Vers la formalisation de propriétés ergonomiques de systèmes interactifs

Laya Madani, Ahmed M'Hiri, Sophie Dupuy-Chessa, Ioannis Parissis

Vers la formalisation de propriétés ergonomiques de systèmes interactifs

Laya Madani[1], Ahmed M'hiri[1], Sophie Dupuy-Chessa[1]
Ioannis Parissis[2]

Universités de Grenoble
[1]Laboratoire d'Informatique de Grenoble
BP 53 38041 Grenoble Cedex 9
[2]Laboratoire de Conception et d'Intégration des Systèmes
BP54 26902 Valence Cedex 09
{laya.madani, sophie.dupuy-chessa, ioannis.parissis}@imag.fr

Résumé

Les applications interactives sont aujourd'hui présentes dans plusieurs domaines et leur utilisation au sein de systèmes critiques est de plus en plus fréquente. Leur puissance ne cesse d'augmenter ainsi que leur complexité ce qui accroît le risque que des fautes soient introduites pendant les différentes étapes de leur développement. Leur correction devient ainsi un enjeu important et leur développement requiert une validation rigoureuse. Outre le besoin de correction fonctionnelle, les exigences attendues d'une application interactive s'expriment en termes d'utilisabilité, définie au moyen de propriétés ergonomiques. La vérification de ces dernières n'est pas une tâche facile, leur définition étant souvent trop informelle ou bien difficilement exprimable en des termes exploitables par des ingénieurs de développement ou de validation. Dans cet article, nous présentons la formalisation de certaines propriétés ergonomiques qui pourront ultérieurement être validées automatiquement. Pour cela, nous nous basons sur une notation courante dans le domaine de l'interaction homme-machine, les arbres des tâches, déjà utilisée dans le cadre de nos travaux antérieurs pour la génération automatique de tests. Nous identifions les insuffisances du modèle sousjacent à cette génération en vue de la prise en compte de propriétés ergonomiques et nous exhibons une extension de ce dernier permettant d'envisager le test de certaines d'entre elles.

1 Introduction

Les applications interactives sont aujourd'hui omniprésentes dans notre vie quotidienne, que ce soit lors de l'utilisation de systèmes commerciaux ou bien de logiciels embarqués sur les nombreux dispositifs portables et communiquants dont nous dépendons de manière croissante, et leur bon fonctionnement est un facteur essentiel de l'économie. Leur utilisation est également fréquente dans des secteurs critiques (transports, aviation ...) ce qui accentue les exigences de qualité qui pèsent sur elles. La multimodalité (capacité d'interagir au moyen de plusieurs modalités, telles que le geste, la voix ...) contribue également à la complexification de ces systèmes, ce qui se traduit par l'augmentation de la taille et de la complexité de leur code et par une augmentation du risque d'introduction de fautes pendant le développement. En conséquence, la validation et la vérification des systèmes interactifs est une activité d'une haute importance et un domaine de recherche actif, qui connaît une évolution considérable ces dernières années.

Les exigences attendues d'un système interactif s'expriment souvent en termes d'*utilisabilité*, qualité qui caractérise sa capacité de permettre à l'utilisateur d'atteindre ses objectifs avec efficacité, en tout confort et sécurité. Elle est principalement déterminée par des propriétés ergonomiques, représentant, d'après [1], la souplesse (degré des possibilités de choix offertes) et la robustesse (capacité de prévenir les erreurs et d'augmenter les chances de succès de l'utilisateur) de l'interaction. Ces propriétés ne sont pas toujours facilement formalisables (souvent elles ne le sont pas du tout), entre autres parce qu'elles peuvent comporter une part d'appréciation subjective. Leur évaluation sur une application donnée se fait, en général, par des utilisateurs humains. Dans [10] les techniques d'évaluation et de validation de systèmes interactifs sont classées en cinq catégories : l'évaluation expérimentale, l'inspection statique, l'interrogatoire, l'analyse de modèles et la simulation. L'existence de modèles, parfois formels, dans les deux dernières catégories rend possible le recours à la validation automatique, tandis que les trois premières comportent des techniques difficilement automatisables.

