Baron, M. et Girard, P. Vers un développement sûr d'applications interactives. In *Proceedings of IHM-HCI 2001* (10-14 septembre 2001, Lille, France), Cépaduès-Éditions, 2001, pp. 155-158.

Vers un développement sûr d'applications interactives

Mickaël Baron, Patrick Girard

Laboratoire d'Informatique Scientifique et Industrielle, ENSMA
1 rue Clément Ader, 86961 Futuroscope Chasseneuil
http://www.lisi.ensma.fr/ihm
{baron, girard}@ensma.fr
Tel. (33/0) 5-49-49-80-70

RESUME
Pour développer des applications interactives sûres, deux approches peuvent aujourd'hui être mises en parallèle : les Systèmes Basés sur Modèles (Model-Based Systems) d'une part, et les approches formelles d'autre part. Malgré des avancées intéressantes, aucune d'elles n'est encore parvenue à s'imposer.
Nous explorons dans ce travail une voie qui peut être qualifiée d'intermédiaire, puisqu'elle associe un développement formel d'application non interactive à un système de conception interactif, dans le but de conjuguer les apports de ces deux classes d'approches.

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

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

INTRODUCTION

Le développement de logiciels de qualité nécessite l'utilisation de techniques très rigoureuses. Elles doivent assurer que les logiciels en développement satisfont les propriétés qui traduisent les exigences définies dans le cahier des charges. Etablir des propriétés sur les éléments d'un programme nécessite l'utilisation de systèmes formels. L'une des démarches proposées consiste dans un premier temps à définir un ou plusieurs modèles à l'aide d'un langage formel pour décrire les objets programmes à développer, puis dans un deuxième temps, à fournir un système de preuve qui permet de prouver des propriétés sur les objets programmes définis. À partir de ces modèles prouvés, il est ensuite possible de générer ou de développer une application, exacte représentation des modèles.

C'est en partie sur ce mode de développement que s'appuient les outils de la famille des Systèmes Basés sur Modèles (MBS en anglais pour Model-Based Systems) qui partent de modèles (modèle de tâche, modèle de l'application…) pour générer une application. S'ils définissent entièrement l'application à l'aide de modèles [7] (comme Mastermind [8] ou ITS [9] par exemple), ces derniers disposent rarement d'outils de vérification. Ne s'appuyant pas sur une sémantique parfaitement définie, ils ne peuvent bénéficier des outils développés dans d'autres domaines. De plus, ces systèmes demandent une maîtrise des concepts qu'ils manipulent, ce qui requiert un important apprentissage.

Sur un autre plan, de nombreux travaux ont été menés dans le but d'introduire une démarche formelle dans le développement des systèmes interactifs [4] et [2]. En dépit de certaines réussites prometteuses, ces approches ne se sont pas imposées, le principal obstacle à leur généralisation résidant dans le surcoût souvent prohibitif de leur mise en œuvre.

Nous explorons dans ce papier une autre voie pour le développement d'applications interactives sûres. Partant d'une étude [3] démontrant comment l'on pouvait réaliser totalement interactivement une application à partir d'un noyau fonctionnel préalablement développé, nous étudions ici la possibilité de s'appuyer sur un noyau fonctionnel totalement sûr, et envisageons les moyens de garantir les propriétés de ce noyau fonctionnel dans l'application finale.

Notre démarche entre dans le cadre des MBS en ce sens qu'elle s'appuie sur un modèle pour construire l'application finale. Ce modèle est un ***modèle*** unique : le ***noyau fonctionnel***. Ce noyau fonctionnel est spécifié formellement, ce qui nous permet de définir des propriétés sur le contenu de ce noyau fonctionnel et de vérifier ces propriétés.

Dans une deuxième étape, l'interface de l'application est réalisée grâce un outil de développement graphique qui utilise ce modèle avec toutes les informations dont il dispose (spécifications formelles, propriétés…). Le développement de l'interface consiste à associer le modèle avec les objets graphiques de l'interface tout en s'assurant que cette interface garantit les propriétés du noyau fonctionnel. Cette démarche, pas complètement formelle, peut être qualifiée de sûre puisqu'elle garantit le respect par l'interface des propriétés formellement établies du noyau fonctionnel.

Dans une première section, nous décrivons le modèle formel et la technique utilisée, en précisant les informations à retenir. Ensuite, nous expliquons le mécanisme de construction de l'interface de l'application et comment

nous lions le modèle avec l'interface de l'application. Enfin, nous mettons en lumière les différents problèmes et ouvertures que cette démarche laisse entrevoir.

