Utilisation de techniques formelles dans la modélisation d'Interfaces Homme-Machine. Une expérience comparative entre B et Promela/SPIN

Yamine Ait-Ameur, Mickael Baron, Nadjet Kamel

Utilisation de techniques formelles dans la modélisation d'Interfaces Homme-Machine. Une expérience comparative entre B et Promela/SPIN.

Y. Aït Ameur M. Baron et N. Kamel
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA, BP 40109, 86961 Futuroscope Cedex, France
{yamine,baron,kamel}@ensma.fr
http ://www.lisi.ensma.fr/ihm

Résumé

Cet article présente une expérience dans l'utilisation des techniques formelles pour la conception, la vérification et la validation d'IHM au travers de l'utilisation de deux techniques dans un développement incrémental. La première technique est fondée sur le raffinement et la preuve avec B et la seconde est fondée sur la vérification sur modèle avec Promela/SPIN. Différentes propriétés telles que la robustesse, l'atteignabilité, l'insistance sont vérifiées grâce à l'utilisation de ces techniques. Nous discutons également les avantages et inconvénients de ces deux techniques.

1 Introduction

La conception et le développement de systèmes interactifs en général et d'Interfaces Homme-Machine (IHM) en particulier met en œuvre différentes techniques issues aussi bien du génie logiciel que de l'ergonomie ou de la psychologie. Ces techniques sont souvent informelles ou bien semi-formelles et sont supportées par des notations où le graphisme est un élément important. Ces techniques sont utilisées dans les différentes phases du développement : spécification, conception, validation, vérification etc. Cependant, le caractère informel associé à ces techniques engendre des ambiguités d'interprétation et donc la multiplication des étapes de validation et de vérification.

Il est maintenant acquis que les techniques formelles utilisées dans les phases de spécification, conception, validation, vérification etc. permettent de lever certaines ambiguïtés issues de l'utilisation de techniques semi-formelles. En effet, la présence d'une sémantique formelle et d'un système de preuve permet de décider, sans ambiguïté, de la consistance d'un modèle et des propriétés associées aussi bien dans le cas d'une validation que dans celui d'une vérification.

L'objectif de nos travaux consiste à étendre l'utilisation de ces techniques formelles, classiquement utilisées en génie logiciel, aux systèmes interactifs que représentent les IHM. Mais, de nombreux obstacles à l'utilisation de telles techniques dans ce domaine d'application subsistent. Les techniques formelles :

- ont été utilisées dans le génie logiciel pour la conception, la validation et la vérification de systèmes logiciels. Leur impact sur les domaines de l'ergonomie et de la psychologie a été très peu étudié [CMAA01]. La formalisation de modèles et de propriétés issus de ces domaines reste un problème entier ;
- permettent l'expression, la vérification et la validation de différentes propriétés des IHM, mais il est bien connu que chaque technique ne peut prendre en compte tous les types de propriétés. Souvent, il est nécessaire de recourir à l'utilisation de plusieurs techniques formelles ;
- sont différentes sur le plan de la sémantique et du système de preuve qu'elles supportent et engendrent donc un hétérogénéité sémantique forte.

Dans ce contexte, nous nous proposons d'étudier l'utilisation de différentes techniques formelles dans le domaine des IHM. Notre objectif est d'étudier au travers d'une étude de cas, l'utilisation de deux techniques formelles différentes et distantes, pour concevoir des systèmes d'IHM et vérifier des propriétés sur de tels systèmes en partant des différentes notations semi-formelles et des pratiques de conception habituelles.

Cet article relate deux expériences conjointes. Il est structuré de la façon suivante. La section 2 présente les différentes notations et modèles utilisés pour le développement d'une IHM. Nous passons en revue les différentes techniques semi-formelles utilisées et nous donnons les motivations pour l'utilisation de

techniques formelles. La section 3 présente une étude de cas interactive et concurrente. Malgré sa simplicité, cette étude de cas illustre les différents problèmes posés par l'utilisation de techniques formelles dans ce domaine d'application. La section 4 présente les utilisations de deux techniques formelles sur l'étude de cas de la section 3. Deux techniques formelles sont présentées et utilisées : une technique formelle fondée sur la preuve et le raffinement avec la méthode B, et une technique formelle fondée sur les systèmes de transitions et la vérification sur modèle (model checking). La preuve et la vérification formelles sont également discutées. La dernière section présente la conclusion et les perspectives ouvertes par cette étude.

2 Modèles et notations dans les IHM

Le domaine des IHM a vu naître de nombreux modèles et notations semi-formels où le graphisme est un élément important. Ces modèles et notations proviennent des différentes communautés citées précédemment (informaticiens, ergonomes, psychologues etc.). Ils ont suivi les différentes évolutions matérielles et logicielles de ces dernières années. Nous les passons brièvement en revue ci-dessous. Nous séparons les notations de descriptions (description de tâches, description de l'interaction), souvent issues des non-informaticiens, des notations en conception et en architecture propres aux développements informatiques.

2.1 Notations pour la description d'IHM

Afin d'exprimer les besoins des utilisateurs, souvent des non-informaticiens, de nombreuses notations de descriptions d'IHM ont été proposées. Elles sont, pour la plupart, centrées utilisateurs et sont donc loin des implantations informatiques.

