SUIDT : Un outil de construction d'interfaces utilisateurs sûres (Article de Démonstration)

Mickael Baron, Patrick Girard

SUIDT : Un outil de construction d'interfaces utilisateurs sûres

Mickaël Baron, Patrick Girard

Laboratoire d'Informatique Scientifique et Industrielle, ENSMA,
1 rue Clément Ader, 86961 Futuroscope Chasseneuil
{baron,girard}@ensma.fr

RESUME

Dans cet article nous présentons un nouvel outil d'aide à la construction d'interfaces utilisateur qui s'appuie sur des formalismes existants. Cet outil de développement appelé SUIDT (Safe User Interface Design Tool) est destiné à des non spécialistes de la programmation. Il utilise pour cela des techniques de programmation visuelles qui aident le développeur, d'une part dans la construction des différents modèles de l'application interactive, et d'autre part dans la validation de cette application.

MOTS CLES : Modèle de tâches, validation de tâches, programmation visuelle, modèle d'architecture, Systèmes basés sur modèles (MBS).

ABSTRACT

We present in this paper a new computer-aided design for user interfaces tool that leans on several well-defined formalisms. This development tool for end-users is called SUIDT (Safe User Interface Design Tool). It uses visual programming techniques to build every application model and allows building the final application with respect to all models.

KEYWORDS : Task model, visual programming, task validation, design notation, Model Based Systems (MBS).

INTRODUCTION

Les outils qui aident les concepteurs à réaliser des applications interactives sont aujourd'hui nombreux. D'une part il y a les constructeurs d'interfaces qui mettent l'accent sur l'aspect présentation et permettent de réaliser le noyau fonctionnel au fur et à mesure de la construction de l'interface. Ces constructeurs d'interfaces ont réellement conquis l'industrie et répondent parfaitement aux exigences de maquettage, mais avec l'inconvénient de ne pas assurer la conformité de l'application au modèle de tâches et de n'être garants d'aucune architecture logicielle. D'autre part, les générateurs automatiques d'applications et systèmes basés sur modèles (MBS), qui sont issus du monde de la recherche, tirent parti des spécifications de modèles (modèle de tâches, modèle de présentation…) pour à la fois raisonner sur ces modèles et en générer une application interactive. Cependant, ces systèmes ne sont pas adaptés aux utilisateurs finaux (possédant peu d'expérience dans la programmation classique) et la nécessité d'apprendre des langages spécifiques complexifie le processus de création.

Notre approche consiste à regrouper ces deux familles d'outils. Elle s'appuie sur des modèles existants et liés dont les sémantiques sont clairement définies. De plus nous fournissons au concepteur des outils interactifs qui permettent de manipuler ces modèles avec le respect de leur sémantique. Dans ce but, nous avons développé un environnement interactif de programmation pour utilisateur final appelé SUIDT (Safe User Interface Design Tool).

SUIDT COTE TECHNIQUE

Le démarche de conception associée à l'outil SUIDT consiste à concevoir de manière interactive le dialogue entre un noyau fonctionnel existant développé formellement et une présentation graphique de l'interface construire de manière classique. Ce dialogue est basé sur une notation de description de l'utilisateur, CTT [5] et sa construction se scinde en deux phases :

Dans un premier temps, un modèle de tâches abstrait sans lien avec une interface graphique est spécifié. Il s'appuie sur des travaux amonts issus des ergonomes. Ce modèle de tâches abstrait a pour objectif de valider (par simulation) la faisabilité des tâches en liaison avec les fonctionnalités d'un noyau fonctionnel existant. Ce dernier est développé formellement au moyen de la méthode B [1] et de l'outil Atelier B[1] et respecte les services préconisés par [4]. Il dispose notamment de fonctions accesseurs, modifieurs et de pré-conditions sur la validité des appels de fonction à l'aide de prédicats de typage et de propriétés. La dynamique du modèle de tâches abstrait se base sur des contraintes de précédence (opérateurs temporels) et sur les pré-conditions des fonctions accesseurs

[1] ClearSy - http://www.atelierb.societe.com/

du noyau fonctionnel. Les post-conditions des tâches, modifiant l'état du noyau fonctionnel par les modifieurs, sont exprimées au niveau des feuilles du modèle de tâches.