UN NOYAU FONCTIONNEL FORMEL

Un premier travail a abouti au système GenBUILD [3]. Cet outil génère automatiquement une application interactive à partir de l'interface d'un noyau fonctionnel préalablement développé. Cette application interactive est constituée d'un ensemble de boutons capables d'appeler chacune des primitives du noyau fonctionnel, invoquant une boîte de dialogue générée automatiquement pour entrer les paramètres de chaque action. L'originalité de la démarche réside dans le fait que cette application générée automatiquement est couplée à un générateur d'interface qui permet de construire une véritable application interactive tout en conservant le lien avec le noyau fonctionnel. Le concepteur de l'application peut passer alternativement d'une phase de création d'interface à une phase de test, et vice versa, sans perdre son contexte de test.

La limite de la méthode réside dans le langage utilisé pour la description du noyau fonctionnel. Nous sommes partis d'un définition des fonctions en langage C^{++}, à laquelle nous avons ajouté des expressions comme des intervalles de valeur, permettant de vérifier dans l'application finale l'appel correct des fonctions du noyau fonctionnel. Cependant, l'absence d'une vraie description formelle empêche tout raisonnement sur le noyau fonctionnel.

Un langage formel

Notre démarche initiale, à base d'expressions correspondant en fait à des pré-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. Nous avons choisi le langage B [1], qui, associé à l'environnement de développement « Atelier B » [6], autorise la conception complète d'applications, depuis la phase de spécification jusqu'à la phase d'implémentation. L'intérêt de la méthode B est d'assurer le respect des propriétés, exprimées au moment des spécifications, tout au long du processus de développement. La technique de preuves utilisée est la démonstration de théorèmes qui une fois prouvés assurent le maintien des propriétés. Nous allons nous attacher premièrement à décrire sommairement la structure d'un modèle en B, puis à comprendre le mécanisme de preuve des propriétés du modèle, pour ensuite finir sur le traitement de ces spécifications pour en générer une interface du modèle pour le développement du reste de l'application.

La notion de « machine abstraite » est le mécanisme de base de la méthode B. La méthode définit trois types de machines identifiées par les mots clefs ABSTRACT, REFINEMENT et IMPLEMENTATION. Ces trois types de composants représentent trois étapes successives vers une spécification proche d'un langage de programmation exécutable. La première machine abstraite, comme son nom l'indique, est le composant fondamental, dans lequel on définit le squelette du futur programme. Elle représente le plus haut niveau de spécification. Cette machine exprime des spécifications formelles dans un langage de haut niveau d'abstraction. Le deuxième type (REFINEMENT) représente les étapes intermédiaires du raffinement. Ainsi les spécifications subissent des transformations permettant de concrétiser les spécifications abstraites, pour se rapprocher de plus en plus de l'implémentation. Finalement, le dernier niveau (IMPLEMENTATION) permet d'atteindre l'étape de codage, en langage de programmation. La méthode B est l'une des rares méthodes formelles qui a démontré sa réelle exploitabilité dans des applications.

Comme dans tout langage, on déclare des variables qui sont tous les attributs représentés dans le modèle. L'invariant décrit les propriétés des attributs. Il s'agit d'expressions logiques qui doivent toujours être assurées en permanence. Les opérations (les fonctions ou les procédures qui accèdent ou modifient les variables) peuvent avoir zéro, un ou plusieurs paramètres d'entrée et zéro, un ou plusieurs paramètres de sortie. Le corps des opérations peut utiliser les paramètres de sortie, les paramètres d'entrée, et toutes les variables de la machine.

Le développement B commence par la construction d'une machine de base qui reprend toutes les descriptions du besoin et décrit les principales variables du système, les opérations et les propriétés que ces variables devront satisfaire. Le modèle B consiste en une spécification de ce que devra réaliser le modèle. Ensuite celui-ci va être spécialisé c'est-à-dire raffiné, jusqu'à obtenir un dernier raffinement qui sera l'implantation finale du noyau fonctionnel.

Valider le modèle formel

Nous allons maintenant voir les principales preuves effectuées lors d'un développement formel B qui nous permettent de valider le modèle formel. Pour synthétiser cette description nous pouvons dire qu'il y a deux familles de preuves à garantir.

La première concerne l'invariant. Nous avons vu que celui-ci dépendait des variables de la machine abstraite B et que seules les opérations de la machine pouvait modifier ces variables. Pour prouver l'invariant, il suffit de montrer qu'il est établi pendant l'opération d'initialisation, et qu'il est préservé par chaque opération. La véracité de l'invariant est alors certaine pendant toute l'exécution du programme.