Des méthodes de vérification automatique ont été proposées dans le passé, notamment s'appuyant sur des modèles développés dans l'étape de conception (après l'analyse des besoins et avant la réalisation). L'utilisation du langage LOTOS a été suggérée pour construire un modèle de l'application sur lequel il est possible de vérifier des propriétés comme l'atteignabilité, la réactivité, la conformité [18][19]. La notation CTT (ConcurrTaskTree) [17] a été proposée comme une extension de cette approche, pour prendre en compte la dynamique de l'interface. L'approche ICO ("Interactive Cooperative Ob-

jects") [4] adopte une démarche de modélisation similaire mais en s'appuyant sur une spécification formelle dans un langage objet où le comportement de l'application est décrit par un réseau de Petri. Dans [2, 3], il est proposé d'utiliser B événementiel pour valider les tâches utilisateur à partir d'un modèle CTT. L'utilisation du langage Lustre a été également proposée pour la spécification et la validation d'interfaces à fenêtres par model-checking [7]. Mais l'adoption de ces approches par la communauté semble difficile à envisager dans un avenir proche, du fait de la nécessité de spécifier la totalité de l'application - opération s'avérant très souvent hors d'atteinte pour les ingénieurs - ainsi qu'à cause de la complexité des preuves mises en oeuvre. Par ailleurs, les formalismes utilisés ne sont pas courants dans la conception des applications interactives.

Des approches de test automatisé ont également été proposées s'appuyant sur des modèles de l'interaction, comme, par exemple, l'utilisation de modèles de Markov pour évaluer l'utilisabilité dans la phase de conception [21]. On peut également citer la méthode de [12] dont le principe est de générer l'ensemble complet de séquences d'interaction à partir de l'analyse de modèles de tâches simplifiés (sans récursivité ni boucles).

Nous avons récemment proposé [14, 15] une démarche pour automatiser le test des applications interactives, adoptant une notation couramment utilisée dans ce domaine, les arbres des tâches (et plus précisément, la notation CTT [17]). En la dotant d'une sémantique formelle appropriée, nous avons montré qu'elle peut servir de formalisme de base permettant l'automatisation de la génération de données de test. Les scénarios de test ainsi décrits et, par conséquent, les données de test produites, sont adaptés à la validation de propriétés fonctionnelles, portant sur les aspects de contrôle de l'application. Dans cet article, nous poursuivons cette démarche en étudiant la possibilité de formaliser des propriétés ergonomiques liées à l'utilisabilité pour pouvoir les valider ultérieurement.

La principale question que nous nous posons est celle de l'adéquation, pour ce type de validation, du modèle utilisé pour la description des scénarios de test. Après une étude des principales propriétés ergonomiques, nous identifions des enrichissements nécessaires et nous définissons un nouveau modèle, permettant l'expression de certaines d'entre elles. Pour d'autres propriétés, cette quête s'avère plus difficile, voire impossible.

Le paragraphe 2 rappelle les principes de notre démarche de test, tandis que le paragraphe 3 définit le cadre de l'étude des propriétés ergonomiques. Notre contribution est présentée dans le paragraphe 4 où l'on définit un modèle formel, obtenu par extension du modèle existant, permettant de prendre en compte les propriétés ergonomiques.

2 Démarche de test

2.1 Arbres de tâches

Dans la suite de cet article, nous illustrerons nos propos principalement sur une application interactive, Memo [6], qui permet d'annoter des localisations physiques avec des notes ("post-it") digitales. Lorsqu'une note a été placée à un endroit, elle peut être lue/portée/supprimée par d'autres utilisateurs. L'utilisateur de Memo est équipé d'un GPS et d'un magnétomètre pour permettre le calcul de la localisation et de l'orientation de ce dernier. Il porte également un casque dont la semi-transparence permet la fusion entre les données électroniques (les notes digitales) et l'environnement physique. Memo, comme c'est souvent le cas pour les applications interactives, a été spécifié en élaborant un arbre de tâches.

Les arbres de tâches, sont des représentations hiérarchiques [8] : une tâche est composée de sous-tâches liées par des opérateurs temporels. Cela signifie que le modèle de tâches fournit des informations sur les sous-tâches qui doivent être exécutées afin d'accomplir une autre tâche plus complexe, ainsi que des informations sur le comportement de système interactif. Nous utilisons la notation bien connue CTT (*ConcurTaskTrees*) [16] qui distingue quatre types de tâches. Une **tâche utilisateur** est une activité cognitive sans interaction avec le système (la réflexion pour résoudre un problème, lecture d'un message...). Une **tâche application** est une action du système, comme l'affichage du résultat d'une requête. Une **tâche interactive** correspond à une action de l'utilisateur avec un feedback immédiat du système, comme l'édition d'un document. Enfin, une **tâche abstraite** est composée d'autres tâches liées par des opérateurs temporels : par exemple, il existe un opérateur d'activation noté $>>$ qui spécifie qu'une tâche est activée par l'exécution d'une autre tâche.