MAD [SPG89], pour Méthode Analytique de Description, utilise une notation arborescente qui décrit les différentes tâches répertoriées par l'utilisateur. Chaque niveau dans l'arbre correspond à un niveau de décomposition de tâches. Une représentation par des expressions régulières peut être extraite de cette notation. HTA (Hierarchical Task Analysis) [DFAB93], est une autre notation, similaire à MAD, qui utilise une décomposition des tâches en fonction de leur but, puis des sous-buts etc. Elle est fondée sur la description des objectifs des tâches. D'autre part, une notation telle que UAN [HH93] et son extension XUAN [GEM94] décrit non seulement le comportement de l'interface mais aussi celui de l'utilisateur. Elle utilise une notation à base de triplets comportant l'action de l'utilisateur, la réaction de l'interface et l'état de l'interface. Dans ce cas, la description d'une interface revient à décrire une table regroupant les triplets. Toutes ces notations informelles constituent, en général, la base de départ de réalisation d'une IHM.

Ces notations ne sont pas utilisées dans cet article.

2.2 Modèles d'architecture et de conception

Tous les modèles d'architecture définis pour les IHM séparent la conception de l'IHM de la conception de l'application ou noyau fonctionnel. Ils utilisent un mécanisme d'association entre les deux parties. Nous donnons une description sommaire de trois d'entre eux.

Le modèle de Seeheim [Pfa85] est un modèle global qui comprend trois modules : la présentation, le contrôle du dialogue et l'interface avec l'application. Le modèle PAC [Cou87] (Présentation, Abstraction et Contrôle), qui inclut le noyau fonctionnel dans l'abstraction, est en quelque sorte une décomposition du modèle de Seeheim. Il décompose l'application en plusieurs agents eux-mêmes séparés en présentation d'un objet de l'interface, application (Abstraction) et contrôle du dialogue. Ce contrôle permet d'établir la liaison entre application et présentation au sens de Seeheim. Enfin, le modèle ARCH [BPR+91] [BHH+88] intègre l'ensemble de l'application depuis le noyau fonctionnel jusqu'à la boîte à outils utilisée pour la présentation en passant par le contrôle de l'application.

Les modèles précédents utilisent une seule logique de décomposition. D'autres modèles intégrant plusieurs points de vue, qualifiés de modèles hybrides, ont été développés [Gui95][Fek96].

Cet article utilise le modèle d'architecture ARCH.

2.3 Propriétés dans les IHM

Les propriétés dans les IHM sont les propriétés couramment définies pour les systèmes interactifs en général telles que les propriétés de sûreté et d'équité (d'honnêteté) auxquelles sont ajoutées les propriétés particulières aux IHM, c'est-à-dire les propriétés inhérentes au domaine des IHM comme celles liées à la représentation ou les propriétés issues des exigences des ergonomes ou des psychologues.

La grande majorité des travaux s'est intéressée au problème de l'expression et de la vérification de propriétés. [DFAB93] [HT] [DH95] [Roc98] décrivent une classification des propriétés. Nous les résumons dans ce qui suit.

Deux grandes classes de propriétés ont été définies : les propriétés de validité et les propriétés de robustesse.

- *Propriétés de validité :* ce sont les propriétés qui caractérisent un fonctionnement attendu ou voulu par un utilisateur. Dans cette catégorie de propriétés, on distingue les propriétés de complétude (réalisation d'un objectif donné), de flexibilité pour la représentation de l'information et pour le déroulement des tâches (utilisateurs multiples, non-préemption, atteignabilité des états), etc.
- *Propriétés de robustesse :* ce sont les propriétés relatives à la sûreté de fonctionnement du système. Parmi ces propriétés, on trouve les propriétés liées à la visualisation du système telles que l'observabilité, l'insistance et l'équité (l'honnêteté).

Cet ensemble de propriétés permet de qualifier la facilité d'utilisation d'une interface.

2.4 Nécessité de la formalisation

Les différents modèles et notations pour la description des tâches et des interfaces ne donnent qu'un synoptique et une structuration de l'interface qui peuvent être utilisés pour le développement. L'absence de sémantique formelle fait que ces notations ne sont applicables qu'empiriquement. De plus, nous ne connaissons pas de système implantant ce type de modèle ou de notation qui permette un développement incrémental.

En conclusion, tous les contrôles sont reportés à l'étape de vérification par les tests et n'utilisent que les techniques classiques de vérification et de test de programmes. Ce sont ces raisons qui font que nous préconisons l'utilisation de techniques formelles.

3 Etude de cas

Nos travaux ont pour domaine d'application les IHM dans les systèmes critiques (avionique par exemple). Pour exposer notre démarche, nous avons choisi une étude de cas simple. Elle est composée de deux applications dépendantes : une application de conversion francs/euros et un compteur.

3.1 Convertisseur francs/euros et Compteur

Le convertisseur est un logiciel utilitaire permettant de réaliser des conversions d'une somme en francs en son équivalent en euros et vice versa. L'utilisateur saisit la valeur qu'il désire convertir, choisit le sens de la conversion, déclenche la conversion, puis lit le résultat dans la monnaie souhaitée. Après chaque conversion l'application compteur incrémente d'une unité une valeur compteur et l'affiche. Au bout de trois conversions, l'application compteur interdit la conversion tant que l'utilisateur n'a pas ré-initialisé la valeur compteur (voir Fig 1).