La deuxième série concerne les preuves de raffinement. Pour évoluer de la machine abstraite vers le code final, le modèle subit des transformations appelées raffinements. Pour montrer qu'un raffinement est correct, il faut démontrer qu'il ne contredit pas les propriétés exprimées dans le raffinement de niveau immédiatement supérieur. La Figure 1 donne un exemple d'un tel raffinement :

```
1   Spécification invariant var1 : NAT
2   Raffinement invariant var1 : 0..10
3   PO : 0..10 ⊂ NAT
```

Figure 1 : Exemple de preuve de raffinement

La ligne 1 décrit l'invariant d'une machine (la variable var1 doit appartenir aux entiers naturels) alors que la ligne 2 exprime un raffinement (la variable var1 doit être comprise entre 1 et 10). La preuve de raffinement consiste à prouver la ligne 3. Ces preuves sont prouvées automatiquement par les outils de l'atelier B afin de valider le modèle. Ensuite ces outils généreront un code exécutable du noyau fonctionnel.

GENERATION D'INTERFACE

La méthode exposée en [3] peut s'appliquer directement à un noyau fonctionnel exprimé en B. L'analyse des spécifications à un niveau suffisant de détail (lorsque les variables sont concrètement exprimées) permet de générer automatiquement une interface de manipulation du modèle. Cette interface représente visuellement le noyau fonctionnel formel et permet de vérifier son fonctionnement. Cette première interface représente une interface pour le développement du reste de l'application. Mais elle assure aussi visuellement que les invariants sont établis. Pour générer cette interface de développement, il faut d'abord récupérer toutes les signatures des opérations, comprenant le nom de l'opération, les paramètres d'entrées et les paramètres de sorties.

```
1   Int_num ← accesseur(ligne, colonne) =
2   PRE         ligne : LIGNE &
3               colonne : COLONNE
4   THEN        Int_num := tab(ligne, colonne)
5   END
```

Figure 2 : Exemple d'opération

Sur la Figure 2 est représentée un exemple d'une opération. on peut y voir deux paramètres d'entrée et un paramètre de sortie. Les paramètres d'entrée et sortie respectent des pré- et post-conditions. Ces conditions permettent de connaître le typage des paramètres ainsi que des conditions sur ces paramètres. Il s'agit en fait des conditions à satisfaire avant et après l'exécution de l'opération. Dans notre exemple les pré-conditions indiquent que les paramètres sont de type *LIGNE* ou *COLONNE*. Ces deux types sont des variables globales à cette machine et typés au niveau de l'invariant. Donc l'outil de génération va pouvoir analyser l'invariant pour déterminer le typage de *LIGNE* et *COLONNE*. De la même façon, dans le cas du paramètre de retour (ligne 4), la post-condition annonce comme information un typage et une affectation.

L'outil de génération n'a pas besoin de prendre en compte tous les composants du modèle B. Les différentes spécifications étant cohérentes les unes par rapport aux autres, il suffit de choisir le niveau le plus riche en informations pertinentes. De même l'outil n'a pas besoin d'analyser la partie implémentation.

L'INTERFACE DE L'APPLICATION

Le concepteur peut ensuite fabriquer l'aspect graphique de l'interface. Il emploie un deuxième outil interactif qui s'appuie sur des techniques de programmation visuelle évoluées comme la programmation sur l'exemple et la manipulation directe qui sont de nature à réduire le cycle de développement de l'interface. C'est un outil de création d'interface classique que l'on trouve sur plusieurs environnements de développement (comme par exemple Delphi© de Borland®). Peu d'outils qui partent de la définition de modèle offrent cependant la possibilité de définir l'interface par la manipulation directe [5].

L'aspect graphique de l'interface terminé, le concepteur s'emploie à définir l'aspect du contrôle du dialogue plus précisément à programmer les actions et les événements de l'interface. Les techniques de programmation vont lui permettre de lier le modèle via l'interface de développement générée par le premier outil avec l'interface graphique de l'application. Pour cela le principe de programmation retenu consiste à associer directement ces actions ou ces événements aux opérations de notre modèle validé. Pour y arriver cet outil de développement d'interface doit proposer un ensemble prédéfini d'actions et d'événements à faire correspondre aux objets de l'interface. Il n'y a alors qu'à choisir le type d'action à placer sur l'objet et y associer une opération du modèle simplement par manipulation visuelle. Des outils comme Visual C^{++}© de Microsoft Corporation® nécessitent à l'inverse un codage de l'application pour chaque événement de l'objet.

Le modèle utilisé propose des spécifications hautement formelles. Chaque opération du modèle est définie par des pré-conditions et des post-conditions. Ces dernières vont ainsi renseigner le concepteur sur la manière dont il faut utiliser les opérations. Les pré-conditions spécifient les propriétés de l'opération avant son exécution, c'est à dire spécifier les paramètres (typage de variables, conditions logiques à respecter). Les post-conditions indiquent les conditions à satisfaire après l'exécution de l'opération. Par exemple le typage des paramètres de sortie ou le domaine des valeurs retournées. La forme des spécifications en B s'avère très intéressante pour la création de l'application interactive. En effet, ces pré- et

post-conditions donnent des informations directement exploitables pour construire l'interface tout en respectant les règles classiques d'ergonomie. Sur l'exemple de la Figure 2, on voit que les paramètres *ligne* et *colonne* doivent être du type *LIGNE* et *COLONNE* et que le paramètre de sortie *Int_num* est du type *tab(ligne, colonne)*. Ceci permet à l'outil de ne proposer lors de la conception que les widgets susceptibles de représenter de tels objets. De même, les informations contenues dans les invariants permettent d'implémenter automatiquement les autorisations/invalidations proactives de commandes.