Dans un second temps, des interfaces graphiques sont associées à ce modèle de tâches abstrait. Elles sont développées par l'intermédiaire d'un outil classique de construction d'interfaces. Le modèle de tâches abstrait est alors raffiné depuis un niveau fonctionnel jusqu'à la spécification des interactions de l'utilisateur et des rendus du système. Plus précisément seules les tâches feuilles du modèle de tâches abstrait sont raffinées dans le but de spécifier sous forme de tâches d'interactions les liens (en entrée et en sortie) avec les objets des interfaces graphiques. Le modèle obtenu porte alors le nom de modèle de tâches concret.

Par rapport aux modèles d'architecture existant, l'approche emploie une architecture basée sur ARCH [3]. En effet, notre approche propose d'un côté une présentation s'appuyant sur la boîte à outils Java/Swing®[2] , un contrôleur de dialogue, une interface du noyau fonctionnel, un noyau fonctionnel indépendant. De plus, le dialogue homme-machine est conçu tout en respectant les contraintes de sûreté imposées par le noyau fonctionnel, par les besoins de l'utilisateur et par la présentation.

SUIDT COTE CONCEPTEUR

L'environnement SUIDT propose plusieurs outils pour modéliser une application interactive, il se compose de :

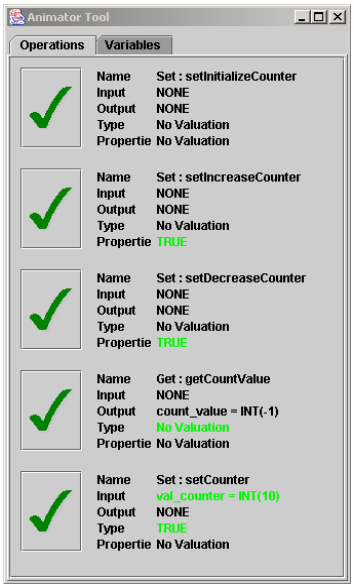

Figure 1 – Outil d'animation du noyau fonctionnel
Exemple du *compteur*

Un outil d'interfaçage avec le noyau fonctionnel appelé **outil d'animation**, *Figure 1*. Il permet au concepteur de raisonner et de tester le noyau fonctionnel en appelant des fonctions accesseurs ou modifieurs. Cette interface reflète l'état du noyau à tout moment de la phase de conception. Il n'existe pas de phase de compilation où le contexte serait perdu à chaque nouvelle exécution. Il s'agit là, d'un point important que les outils comme les constructeurs d'interfaces, ou la plupart des MBS, ne possèdent pas.

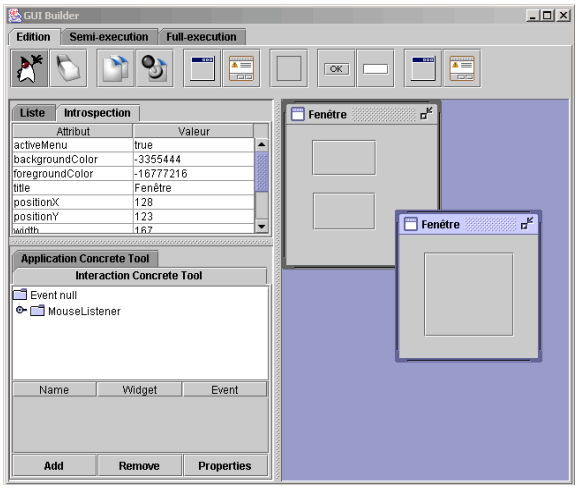

Figure 2 - Outil de construction d'interfaces

Un outil de construction d'interfaces interactives ou couramment appelé GUI-Builder, *Figure 2*. Il propose tous types de composants graphiques issus des boîtes à outils classiques (bouton, label, champs de texte…) pour que le concepteur puisse concevoir les interfaces de l'application. A la différence des GUI-Builder classiques, celui-ci permet l'exécution instantanée de l'interface puisque le concepteur manipule directement les objets de l'interface de la future application et non une simple représentation graphique qui en est faite.

