et interfaces.

[1] *http://www-rocp.inria.fr/merlin/kmade/*

**Tâches élémentaires du modèle de tâches**

K-MAD est un modèle hiérarchique. Il représente l'activité de l'utilisateur sous forme d'arbre de tâches, du plus général (tâche mère) au plus détaillé (actions élémentaires), en passant par des tâches intermédiaires [28]. La *figure 1* présente le modèle de tâche du « Convertisseur Franc/Euro » qui constitue notre étude de cas. Sur la *figure 1*, *Conversion_Franc_Euro* modélise la tâche liée à l'application du convertisseur. Elle décrit le fait que l'utilisateur doit choisir la valeur à convertir (*Valeur_Conversion*), puis saisir cette valeur (*Saisir_Valeur*), et enfin effectue la conversion soit par *Conversion_Franc* soit par *Conversion_Euro*. Une fois le choix de la conversion effectué, la valeur convertie est affichée (tâche *Valeur_Convertie*). A tout moment du déroulement de la tâche *Conversion_Franc_Euro* ou *Saisir_Valeur*, l'utilisateur peut la désactiver par la tâche *Quitter*.

Une tâche est composée d'un ensemble de caractéristiques représentant son pouvoir d'expression. Ses caractéristiques se divisent en trois catégories :

- les caractéristiques générales (numéro, nom…) ;
- les caractéristiques liées à l'ordonnancement (Décomposition, nécessité, interruption, …) ;
- les caractéristiques liées aux traitements des actions (événement, garde, post condition).

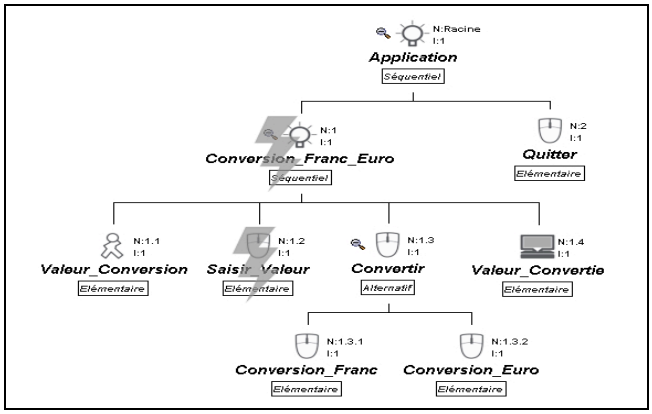


***Figure 1 :*** Modèle de tâches K-MADe du Convertisseur.

Dans l'environnement K-MADe, l'ordre des sous-tâches d'une même tâche-mère (priorité des tâches ou contrainte de précédence) est défini par l'ordre gauche-droit sur l'espace de tâches. L'une des caractéristiques générales d'une tâche est son exécutant. Il peut être :

- « Utilisateur » : tâche réalisée par l'utilisateur autre qu'une action sur le système ; exemple : tâche *Valeur_Conversion* (*Figure 1)* ;
- « Système » : tâche réalisée par le système ; exemple : tâche *Valeur_Convertie* (*Figure 1*) ;
- « Interactif » : tâche déclenchée par l'utilisateur et réalisée conjointement sur le système ; exemple : tâche *Conversion_Franc* (*Figure1*) ;
- « Abstrait » : soit la tâche n'est pas complètement décrite, soit la tâche contient des sous-tâches d'exécutants différents ; exemple : tâche *Conversion_Franc_Euro* (*Figure 1*) ;

Une tâche élémentaire correspond à une action élémentaire non décomposable ou non décomposée du modèle de tâches. Elle ne peut être de type « Abstrait », donc de type « Utilisateur », « Interactif », ou « Système ». Les tâches élémentaires sont des tâches feuilles du modèle de tâches. En suivant la décomposition hiérarchique des tâches, seules les tâches élémentaires apparaissent susceptibles de concerner une action sur l'IHM. Du modèle de tâche de notre exemple, représenté par la *figure 1*, on déduit que les tâches élémentaires sont : *Saisir_Valeur*, *Conversion_Franc*, *Conversion_Euro*, *Valeur_Convertie* et *Quitter*.