(a) (b)

FIG. 1 – Etude de cas composée d'un convertisseur francs/euros (a) et d'un compteur (b)

3.2 Aspects interactifs de l'étude de cas

L'utilisateur interagit avec les applications convertisseur et compteur via des interfaces graphiques composées de widgets (composants graphiques) comme des fenêtres, des boutons, des textes et des zones de saisie. Les interactions de l'utilisateur se limitent à la saisie de la valeur à convertir, à l'appui sur un bouton *E»F* ou *F»E* et à l'initialisation du compteur par l'appui du bouton *Initialiser*. Quand trois conversions ont été effectuées les interactions de l'utilisateur se limitent à l'application compteur. Inversement tant que ces trois conversions n'ont pas été réalisées, l'utilisateur ne peut interagir que sur le convertisseur. La valeur compteur est incrémentée à chaque nouvelle conversion. La visualisation des données est obtenue

par une zone de texte qui retourne la valeur convertie et d'un texte qui affiche la valeur du compteur. Dans cette étude de cas que nous traitons, nous nous penchons principalement sur une application de type WIMP (Windows, Icons, Menus and Pointers). Il s'agit du type d'applications le plus couramment utilisée.

4 Modélisation formelle d'une IHM

Les insuffisances évoquées précédemment ont conduit les chercheurs de la communauté à proposer l'utilisation de techniques formelles pour résoudre certains de ces problèmes (de vérification et de validation). L'application des techniques formelles dans les IHM a suivi l'évolution historique de ces techniques et de leur utilisation dans le génie logiciel [Gau95].

Les premiers modèles ont utilisé des automates et/ou des extensions d'automates tels que les *statecharts* [Har87] ou les ATN [Was81]. [Jac82] fut le premier à utiliser les automates pour la spécification du comportement des interfaces à l'aide de systèmes de transition. Des modèles tels que H^4 [Gui95] et [Was81] ont utilisé les ATN afin d'éviter le problème de l'explosion du nombre d'états. D'autres modèles se sont fondés sur des réseaux de Petri [Pal92] comme moyen de représentation du système interactif. Plus récemment, des extensions des automates (BDD) ont été utilisées comme modèle pour des logiques temporelles telles que CTL, CTL* [CES86] ou XTL [Bru97]. Le système SMV [McM92] a été mis en œuvre par [AWM95] pour vérifier, par *model-checking*, des propriétés des IHM exprimées dans la logique CTL. Le langage synchrone LUSTRE a permis de vérifier et de générer du code dans les IHM à partir de descriptions écrites en UiL/X [Roc98].

En parallèle, d'autres techniques orientées modèle, basées sur la vérification d'obligations de preuve et sur la description incrémentale, ont été expérimentées sur les IHM comme Z [DH93b] [DH95] et VDM [Mar86].

La spécification et la vérification d'IHM ont vu aussi l'utilisation des techniques algébriques. [PF92] furent les premiers à introduire la notion d'interacteur en utilisant LOTOS [Sys84] (issu de la fusion de ACT ONE pour la partie données et de CCS pour la partie interaction).

Enfin, nous ne saurions être exhaustif si nous ne citions pas les travaux basés sur les systèmes à base de logique d'ordre supérieur. En effet, le système HOL [GP94] a été utilisé par [BACL95] pour la description d'IHM et pour la vérification de propriétés. Le *theorem prover* de HOL est utilisé dans ce but.

Toutes les techniques précédemment décrites permettent l'expression, à un haut niveau d'abstraction, de descriptions, de comportements et de fonctions des interfaces ainsi que la possibilité de les valider. La vérification de propriétés est aussi autorisée dans ce type de système soit automatiquement (par *model-checking*) soit semi-automatiquement (en utilisant un *theorem prover* qui permet de décharger les obligations de preuve).

4.1 Systèmes de transitions étiquetés : STE

Les systèmes de transitions étiquetés (STE) décrivent le comportement des systèmes. Un système est décrit par un ensemble d'états, un ensemble d'actions, un état initial et une relation entre les états, appelée relation de transition. Les actions effectuées par le système permettent la transition d'un état à un autre. Un système complexe peut être obtenu par composition de sous systèmes, chacun étant représenté par un STE. La composition est obtenue par une opération de produit synchronisé .

Les systèmes interactifs, en particulier les IHM, peuvent être représentés par des systèmes de transitions. En effet, ces systèmes permettent de représenter différents composants (chacun par un STE) qui sont assemblés par une relation de synchronisation (produit synchronisé). Lorsqu'une telle modélisation est obtenue, il est possible de représenter des propriétés et de les vérifier ou bien de générer du code. L'approche de formalisation de systèmes interactifs a donné lieu à de nombreux travaux [Pal92][HT90][DH93a][PF92].

Dans nos travaux, nous avons choisi deux techniques pour coder des STE. La première est une technique fondée sur la preuve et le raffinement avec B [Abr96a] et la seconde est fondée sur la vérification sur modèle (model checking) avec Promela-Spin [Pro93].

La suite de cette section présente les deux techniques B et Promela Spin et leurs mises en œuvre sur l'étude de cas de la section 3. Nous utilisons un même modèle d'architecture : ARCH.