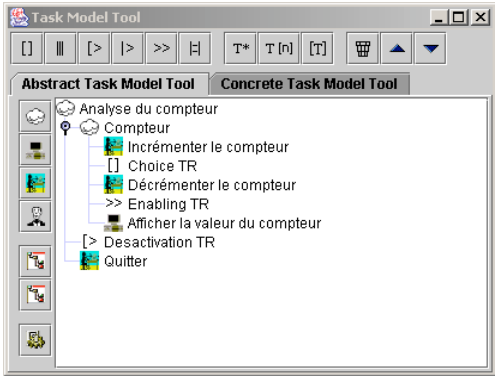

Figure 3 - Editeurs de modèle de tâches abstrait et concret

Un éditeur de modèles de tâches (abstrait et concret) basés sur CTT, *Figure 3*, permet de construire le dialogue de l'application. Le modèle de tâches abstrait est construit au moyen des quatre catégories du formalisme originel CTT (tâche abstraite, tâche interaction, tâche application et tâche utilisateur). Tandis que la construction du modèle de tâches concret s'appuie sur la descrip-

[2] Sun Microsystems – http://java.sun.com

tion concrète des tâches interactions et applications du modèle de tâches abstrait. Chaque tâche est reliée à sa précédente au moyen des opérateurs du formalisme CTT (activation, interruption, désactivation…). La vue des arbres des modèles de tâches n'est pas semblable à celle trouvée dans l'outil CTTE [5]. Le choix du widget arbre a été motivé pour des raisons de taille et de facilité.

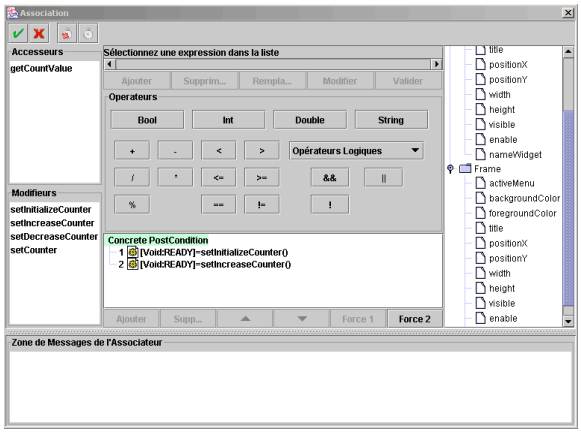

Figure 4 - Outil de modélisation des objets de la tâche

La modélisation des pré-conditions (est-ce que la tâche est activable?) et post-conditions (résultat après traitement de la tâche) des tâches est réalisée au moyen d'**un outil de modélisation des objets de la tâche** *Figure 4*. La modélisation de ces éléments est obtenue à partir des fonctions du noyau fonctionnel pour le modèle de tâches abstrait et auxquelles s'ajoutent les attributs de la présentation pour le modèle de tâches concret. Ces spécifications sont réalisées sous forme d'expressions qui peuvent être évaluées au moment de leur création (le noyau fonctionnel est toujours en exécution) pour vérifier par exemple la conformité des types.

La validation et la vérification des deux modèles de tâches sont obtenues par un **outil de simulation** à tout moment de la conception, *Figure 5*. A partir de la simulation du modèle de tâches abstrait, le concepteur anime le dialogue abstrait en déclenchant les tâches à la manière de la simulation de CTTE. Mais en plus de vérifier l'ordonnancement des tâches, le concepteur vérifie le comportement du noyau fonctionnel. En revanche pour la simulation du modèle de tâches concret, le concepteur peut tester l'application à développer, soit en utilisant l'interface graphique, soit en déclenchant les tâches. Le rendu du déroulement de la simulation est alors disponible directement sur le noyau fonctionnel mais aussi sur l'interface de l'application. Tout au long de la simulation certains éléments de la modélisation sont modifiables? la modélisation des pré- et post-conditions des tâches et les objets de la présentation, sous réserve de ne pas supprimer des composants graphiques référencés par des tâches. Enfin, l'outil simulation autorise l'enregistrement et le rejeu de scénarii des modèles de tâches abstrait et concret.