**Actions élémentaires de l'interface utilisateur**

L'IHM est un médiateur. C'est à travers elle que l'utilisateur exerce les différentes actions lui permettant d'exécuter sa tâche. Du point de vue informatique, l'interface utilisateur reflète exactement et fidèlement le comportement de l'application [17]. Les actions élémentaires sur l'interface sont celles exercées par l'utilisateur lors de ses actions, et celles programmées en retour pour refléter l'état de l'application.

Les actions élémentaires de notre interface (figure 2) se résument à la saisie d'une valeur dans la zone de champs « *Valeur à convertir* » et au clic sur l'un des deux boutons « *Convertir en Euros* », ou « *Convertir en Franc* ». Tout autre action élémentaire issue d'une interaction quelconque de l'utilisateur ne sera pas considérée et par conséquent n'entraînera aucun changement de l'état du système.

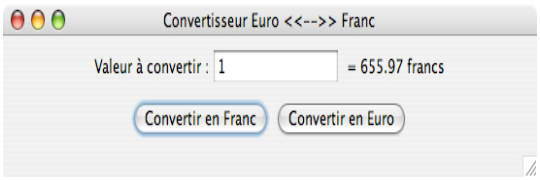


***Figure 2 :*** Interface du Convertisseur Franc/Euro.

**Liaisons entre tâches élémentaires du modèle de tâches et actions sur l'interface utilisateur**

Partant de l'idée intuitive d'une équivalence potentielle entre les feuilles d'un arbre de tâches raffiné jusqu'aux actions du système et certains éléments de l'interface, nous allons chercher maintenant à établir plus précisément le lien qui peut être établi entre eux.

Les tâches de type « Abstrait » sont des éléments structurants du modèle de tâches. Elles sont essentielles à la définition de la dynamique du modèle (et donc à la réalisation de la simulation) car elles portent la plupart des opérateurs et attributs nécessaires pour définir cette dynamique (opérateur de décomposition, multiplicité, interrup-

tion, etc.). Malgré cela, elles ne rentrent pas en ligne de compte dans l'écriture de scénarii. En effet, ces derniers ne sont constitués que d'enchaînement de tâches « feuilles » du modèle, qui ne peuvent être de type « Abstrait », comme nous l'avons dit plus haut.

Les tâches de type « Utilisateur » et de type « Système » ne donnent pas lieu à une interaction de l'utilisateur sur l'interface. Les premières ne sont liées au système que par le fait qu'elles requièrent, de la part du système, la mise à disposition des informations nécessaires à leur accomplissement. Ce sont en général des tâches de réflexion. Un lien entre ce type de tâches et l'interface ne pourrait se faire qu'à travers des états de l'interface. Les tâches de type « Système » correspondent à une action de l'application, qui engendre normalement (principe d'observabilité) un retour d'information vers l'utilisateur. Ce retour d'information, qui constitue une transition entre deux états de l'interface, est un bon candidat pour établir un lien avec la tâche.

Les tâches de type « Interactif » sont les candidats naturels au lien avec l'interface. Elles correspondent à une interaction entre l'utilisateur et le système, donc obligatoirement à au moins une action du premier sur le deuxième.