4.2 Une approche par composition : Raffinement et preuve avec B

La méthode B permet de conduire des développements de façon incrémentale. La spécification est construite au fur et à mesure du développement. C'est le raffinement qui donne une définition de plus en

plus précise de cette spécification. De plus la méthode B autorise la modularité par la présence d'opérateurs particuliers permettant l'inclusion, l'extension de machines B. A la différence de Promela-Spin, le mécanisme de preuve en B n'est pas fondé sur une évaluation de propriétés sur modèle (model-checking). B utilise un prouveur qui permet de décharger des obligations de preuve. Par l'intermédiaire du raffinement, la plupart de ces obligations de preuves sont déchargées automatiquement. Enfin, la méthode B est parfaitement instrumentée par l'environnement de développement *Atelier B* [Cle97] qui permet de supporter toutes les phases de développement depuis la spécification jusqu'à la génération de code. Il génère automatiquement toutes les obligations de preuve et est doté à la fois d'un prouveur automatique et interactif.

4.2.1 B et l'architecture logicielle

A l'image des travaux [AAGJ98a][AAGJ98b], nous utilisons le concept simplifié d'encapsulation pour construire les machines abstraites B et nous suivons une structure propre au modèle d'architecture ARCH. Différentes machines abstraites sont définies pour correspondre au domaine, adaptateur du domaine, contrôleur de dialogue, présentation et boîte à outils. La modularité de B est employée pour assembler en différentes machines (boîte à outils, contrôleur de dialogue, noyau fonctionnel...).

La construction du contrôleur de dialogue se base sur une approche à base d'événements dans le sens où les actions émises par l'utilisateur correspondent à des événements instantanés que ce contrôleur de dialogue doit gérer. Nous utilisons une extension de B définie dans [Abr96b] appelée *B-événementiel*. Cette extension permet de s'intéresser par exemple aux systèmes distribués, réactifs, interactifs, multi-modaux.

Un modèle B comprend un ensemble d'événements faisables qui s'exécutent instantanément. Les événements sont décrits par des substitutions généralisées particulières (affectation, ANY, BEGIN et SELECT). Seule la clause E = SELECT P THEN S END est utilisée dans cet article. Elle signifie que l'événement E est déclenché si la garde P est vraie, alors S est exécutée. La sémantique associée est une sémantique à base de traces d'événements.

Le contrôleur de dialogue est donc un ensemble d'événements gardés par un prédicat fort qui doit être respectée pour considérer que l'événement est émis et qui assure que l'opération associée a été réalisée. L'ensemble de ces événements définit un STE global qui décrit le système.

Dans le cas, d'une application décrite par plusieurs systèmes, notre démarche utilise la technique de raffinement pour introduire les événements des STE composés. Chaque système est alors décrit au fur et à mesure des raffinements de manière incrémentale. Ainsi à un niveau de raffinement correspond à une composition d'un ou plusieurs STE.

Pour notre étude de cas, la décomposition modulaire d'une application interactive en B suit le modèle d'architecture ARCH. Les modules domaine, adaptateur du domaine et présentation de chaque sous système sont décrits par des machines différentes alors ques les machines de la boîte à outils sont communes. Enfin le module contrôleur de dialogue qui centralise le dialogue des sous systèmes, est décrit par une machine abstraite et plusieurs machines raffinements.

4.2.2 Application à l'étude de cas

Dans la spécification ci-dessous, de la machine abstraite *BController*, l'événement *clickEF* est émis quand l'utilisateur clique sur le bouton de conversion de francs vers euros.

MODEL *BController*
  ...
EVENTS
clickEF =
  SELECT
    $b\_mouse\_y \in b\_frame\_y + b\_button\_EF\_y \cdots \wedge$
    $b\_buttonEF\_state = unpressed \wedge$
    $b\_mouse\_state = clicked \wedge$
    ...
  THEN
    $\cdots$ *CORPS de l'evenement*
  END
  ...

Cet événement ne peut être émis que si la garde est respectée (traduit par la substitution SELECT) et que si le corps est réalisé. Ici, il s'agit de vérifier que le curseur de la souris est positionné sur le bouton, que l'état de la souris est *cliqué*, et que le bouton n'est pas déjà *pressé*. Ensuite le corps de l'événement modifie les variables du domaine et de la présentation (absent dans l'exemple).

Le raffinement *BController_With_Counter*, présenté ci-dessous, complète le contrôleur de dialogue de l'étude de cas (convertisseur/compteur) par l'introduction de nouveaux événements. En comparaison avec la partie (STE), il compose les STE relatifs aux systèmes du convertisseur et du compteur. *BController_With_Counter* raffine *BController* en y ajoutant les informations du compteur ce qui se traduit par la présence d'un nouvel événement *initCounter* qui décrit l'action de l'utilisateur sur le bouton reset. Sa garde vérifie entre autre que la valeur du compteur doit être égal à trois pour que cet événement soit émis. Lorsque la valeur *v_counter* du compteur est inférieure à 3, les boutons de conversions sont actifs, la valeur du compteur est affichée et le bouton d'initialisation du compteur est désactivé. De même la garde de *clickEF* est raffinée, ainsi tant que $v\_counter < 3$, l'événement *clickEF* ne peut être émis (l'utilisateur ne peut plus convertir en francs ni en euros).