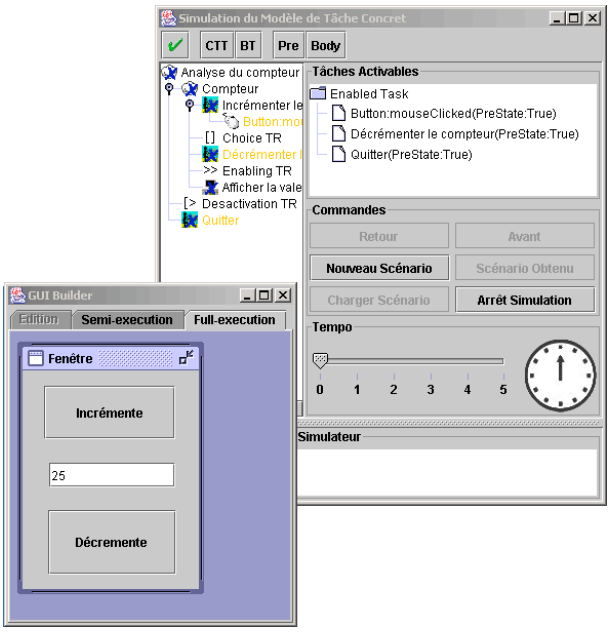

Figure 5 - Simulateur d'un modèle de tâches concret

DISCUSSION

A l'inverse des éditeurs de modèle de tâches comme CTTE, SUIDT regroupe un noyau fonctionnel, un modèle d'interface et un modèle de tâches qui permet d'associer aux tâches des références aux objets de ces modèles. Ainsi CTTE contrôle le modèle de tâches pour toutes sortes d'incohérences (boucle sans fin, tâche inateignable…) et vérifie aussi l'exactitude avec les exigences édictées dans le cahier des charges. SUIDT vérifie également les erreurs de constructions du modèle de tâches mais il assure en plus que les appels du noyau fonctionnel et de l'interface interactive sont cohérents. Dans CTTE, les tâches qui sont activées par l'utilisateur ne modifient pas les modèles de l'application. Ceci n'est pas vrai dans SUIDT, où les attributs des références des objets associées aux tâches sont modifiés.

SUIDT et MOBILE [6] qui est un système basé sur modèles ont des avantages communs. Un des principaux, est qu'ils utilisent la description du dialogue d'une application (traduite au moyen d'un modèle de tâches) comme la base pour créer de manière ascendante l'interface de l'application à l'aide d'un GUI-Builder (à la manière d'un constructeur d'interface). Ceci permet de laisser une grande liberté au concepteur pour la création de la couche de présentation en termes de choix des éléments de présentation et des choix des méthodes d'interaction (ergonomie, esthétique…). Au contraire les autres systèmes basés sur modèles s'attachent à utiliser des générateurs qui ne permettent pas un contrôle sur la génération.

Plus généralement où SUIDT apporte une réelle contributon dans les approches des MBS, se situe au niveau de l'aspect des outils d'évaluation, des outils d'implémentation et des outils de modélisation?

Sur le premier point, l'environnement SUIDT fournit plusieurs outils de validation (outil de simulation et outil d'animation) qui interviennent à tous les niveaux du modèle (noyau fonctionnel, modèle de tâches abstrait et concret). Au contraire, les outils des systèmes basés sur modèles, quand ils existent, n'interviennent qu'au stade final du développement. Ils ne s'intéressent dans la majorité des cas qu'au niveau concret du modèle. Ainsi, l'approche SUIDT permet-elle au concepteur d'optimiser les modèles en fonction d'éventuelles incohérences établies par les outils de validation.

Au niveau du deuxième point, l'outil SUIDT n'a pas besoin de générer l'interface pour la tester. Elle est directement interprétée puisque le GUI-Builder, manipule directement des objets de la présentation. Plus précisément, le noyau fonctionnel de l'application est existant et il est indirectement lié avec la couche présentation. L'avantage est de donner au concepteur la possibilité d'alterner la phase de modélisation et celle de test sans perdre le contexte d'exécution.