D'une manière pratique, comment s'établit cette correspondance entre tâches élémentaires et transitions de l'interface ? Dans notre étude de cas, la tâche « *Saisie_Valeur* » correspond sur l'interface à la saisie d'une valeur numérique dans le champs de texte prévu à cet effet. Les deux tâches « *Conversion_...* » correspondent à l'appui sur l'un des boutons représentés sur l'interface. Enfin, la tâche système est représentée par l'affichage de la valeur convertie dans le label prévu. On établit ainsi une table de correspondance très simple (Tableau 1).

| *Actions élémentaires de l'interface utilisateur* | *Tâches élémentaires du modèle de tâches* |
|---|---|
| Une saisie dans « Valeur à convertir » | « Saisir_Valeur » |
| Un clic sur le bouton « Convertir en Franc » | « Conversion_Franc » |
| Un clic sur le bouton « Convertir en Euro » | « Conversion_Euro » |
| Mise à jour du label d'affichage de la valeur convertie | « Valeur_Convertie » |

***Tableau 1 :*** Correspondance entre actions élémentaires de l'interface et tâches du modèle de tâches.

Notons que la relation qui s'établit entre le modèle de tâches et l'application n'est en aucun cas une bijection. Rien n'empêche de prévoir la réalisation de la tâche par deux dispositifs d'interaction. Ce serait par exemple le cas si l'on décidait de doubler les boutons servant à la conversion par des items de menu. De même, rien n'interdit d'utiliser la même interaction pour des rôles différents en fonction du contexte. La conséquence de ce point est que le lien est fonction de ce contexte. Nous discuterons ce point dans la suite. Maintenant, comment nous pouvons réellement tester une application à partir de son lien avec le modèle de tâche ?

## GENERATION DE SCENARII A PARTIR DE L'INTERFACE EN EXECUTION

Un scénario peut être défini comme une utilisation particulière du système dans un contexte précis. Le modèle de tâches, lui au contraire est une description globale. Du point de vue de ce dernier, un scénario est en fait une succession de tâches élémentaires sans structuration autre que la séquence. Vérifier si le système est conforme au modèle de tâches consiste à s'assurer que la succession des tâches respecte les conditions posées par les opérateurs du modèle de tâches [14, 33]. Cette vérification peut se faire en utilisant un outil existant dans un outil comme K-MADe (ce serait aussi le cas avec CTTe utilisé dans [14] pour les mêmes objectifs) : l'outil de simulation. Si les scénarii issus de l'utilisation du système peuvent êtres exécutés avec succès par le simulateur de K-MADe, cela signifie qu'ils sont conformes au modèle de tâches.

### Définition des liens

Notre démarche s'appuie sur l'approche présentée dans [35] où nous proposons la définition d'une boîte à outils permettant simplement d'enregistrer et rejouer les actions de l'utilisateur. L'idée générale consiste à spécialiser une boîte à outils standard, en l'occurrence Java Swing, pour lui ajouter des fonctionnalités d'enregistrement/rejeu (base de la programmation sur exemple) à moindre coût pour le développeur d'application. Ainsi, il s'agit ici de permettre au développeur d'établir directement et le plus simplement possible le lien entre son code (donc l'IHM réelle de l'application finale) et le modèle de tâches.

### Implémentation du lien pour les tâches interactives

Le mécanisme de base permettant de programmer une boîte à outils événementielle telle que Swing est celui des événements. Afin de programmer la réaction du système aux actions de l'utilisateur, le programmeur doit "abonner" un traitement à chaque objet interactif susceptible d'être actionné par l'utilisateur. Ce mécanisme d'abonnement est le candidat idéal pour l'établissement du lien entre l'application et le modèle de tâche. Cependant, il ne peut concerner que la catégorie de tâches d'interactions du modèle de tâches, qui correspondent effectivement à une action de l'utilisateur sur le système. Dans notre exemple, les tâches concernées sont les tâches de déclenchement de la conversion (*Conversion_Franc* et *Conversion_Euro*) et la tâche de saisie de la valeur.