REFINEMENT *BController_With_Counter*
REFINES *BController*
...
EVENTS
clickEF =
SELECT
$b\_mouse\_y \in b\_frame\_y + b\_button\_EF\_y \cdots \wedge$
$b\_buttonEF\_state = unpressed \wedge$
$v\_counter < 3 \wedge$
...
THEN
...
$v\_counter := \cdots;$
IF $v\_counter = 3$ **THEN**
setButtonFrancEuroStateDesactivate;
setButtonResetStateActivate
ELSE
...
END
initCounter =
SELECT
$b\_mouse\_y \in b\_frame\_y + b\_button\_reset\_y \cdots \wedge$
$v\_counter = 3 \wedge$
...
THEN
$\cdots$ *CORPS de l'evenement*
END
...

La totalité de l'étude de cas a été développée en suivant cette démarche.

4.3 Une approche par composition : vérification sur modèle avec Promela/SPIN

L'outil Proméla/SPIN permet l'analyse de systèmes concurrents décrits en Promela (Programming Meta Language). Ce dernier est un langage de spécification dont la syntaxe est proche de celle du langage C avec quelques primitives de communication [Pro93]. Il décrit le comportement de chacun des processus d'un système, et les interactions entre ces processus. Chaque processus est explicitement décrit par une STE. La communication entre les processus se fait pas transmission de message via des canaux ou par variables partagées. La communication via les canaux peut être synchrone ou asynchrone. Un port *port* pouvant contenir n messages de type *typ* est déclaré par $Chan\ port = [n]\ of\ \{typ\}$. Si n vaut 0, alors la communication est synchrone. La synchronisation est réalisée à l'aide de *rendez-vous* sur les ports de communication. Un port de *rendez-vous* est déclaré en Promela par $Chan\ message = [0]\ of\ \{byte\}$.

L'émission de la valeur $\sim$ sur le port *message* est spécifiée par l'instruction *message !*$\sim$. La réception de la valeur $\sim$ sur le port *message* est spécifiée par l'instruction *message ?*$\sim$. L'émission et la réception de la valeur $\sim$ sur le port *message* se passent en même temps. Ce sont les seules actions qui doivent être exécutées en mode synchrone. Ainsi l'approche fondée sur Promela/SPIN permet l'expression :

- de la composition parallèle des processus dans le langage Promela ;
- de la communication synchrone entre les processus ;
- de l'atomicité d'une suite d'actions, ce qui empêche l'entrelacement d'autres actions avec cette suite d'actions ;
- et la vérification de propriétés temporelles.

Étant donné un système spécifié en Promela, Spin peut effectuer une simulation aléatoire ou dirigée de l'exécution du système et peut générer un programme C. Le programme peut être utilisé pour effectuer la vérification des propriétés du système. Ces propriétés sont exprimées par des formules de la logique temporelle linéaire (avec les opérateurs $\Box$ et $\Diamond$).

4.3.1 Promela et l'architecture logicielle

A l'image de ce qui a été présenté en B en section 4.2, nous nous sommes fondés sur le modèle ARCH. Les éléments de la présentation et ceux du domaine évoluent en parallèle et d'une manière asynchrone. La synchronisation entre la présentation, contrôleur de dialogue et domaine et ce même contrôleur de dialogue contrôle l'évolution du système de manière à éviter les comportements incorrects du système. L'utilisateur n'étant pas modélisé dans notre cas, nous distinguons trois classes de processus, les processus

- correspondants aux éléments de la présentation. Un processus pour chaque objet de la présentation ;
- correspondants aux éléments du domaine ;
- décrivant le contrôleur de dialogue.

Chaque processus est modélisé par un STE, et communique en parallèle par l'intermédiaire de variables partagées et envois de messages synchrones.

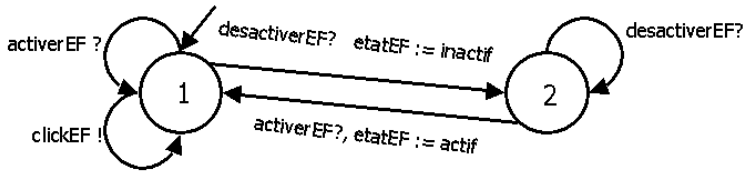

FIG. 2 – Automate du bouton de conversion des euros vers francs

4.3.2 Application à l'étude de cas

Chaque élément de la présentation est représenté par un processus qui interagit avec le contrôleur de dialogue. Il envoie des messages au contrôleur de dialogue et réagit aux messages retour. La Figure 2 représente l'automate qui décrit le comportement du bouton de conversion des euros vers les francs grâce au STE *ButtonEF*. Initialement l'état du bouton, défini par la variable *etatEF* est *actif*. Quand le message envoyé par le contrôleur sur le port *désactiverEF* est reçu, le bouton change d'état. Il ne revient à l'état *actif* que s'il reçoit un message sur le port *activerEF*. Le bouton peut envoyer un message sur le port *clickEF* au contrôleur pour activer la conversion de la valeur saisie des euros vers les francs. La traduction dans le langage Promela est décrite ci-dessous.

```
proctype buttonEF()
{
etatEF = actif;
t1 : if
        :: desactiverEF?1 → atomic{etatEF = inactif; goto t2;}
        :: activerEF?1 → goto t1;
        :: atomic{clickEF!1; goto t1;}
   fi;
t2 : if
        :: activerEF?1 → atomic{etatEF = actif; goto t1;}
        :: atomic{desactiverEF?1 → goto t2;}
   fi;
}
```