Peut-on pour autant garantir que les invariants du modèle sont toujours respectés au cours de l'exécution de l'application ? Les invariants sont-ils préservés par chaque opération ? L'architecture choisie permet d'obtenir une telle garantie : au niveau du modèle, l'invariant sera toujours respecté car le modèle a été prouvé. Mais peut-on être sûr que l'interface satisfait les exigences des invariants ? Le fait d'associer directement les opérations aux actions et aux états de l'interface dans le respect des pré- et post-conditions permet d'obtenir très simplement cette preuve : le système interactif utilise le noyau fonctionnel d'une façon cohérente avec sa spécification. Il respecte donc ses propriétés.

Enfin, une des caractéristiques importantes de notre approche est le test de l'application avant même que l'interface finale ne soit finalisée. Étant donné que nous allons directement associer l'interface du modèle avec l'interface de l'application nous n'aurons pas besoin de passer par une phase de compilation classique. C'est un peu le principe de Mastermind, qui ne permet cependant pas de tester le modèle avant qu'il ne soit terminé. Mais cette approche ne permet pas d'assurer que le modèle assure un niveau de sécurité élevé du noyau fonctionnel. De plus sans cette phase de compilation le passage d'un état d'édition à un état de test conserve le contexte d'exécution. Ce qui peut être intéressant pendant des phases de tests intensifs.

CONCLUSION ET PERSPECTIVES

Notre démarche autorise la conception d'application en garantissant les propriétés de l'application au niveau du noyau fonctionnel et de l'interface. Elle offre l'avantage de concevoir un noyau fonctionnel considéré comme un modèle pour lequel on est certain qu'il garantira les exigences souhaitées. Ensuite un deuxième outil interactif va simplifier le travail de création d'interface simplement en associant des opérations à cette interface.

La limite actuelle de la méthode réside dans l'absence de modèle de tâche de l'application à réaliser. Le fait de garantir que l'interface obéit aux règles édictées par le noyau fonctionnel ne garantit en aucune façon que les buts des utilisateurs peuvent être atteints. C'est un point important que nous envisageons dans un proche avenir.

Au niveau de la génération nous avons parlé d'une méthode que devrait suivre le concepteur pour modéliser son modèle. Outre le fait qu'il doit spécifier, raffiner et implémenter son modèle, il doit aussi s'assurer que son modèle respecte des exigences de conception dont l'outil de génération a besoin.

Il reste un travail important à accomplir dans le domaine de la visualisation des spécifications formelles des opérations, i.e. les pré-conditions et les post-conditions, pendant la programmation de l'interface. Sous quelle forme les afficher ? Doit-on montrer directement les assertions logiques, ou doit-on passer par une représentation schématique ?

BIBLIOGRAPHIE

1. Abrial, J.-R. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.
2. Aït-Ameur, Y., Girard, P. et Jambon, F. A Uniform approach for the Specification and Design of Interactive Systems: the B method. In *Proceedings of Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)* (3-5 June, Abingdon, UK), 1998, pp. 333-352.
3. Baron, M. et Girard, P. Construction interactive d'application à partir du noyau fonctionnel. In *Proceedings of Ergonomie et informatique avancées (Ergo-IHM'2000)* (3-6 octobre 2000, Biarritz, France), ESTIA, 2000, pp. 85-93.
4. Jambon, F., Brun, P. et Aït-Ameur, Y. Spécifications des systèmes interactifs (chapitre 6). Kolski, C. (Ed.). In *Analyse et conception de l'I.H.M. / Interaction Homme-Machine pour les S.I. vol.1*, Hermès Science, Paris, France, 2001, pp. 175-206.
5. 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.
6. Steria Méditerranée. *Atelier B*. 1997.
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.
8. Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. et E. Salcher. Declarative interface models for user interface construction tools : the MASTERMIND approach. In *Proceedings of IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)* (14-18 August, Grand Targhee Resort (Yellowstone Park), USA), Chapman & Hall, 1995, pp. 120-150.
9. Wiecha, C., Bennet, W., Boies, S., Gould, J. et Greene, S. ITS: a tool for rapidly developing interactive applications. *ACM Transactions on Information Systems*. 8, 3 (1990), pp. 204-236.