Les deux premières tâches sont équivalentes et correspondent, d'un point de vue purement Swing, au traitement de l'événement ActionEvent, base du comporte-

ment d'un **JButton**[2]. Créer un lien entre code et modèle de tâches consiste dans le cas présent à associer le déclenchement de l'événement **ActionEvent** sur le bon bouton avec l'identification de la tâche élémentaire correspondante. Pour cela, nous avons spécialisé les widgets Swing utilisés en surchargeant leur(s) méthode(s) d'abonnement. Dans le cas du **JButton**, il s'agit d'ajouter un paramètre (chaîne de caractères) permettant de représenter l'identifiant de la tâche correspondante dans le modèle de tâche. La classe **JButton** est donc étendue en une classe **TTJButton** (le préfixe TT signifiant TaskTestable) comme suit :

```
public class TTJButton extends JButton {
      ...
      public void addActionListener
            (ActionListener action,
             String taskName) {...}
}
```

Du point de vue du développeur qui code son application, le seul travail qu'il a à faire est de préciser lors de l'association du listener la tâche élémentaire, ce qui donne l'appel :

```
boutonFE=new TTJButton ("Convertir en Franc");
boutonFE.addActionListener(monActionListener,
                  "Conversion_Franc");
```

Cette solution n'interdit pas à l'utilisateur d'utiliser les autres méthodes d'abonnement telles que les classes anonymes, ou l'auto-abonnement, puisque le Listener en lui-même n'est pas concerné par la modification.

Notons que la solution que nous proposons ici s'apparente à celle proposée par [28], qui utilise la technique de la programmation par aspects. Cependant, cette dernière technique nécessite une étape supplémentaire de programmation, susceptible de générer des erreurs par omission.

Dans le cas de la tâche de saisie, l'étude du widget classique **JTextString** montre qu'il existe plusieurs façons de gérer les événements. Le fonctionnement de base consiste à gérer la saisie par l'intermédiaire de l'événement **ActionEvent**, déclenché par l'appui sur la touche return du clavier. Cependant, un programmeur avisé en Swing évitera le plus souvent la programmation de ce seul événement, car dans ce cas, l'application ne sera pas capable de fournir un feedback approprié à la saisie tant que l'utilisateur n'aura pas frappé cette touche. Dans notre exemple, il serait souhaitable de programmer un feedback proactif sur les boutons, qui interdise à l'utilisateur de déclencher une conversion (désactivation des boutons) lorsque le champ de saisie n'est pas convertible, et qui active ces mêmes boutons lorsqu'il l'est. Dans ce cas, l'utilisateur devra programmer des événements différents, qui sont en fait accessibles par la hiérarchie d'héritage dans le **JTextString**. Une solution plus complète consistera à utiliser l'architecture MVC des composants Swing pour programmer la réaction du composant au niveau du Model, ce qui garantit une réaction à tout changement de la donnée contenue dans le **JTextString**. Ces différentes considérations expliquent pourquoi, pour permettre au programmeur de demeurer libre de ses choix, il nous a fallu surcharger tous les listeners disponibles au niveau de chaque composant, y compris ceux disponibles à travers l'héritage.

#### Implémentation du lien pour les tâches système

Les tâches de la catégorie Système sont d'une toute autre nature que les tâches d'interaction. Comme nous l'avons vu précédemment, elles se caractérisent principalement par un retour d'information (feedback) vers l'utilisateur. Le mécanisme utilisé pour les tâches d'interaction n'est donc pas utilisable. L'idée ici encore est de fournir une solution qui soit la plus légère d'utilisation pour le programmeur, afin de réduire les risques d'oubli ou d'erreur.

Plusieurs solutions sont envisageables. Une première solution consisterait à considérer un composant comme responsable du feedback d'une tâche système au maximum. Il suffit alors de surcharger tous les composants avec une fonction permettant d'enregistrer, pour un widget donné, le nom de la tâche système associée. L'inconvénient de cette méthode est qu'elle restreint un composant à un seul usage, ce qui peut sembler contraignant à l'usage. Une autre solution consisterait à surcharger tous les modifieurs des composants de la boîte à outil, afin de leur adjoindre la chaîne de caractères correspondant à la tâche invoquée. Le mécanisme est dans ce cas analogue à celui utilisé pour les tâches interaction. Le problème majeur de cette solution est la lourdeur du développement de la boîte à outils. Alors que les événements, et donc les listeners différents, sont en nombre relativement limités, les fonctions d'accès aux composants, eu égard à la relation d'héritage, sont particulièrement nombreuses.