Le processus du contrôleur de dialogue s'occupe du dialogue entre les éléments de la présentation et ceux du domaine (Fig 3). Sa présentation en Promela est donnée ci-dessous. Nous ne la commenterons pas pour

des raisons d'espace dans cet article, mais elle est donnée pour montrer la présence des communications entre les différents processus via des ports communs aux STE *ButtonEF* et *Controller*.

```
proctype controller()
{
t1 : if
        :: cp == 0 → atomic{activerEF!1; activerFE!1;
                desactiverR!1; goto t2;}
        :: atomic{(cp < 3&cp > 0) → gotot2;}
        :: cp == 0 → atomic{activerR!1; desactiverEF!1;
                desactiverFE!1; goto t3;}
   fi
t2 : valsaisi?1 → goto t4;
t3 : clickR?1 → atomic{rz!1; affichecpt!1; goto t1};
t4 : if
        :: clickEF?1 → {convEF!1; inc!1; affichevals!1;
                affichecpt!1; goto t1};
        :: clickFE?1 → {convFE!1; inc!1; affichevals!1;
                affichecpt!1; goto t1};
   fi
}
```

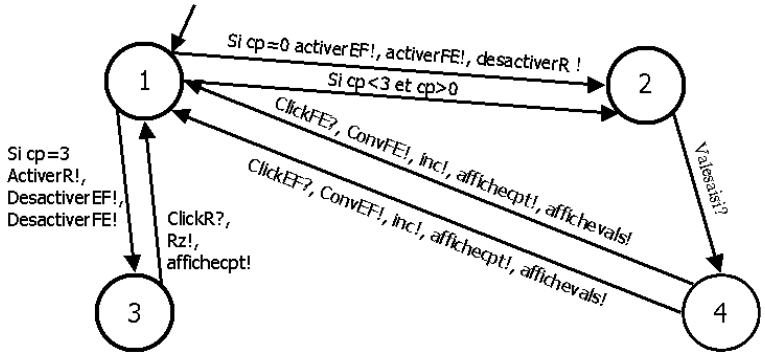

FIG. 3 – Automate du contrôleur de dialogue

L'application est obtenue par composition parallèle de l'ensemble des processus de la présentation, du domaine et du contrôleur de dialogue. Les processus s'exécutent d'une manière asynchrone sur l'ensemble des actions sauf pour l'envoi et la réception des messages. Ces derniers s'effectuent d'une manière synchrone.

4.4 Comparaison de la modélisation

La démarche, proposée avec la méthode B, a montré la modélisation et la composition de STE. Cette modélisation d'un système (STE) est rendue possible par l'intermédiaire de l'utilisation de la méthode B dans sa version *B Evenementiel*. En effet les relations de transition sont décrites par les gardes des opérations événements. Quand à la composition, elle a été obtenue par l'utilisation du raffinement qui autorise l'apparition de nouveaux événements provenant éventuellement d'autres systèmes de transitions.

De plus, le raffinement a permis une décomposition de la complexité des obligations preuves et donc de la preuve. Le prouveur de l'outil *Atelier B* peut décharger plus facilement ces obligations de preuve. La méthode B *ne manipule pas explicitement les STE*. Le concepteur ne décrit que les événements en donnant les conditions d'apparition de l'événement (garde) et l'action instantanée réalisée lorsque la garde est vraie et que l'événement est présent.

En revanche, le langage Promela permet de définir des processus qui s'exécutent en parallèle et qui se synchronisent sur des ports d'émission et de réception de message. Des processus complexes peuvent être conçus par composition d'autres processus. Chaque processus est décrit par un STE et la composition revient à décrire une composition de STE. Dans ce cas, et à l'opposé de la méthode B, Promela/SPIN *manipule explicitement les STE*. Le concepteur doit connaître les différents états ainsi que les synchronisations entre les envois et les réceptions de messages.

Comme pour B, l'approche proposée par Promela/SPIN autorise une conception incrémentale du système. L'incrémentalité est assurée par la composition qui enrichit le système global au fur et à mesure que les compositions sont réalisées.

En conclusion, les deux techniques ont permis la conception totale, mais de manières différentes de l'étude de cas. Il reste à étudier l'hétérogénéité qui résulterait d'une situation ou une partie de l'étude de cas serait représentée en B et l'autre en Promela/SPIN. Dans ce cas, comment définirait-on la composition ?

4.5 Vérification et preuve

Des propriétés de robustesse et d'atteignabilité ont été exprimées et vérifiées suivant les approches B et Promela/Spin. Il s'agit de propriétés qui assurent un fonctionnement correct du système. Ces preuves et vérification n'ont pas été abordées dans cet article par manque de place. Nous nous sommes intéressés à la modélisation formelle du système plutôt qu'aux propriétés et à leur vérification. Notre approche est une approche orientées modèles.