Enfin, les outils de modélisation de l'approche SUIDT utilisent des techniques (programmation visuelle, notion de langage de programmation réduit) qui permettent à des non spécialistes en programmation de prototyper rapidement des applications interactives.

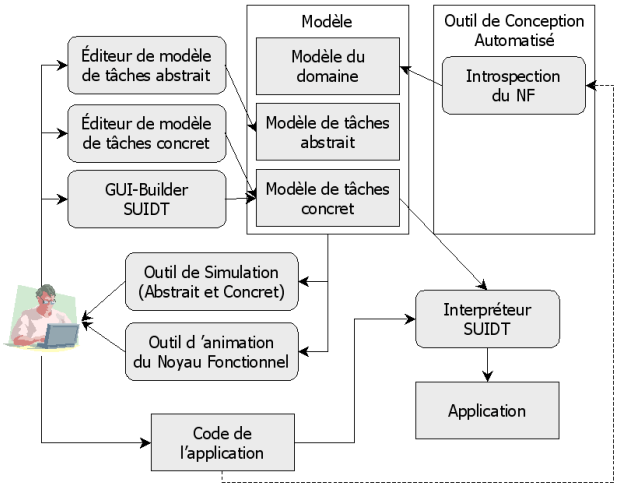

Figure 6 – Processus de développement de SUIDT, adapté de [7].

Nous présentons sur la *Figure 6*, le processus de développement d'une application interactive avec l'outil SUIDT suivant l'architecture générale du processus de développement d'un MBS défini par [7]. Elle met en avant le fait que SUIDT cadre parfaitement dans les approches des systèmes basés sur modèles.

EXEMPLES

Nous avons réalisé un certain nombre de noyaux fonctionnels formels afin de tester l'utilisabilité de l'environnement⊡un compteur avancé, un convertisseur franc/euro et une base de données qui gère une liste d'étudiants.

CONCLUSION

Nous avons présenté l'outil de construction SUIDT (Safe User Interface Development Tool) qui permet de concevoir et de vérifier des applications interactives suivant une architecture précise et la prise en compte de besoins utilisateurs. La prise en main est facilitée par la programmation visuelle qui permet à des non spécialistes de la programmation de prototyper rapidement des systèmes interactifs sûrs.

Son développement a été réalisé au moyen du langage JAVA®, de la boîte à outils Swing®, du langage B et de l'Atelier B. Une version téléchargeable est disponible à cette adresse⊡http://www.lisi.ensma.fr/members/baron.

Des informations complémentaires au sujet de l'approche SUIDT peuvent être trouvées dans [2].

BIBLIOGRAPHIE

1. Abrial, J.-R. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
2. Baron, M. et Girard, P. SUIDT : A task model based GUI-Builder. In *Proceedings of TAMODIA : Task MOdels and DIAgrams for user interface design* (July 18-19, 2002, Romania, Bucharest), Inforec Printing House, 2002, pp. 64-71.
3. Bass, l., Pellegrino, R., Reed, S., Sheppard, S. et Szezur, M. The Arch Model : Seeheim revisited. In *Proceedings of User Interface Developper's Workshop* ,1991.
4. Fekete, J.-D. Les trois services du noyau sémantique indispensables à l'IHM. In *Proceedings of Journées Francophones sur l'Ingénierie de l'Interaction Homme-Machine (IHM'96)* (16-18 septembre, Grenoble), Cépaduès, 1996, pp. 45-50.
5. Paternò, F. *Model-Based Design and Evaluation of Interactive Applications*. Springer, 2001.
6. Puerta, A.R., Cheng, E., Ou, T. et Min, J. MOBILE : User-Centered Interface Building. In *Proceedings of* (15-20 May, Pittsburgh PA USA), ACM/SIGCHI, 1999, pp. 426-433.
7. Szekely, P. Retrospective and challenge for Model Based Interface Development. Bodart, F. et Vanderdonckt, J. (Ed.). In *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96)*, Springer-Verlag, Namur, Belgium, 1996, pp. 1-27.