La solution que nous avons privilégiée, toujours dans le but de minimiser l'intervention du développeur, mais également de proposer une solution acceptable du point de vue de l'implémentation, consiste à fournir une primitive statique permettant au programmeur de spécifier explicitement dans son code qu'il active une fonction système. Dans notre exemple, cela donne :

```
TaskTestToolkit.execSystemTask(
                        "Valeur_Convertie");
```

La responsabilité de la définition du lien reste au programmeur, qui peut très bien oublier de la déclarer. Mais la simplicité et l'universalité de la solution compensent largement cette limite.

#### Génération de scénarii et test de l'application

Nous avons déjà écrit que le scénario est en fait un enchaînement de tâches élémentaires. Dans K-MADe, la

[2] *http://java.sun.com/j2se/1.5.0/docs/api/*

manipulation de scénarii distingue deux aspects : l'enregistrement de scénarii et le rejeu de scénarii. L'enregistrement consiste à simuler un arbre de tâches en choisissant, tout au long de ces différents états, les actions à appliquer d'une part et les valeurs particulières d'autre part. Le rejeu d'un scénario consiste à parcourir un chemin particulier donné par le scénario dans le modèle de tâches K-MAD. En fonction du but recherché par le test, plusieurs aspects peuvent être vérifiés : atteignabilité de la tâche dans le modèle ; atteignabilité d'un état particulier dans les objets du modèle concret ; finalisation complète de la suite de tâches dans l'arbre de tâches ; etc.

Pour générer un scénario à partir de l'application, les noms des tâches insérées par le développeur sont accumulés dans une liste au fur et à mesure de l'exécution de l'IHM. À chaque interaction significative de l'utilisateur (i.e. instrumentée par le concepteur), le système engendre la tâche élémentaire correspondante que l'on insère dans la liste. L'ordre d'insertion dans la liste détermine l'ordre d'ordonnancement des tâches. Pendant la durée d'une exécution, l'utilisateur réalisera un certain nombre de tâches élémentaires pour atteindre son but. Le résultat de l'espionnage de cette exécution construit donc automatiquement un scénario. Cette phase est tout à fait analogue à la phase d'enregistrement d'un scénario dans l'environnement K-MADe. Seules les identités des tâches sont enregistrées. Le contexte n'est pas pris en compte ce qui fait qu'aucune contrainte n'est jointe à la tâche issue de l'IHM.

Le fichier enregistré est traduit en XML selon la DTD (Document Type Definition) défini par K-MADe. Ce fichier XML peut ensuite être rejoué dans l'espace de simulation de l'environnement K-MADe. Les valeurs des contraintes n'ayant pas été prises en compte seront celles du contexte du modèle de tâches à sa dernière simulation dans l'environnement. Le déroulement du rejeu se fait suivant l'animation du modèle de tâches et les erreurs dans le scénario sont détectées ou présentées de façon simple dans l'environnement : la simulation se termine, ou la tâche suivante n'est pas atteignable.

## BILAN ET CONCLUSION

La solution présentée dans cet article, bien que très simple dans son principe, permet de relier une application concrète avec un modèle de tâche censé la représenter. La vérification de la conformité de l'application au modèle de tâche permet de montrer de façon indubitable des erreurs de conception. Par exemple, la réalisation de la tâche de conversion suppose la saisie d'une valeur à convertir valide. Cette propriété est exprimable dans le modèle K-MAD par l'intermédiaire d'une pré condition liée à la tâche « *Convertir* », qui engendre une interdiction de celle-ci tant que l'on n'a pas commencé la tâche de saisie. Si l'application n'implémente pas cette vérification, il sera possible de créer un scénario qui sera non conforme au modèle.