En B, les propriétés sont exprimées dans les clauses *INVARIANT* et *ASSERTIONS* présentes dans B et associées à la modélisation du système interactif. Parmi ces propriétés, nous avons exprimé, entre autres, les propriétés d'observabilité, d'insistence, de non blocage et d'équité. Notons que le processus de preuve de B n'est pas complètement automatique mais est supporté par un outil de preuve automatique avec lequel le concepteur interagit, mais l'utilisation de cette technique ne se limite pas à la modélisation de systèmes finis.

En Promela/SPIN, les mêmes propriétés qu'en B ont été exprimées. On retrouve les propriétés d'observabilité, d'insistence, de non blocage et d'équité. Le processus de preuve est complètement automatique vu que Promela/SPIN effectue de la vérification sur modèle. Cependant, il n'est possible de modéliser que les systèmes finis sur lesquels des parcours exhaustifs peuvent être décrits.

Enfin, cette comparaison ne peut être complète sans aborder la complexité des preuves. En effet, le raffinement permet de répartir les preuves. Les propriétés sont vérifiées une seule fois au niveau où elles sont introduites dans la modélisation. Les invariants de collage assurent que ces propriétés sont toujours vérifiées dans les raffinements futurs. Cela tient à la méthode B. En promela/SPIN, cela n'est pas le cas. A chaque composition, toutes les propriétés sont prouvées une nouvelles fois. L'augmentation de la complexité de la description du système après composition, fait que la preuve des propriétés devient plus compliquée également. Cependant, la possibilité d'introduire des raffinements dans les techniques fondées sur les automates et leur composition permet d'alléger cette preuve.

Une solution aux problèmes évoqués, serait de combiner, judicieusement, la composition au sens des automates de Promela/SPIN avec le raffinement au sens de B. En d'autres termes à l'image de B, définir les décompositions d'automates de Promela/Spin comme des raffinements préservant les propriétés des automates décomposés.

5 Conclusion et perspectives

L'utilisation de techniques formelles au travers de l'étude de cas interactive et concurrente présentée dans cet article a démontré la possibilité de concevoir des modèles formels sur lesquels des propriétés formellement exprimées pouvaient être vérifiées. Cette étude a mis en œuvre deux techniques formelles différentes : d'une part B avec le raffinement et la preuve et d'autre part Promela/SPIN avec la logique temporelle et la vérification sur modèle. En plus des aspects liés à la conception, ces deux techniques ont permis la vérification de propriétés de robustesse et d'atteignabilité ainsi que la prise en compte des notations semi-formelles issues du domaine d'application (les IHM) comme la notation d'architecture ARCH.

Par ailleurs, cette étude a permis de réaliser la conception de systèmes par composition :

- la méthode B a permis l'introduction de nouveaux composants grâce au raffinement en faisant apparaître de nouveaux événements dus à la composition de nouveaux systèmes de transitions. La manipulation des systèmes de transitions n'est pas explicite ;
- Promela/SPIN a permis la composition de différents systèmes de transitions conçus séparément en utilisant des communications entre les différents automates via des ports de communication. La manipulation des systèmes de transitions est explicite.

Ces travaux ne sont pas achevés. De nombreuses perspectives devraient être étudiées à l'avenir. Nous citerons pèle-mêle la validation de tâches (qui permettra la prise en compte des utilisateurs et des notations de la section 2.1), la vérification de propriétés issues des études de l'ergonomie, la retro-conception et la

prise en compte de l'hétérogénéité des techniques et des conceptions. Un autre point nous semble également important, il concerne la composition au niveau des propriétés.

Références

[AAGJ98a] Yamine Aït-Ameur, Patrick Girard, and Francis Jambon. A uniform approach for the specification and design of interactive systems : the b method. In Panos Markopoulos and Peter Johnson, editors, *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, volume Proceedings, pages 333–352, Abingdon, UK, 1998.

[AAGJ98b] Yamine Aït-Ameur, Patrick Girard, and Francis Jambon. Using the b formal approach for incremental specification design of interactive systems. In Stéphane Chatty and Prasun Dewan, editors, *Engineering for Human-Computer Interaction*, volume 22, pages 91–108. Kluwer Academic Publishers, 1998.

[Abr96a] J-R Abrial. *The B Book : Assigning Programs to Meanings.* Cambridge University Press, 1996.

[Abr96b] J-R Abrial. Extending b without changing it (for developing distributed systems). In H Habrias, editor, *First B Conference, Putting Into Pratice Methods and Tools for Information System Design*, page 21, Nantes, France, 1996.

[AWM95] Gregory D. Abowd, Hung-Ming Wang, and Andrew F. Monk. A formal technique for automated dialogue development. In Gary M. Olson and Sue Schuon, editors, *DIS'95, Design of Interactive Systems*, pages 219–226, Ann Arbor, Michigan, 1995. ACM Press.

[BACL95] P. Bumbilis, P.S.C. Alencar, D.D. Cowan, and C.J.P Lucena. Combining formal techniques and prototyping in user interface construction and verification. In Philippe Palanque and Rémi Bastide, editors, *2nd Workshop on Design, Specification and Verification of Interactive Systems (DSVIS)*, Springer Computer Science, pages 174–192, Bonas, 1995. Springer-Verlag.

[BHH+88] L. Bass, E. Hardy, K. Hoyt, R. Little, and R. Seacord. The arch model : Seeheim revisited, the serpent run time architecture and dialog model. Technical Report CMU/SEI-88-TR-6, Carnegie Melon University, 1988.