Le lien en fonction du contexte n'est pas un problème majeur. Il engendre simplement une multiplication des scenarii à chaque choix dans le sens application vers modèle de tâches, alors qu'il n'a pas d'incidence dans l'autre sens. Néanmoins, on peut reprocher à cette méthode d'être plus destructive que constructive. En effet, les résultats les plus faciles à mettre en évidence sont des non-conformités au modèle de tâches : dès qu'un scénario n'est pas conforme, il arrête le simulateur. À l'inverse, il n'est possible de prouver la complétude de l'application par rapport aux tâches prescrites qu'en bâtissant un jeu de scénarii couvrant tous les chemins du modèle de tâche, ce qui est toujours très difficile en test. De plus, le diagnostic d'erreur est très pauvre : en l'état actuel du simulateur de K-MADe, qui, rappelons-le, n'a pas été conçu pour cet usage, il est difficile de comprendre où se situe l'erreur. Mais le simulateur de K-MADe affiche des messages d'erreur de non-exécution d'une tâche. Une remontée manuelle est nécessaire à l'interface pour cibler la cause. Peut-être qu'un travail d'une autre envergure doit être réalisé sur K-MADe.

De nombreuses pistes s'ouvrent à nous pour enrichir les capacités de cette approche. K-MAD permet de définir des objets servant à construire des expressions entrant en ligne de compte dans l'évaluation des conditions sur les tâches. Notre méthode d'établissement du lien entre application et modèle de tâches peut s'étendre à cet aspect. Afin d'obtenir un outil permettant d'effectuer des tests de non-régression, il faudrait être en mesure de rejouer un scénario validé sur l'application ; l'utilisation de l'approche sur exemple pour construire le scénario entrouvre de nombreuses portes. Enfin, il convient de déterminer une méthode pour obtenir d'autres résultats que la conformité aux tâches, comme la complétude des tâches ou la simulation simultanée dans l'environnement K-MADe lors de l'exécution de l'application.

## BIBLIOGRAPHIE


1. Aït-Ameur, Y., P. Girard, and F. Jambon. Using the B formal approach for incremental specification design of interactive systems. in Engineering for HCI. 1998: Kluwer Academic Publishers.
2. Wells, D. Extreme Programming : a gentle introduction. 2006 17 fevrier 2006 [cited; Available from: http://www.extremeprogramming.org/.
3. Mentor, O. Resources for Test Driven Development. 2001-2004 from: http://www.xunit.org/index.htm.
4. Cockton, G., S.W. Draper, and G.R.S. Weir, eds. People and Computers IX, Proceedings of the HCI'94 Conference. British Computer Society Conference Series. 1994, Cambridge University Press.

5. Dix, A., et al., Human-Computer Interaction. Second Edition ed. 1998: Prentice Hall. 638.
6. Scapin, D. and J-M. C. Bastien, 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, C. Kolski, Editor. 2001, Hermès Science: Paris, France.
7. Norman, D., User Centered System Design. 1986: Lawrence Erlbaum Associates.
8. Aït-Ameur, Y., P. Girard, and F. Jambon. A Uniform approach for the Specification and Design of Interactive Systems: the B method. in Eurographics Workshop on Design, Specification, and Verification of Interactive Systems. 1998. Abingdon, UK.
9. Girard, P., F. Jambon, and M. Baron. Using formal methods in safety-critical interactive system design : from architecture-based approaches to tool-based development. in HCI 2005. Las Vegas, Nevada.
10. Bastide, R. PetShop : a tool for the formal specification of CORBA systems. in Conference on object Oriented Programming Systems Languages and Applications. 2000. Minneapolis, United States: ACM.
11. Silvia Berti, F.C., Guilio Mori, Fabio Paternò, Carmen Santoro. TERESA: a transformation-based environment for designing and development multi-device interface. in Conference on Human Factors in Computing Systems - CHI'04. 2004. Vienna, Austria.
12. Spannagel, C. Jacareto. [cited 1999-2008; Available from: http://jacareto.sourceforge.net/.
13. Atif M. Memon, M.L.S. Regression testing of GUIs. in Foundations of Software Engineering. 2003. Helsinki, Finland: ACM New York.
14. Tarby, J.-C. Evaluation précoce et conception orientée évaluation. in Ergo'IA 2006. Biarritz France.
15. Sandrine Balbo, D.D., Christof Lutteroth, Gerald Weber. Appropriateness of user interfaces to tasks. in TAMODIA'05. 2005. Gdansk, Poland: ACM
16. Myers, B.A., User Interface Software Tools. ACM Transactions on Computer Human Interaction, 1995. **2**(1): pp. 64-103.
17. Coutaz, J. Interface Homme-Machine : un regard critique. in Journées d'Études AFCET : Interfaces Homme-Machine. 1992. Paris: AFCET.
18. Shneiderman, B., Designing the User Interface. 3 ed. 1998: Addison-Wesley. 639.
19. Myers, A.B. and M.B.S. Rosson. Survey on user interface programming. in Human Factors in Computing Systems (CHI'92). 1992. Monterey, USA: ACM/SIGCHI.
20. Paternò, F., Model-Based Design and Evaluation of Interactive Applications. 2001: Springer. 208.
21. Brun, P., XTL : une logique temporelle pour le spécification formelle des systèmes interactifs, in LRI. 1998, Université Paris -Sud: Orsay.
22. Navarre, D., Contribution à l'ingénierie en Interaction Homme-Machine, in LIHS. 2001, Université Toulouse 3: Toulouse. p. 219.
23. Limbourg, Q. and J. Vanderdonckt, Comparing Task Models for User Interface Design (Chapter 6), in The Handbook of Task Analysis for HCI, D. Diaper and N. Stanton, Editors. 2003, Lawrence Erlbaum A.
24. Baron, M., Vers une approche sûre du développement des Interfaces Homme-Machine (Thesis). 2003, Université de Poitiers: Poitiers. p. 255.
25. Shepherd, A., Analysis and training in information technology tasks, in Task Analysis for HCI, D. Diaper, Editor. 1989, Ellis Horwood: Chichester, USA. p. 15-55.
26. Hix, D. and H.R. Hartson, Developping user interfaces: Ensuring usability through product & process. Wiley professional computing. 1993, Newyork, USA: John Wiley & Sons, inc.
27. Paternò, F., G. Mori, and R. Galimberti. CTTE: An Environment for Analysis and Development of Task Models of Cooperative Applications. in ACM CHI 2001. Seattle: ACM Press.
28. Lucquiaud, V. Proposition d'un noyau et d'une structure pour les modèles de tâches orientés utilisateurs. in IHM 2005. Toulouse, France.
29. M. Baron, D.Scapin. K-MADe : un environnement pour le noyau du modèle de description de l'activité. in IHM. 2006. Montréal, Québec.
30. Balbo, S., Évaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation, in IMAG. 1994, Université Joseph Fourier: Grenoble. 206p.
31. Paternò, F. and C. Santoro. One model, many interfaces. in Computer-Aided Design of User Interfaces. 2002. Valenciennes, Frances: Kluwer Academics.
32. Wilson, S. and P. Jonhson. Bridging the Generation Gap : From Task to User Interface Designs. in CADUI'96. 1996. Namur, Belgium.
33. A. Lewandowski, G.B., Jean-Claude Tarby. Vers des composants logiciels orientés tâches. in IHM. 2006. Montréal, Québec.
34. Balbo, S. Projet WAUTER (Website Automatic Usability Testing EnviRonment. [cited; Available from: http://wauter.weeweb.com.au.
35. L. Sanou, P. Girard, L. Guittet. Introducing Programming by Demonstration techniques at the toolkit level : a case study. in HCI International 2005. Las Vegas, Nevada, USA.