[BPR+91] l. Bass, R. Pellegrino, S. Reed, S. Sheppard, and M. Szezur. The arch model : Seeheim revisited. In *User Interface Developper's Workshop*, 1991.

[Bru97] Philippe Brun. *XTL : a temporal logic for the formal development of interactive systems*, pages 121–139. Springer-Verlag, 1997.

[CES86] E M Clarke, E A Emerson, and A P Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 2(8) :244–263, 1986.

[Cle97] ClearSy. Atelier b - version 3.5, 1997.

[CMAA01] K. Chebieb, D. Mansour, and Y. Ait-Ameur. Analyse et Evaluation de Propriétés dans les IHM. In *Proceedings of the 7th International Symposium on Programming and Systems ISPS 2001)*, pages 241–252, 2001.

[Cou87] J. Coutaz. Pac, an implementation model for the user interface. In *IFIP TC13 Human-Computer Interaction (INTERACT'87)*, pages 431–436, Stuttgart, 1987. North-Holland.

[DFAB93] Alan Dix, Janet Finlay, Gregory Abowd, and Rusell Beale. *Human-Computer Interaction.* Prentice Hall, 1993.

[DH93a] D. Duke and M. D. Harrison. Abstract Interaction Objects. In *Proceedings of Eurographics conference and computer graphics forum*, volume 12, pages 25–36, 1993.

[DH93b] David J. Duke and Michael D. Harrison. Abstract interaction objects. *Computer Graphics Forum*, 12(3) :25–36, 1993.

[DH95] D Duke and M. D. Harrison. Event model of human-system interaction. *IEEE Software*, 1(10) :3–10, 1995.

[Fek96] Jean-Daniel Fekete. *Un modèle multicouche pour la construction d'applications graphiques interactives.* Doctorat d'université (phd thesis), Université Paris-Sud, 1996.

[Gau95] M. C Gaudel. Formal specification techniques for interactive systems. In Philippe Palanque and Rémi Bastide, editors, *2nd Workshop on Design, Specification and Verification of Interactive Systems (DSVIS)*, Springer Computer Science, pages 21–26, Bonas, 1995. Springer-Verlag.

[GEM94] Phil Gray, David England, and Steve McGowan. Xuan : Enhancing the uan to capture temporal relation among actions. Department research report IS-94-02, Department of Computing Science, University of Glasgow, February 1994. Modifications par rapport à UAN : - Aspect symétrique USER/SYSTEM - Contraintes temporelles - Paramètrisation des tâches - Pré et Post-conditions.

[GP94] M. Gordon and A. Pitts. The hol logic and system. Technical report, 1994 1994. Condensed from Introduction to HOL by Gordon and Melham.

[Gui95] Laurent Guittet. *Contribution à l'Ingénierie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO.* Doctorat d'université (phd thesis), Université de Poitiers, 1995.

[Har87] D. Harel. Statecharts : a visual formalism for complex systems. *Science of Computer Programming*, 8(3) :231–274, 1987.

[HH93] Deborah Hix and H Rex Hartson. *Developping user interfaces : Ensuring usability through product & process.* Wiley professional computing. John Wiley & Sons, inc., Newyork, USA, 1993.

[HT] Michael D. Harrison and Harold W. Thimbleby, editors. *Formal Methods in Human-Computer Interaction.* Human-Computer Interaction. Cambridge University Press, Cambridge.

[HT90] M. Harrison and H. Thimbleby. *Formal Methods in Human-Computer Interaction.* Series on Human computer interaction. Cambridge University Press, 1990.

[Jac82] R.J.K Jacob. Using formal specification in the design of human-computer interface. In *Human Factors in computing systems*, pages 315–321, 1982.

[Mar86] L S Marshall. *A Formal Description Method for User Interface.* Ph.d thesis, University of Manchester, 1986.

[McM92] K McMillian. The smv system. Technical report, Carnegie Mellon University, 1992.

[Pal92] Philippe Palanque. *Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur.* Doctorat d'université (phd thesis), Université de Toulouse I, 1992.

[PF92] Fabio Paternò and Giorgio P. Faconti. *On the LOTOS use to describe graphical interaction*, pages 155–173. Cambridge University Press, 1992.

[Pfa85] Günther E Pfaff, editor. *User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim.* Eurographic Seminars. Springer-Verlag, Berlin, 1985.

[Pro93] Spin - version 4.0.1, 1993.

[Roc98] Pierre Roche. *Modélisation et validation d'interface homme-machine.* Doctorat d'université (phd thesis), École Nationale Supérieure de l'Aéronautique et de l'Espace, 1998.

[SPG89] D L Scapin and C Pierret-Golbreich. Mad : Une méthode analytique de description des tâches. In *Colloque sur l'Ingénierie des Interfaces Homme-Machine (IHM'89)*, pages 131–148, Sophia-Antipolis, France, 1989.

[Sys84] ISO Information Processing Systems. Definition of the temporal ordering specification language lotos. Technical Report TC 97/16 N1987, ISO, 1984.

[Was81] A Waserman. User software engineering and the design of interactive systems. In *5th IEEE International Conference on Software Engineering*, pages 387–393. IEEE society press, 1981.