Dans Memo, trois tâches principales sont définies : (1) le changement d'orientation et de localisation de l'utilisateur mobile ; le système affiche sur le casque les notes visibles selon la position et l'orientation courantes (2) la manipulation d'une note (récupérer, placer, supprimer) et (3) la sortie du système. L'utilisateur mobile peut ainsi récupérer une note qu'il portera pendant son déplacement. Il ne peut porter qu'une note à la fois. Enfin, il peut supprimer une note portée ou visible dans son casque. La figure 1 présente l'arbre des tâches de l'application "Memo" dans la notation CTT. Cet arbre montre que l'utilisateur peut utiliser l'application itérativement (opérateur d'itération *) et peut être interrompu (opérateur de désactivation $[>$) par la tâche "exit". La tâche "use memo system" est un choix (opérateur []) entre

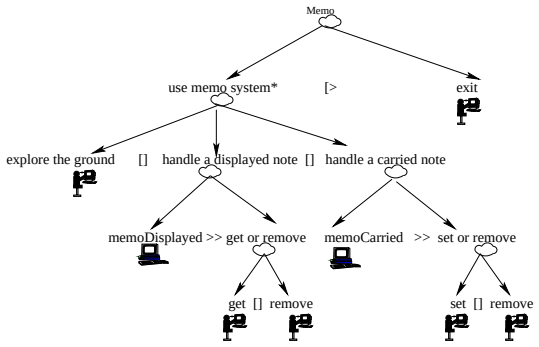

FIG. 1 – Arbre de tâches de Memo

les trois tâches suivantes : découvrir le terrain ("explore the ground"), la manipulation d'une note visible ("handle a displayed note") ou la manipulation d'une note portée ("handle a carried note"). Si le système affiche une note ("memoDisplayed"), l'utilisateur peut (opérateur d'activation $>>$) récupérer ou supprimer cette note. Si l'utilisateur est en train de porter une note ("memoCarried"), l'utilisateur peut la placer ou la supprimer. Cette spécification va être la base de notre travail de validation de systèmes interactifs. Elle va nous permettre d'en automatiser partiellement le test.

2.2 Test à partir d'arbres de tâches

L'arbre de tâches que nous venons de présenter sert de modèle de départ à la génération de tests. Plus précisément, à partir de ce modèle, nous construisons un nouveau modèle formel, décrit dans la suite.

Modèle formel pour la génération de test

Une tâche CTT peut être une tâche utilisateur, une tâche abstraite, une tâche application ou une tâche interactive.

Les tâches utilisateur ne sont pas intéressantes du point de vue de la génération de tests, car elles ne correspondent à aucune interaction avec le système (ni action ni réaction).

Nous assimilons une tâche application o à une machine d'états élémentaire à deux états, dont la seule transition consiste à passer de l'état initial à l'état final en émettant une sortie. Formellement, elle est modélisée par la machine élémentaire $M_o = (Q_o, qi_o, qf_o, I_o, O_o, trans_o)$ où :

- $Q_o = \{qi_o, qf_o\}$ est l'ensemble d'états ; qi_O , qf_O sont respectivement l'état initial et l'état final de M_O.

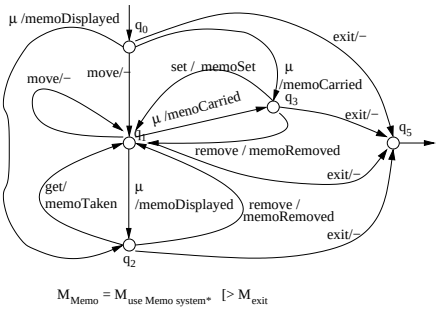

FIG. 2 – Automate de Memo

- $I_o = \{\mu\}$ est l'ensemble d'entrées de la tâche, assimilé à l'entrée vide μ (l'utilisateur ne procède à aucune action).
- $O_o = \{o\}$ est l'ensemble de sorties (contenant un seul élément : la sortie de la tâche).
- $trans_o = \{qi_o \xrightarrow{\mu/o} qf_o\}$ est un ensemble de transitions à un seul élément : $qi_o \xrightarrow{\mu/o} qf_o$ dénote la transition de l'état qi_o vers l'état qf_o avec l'entrée vide μ est la sortie o.

La principale hypothèse de notre approche concerne les tâches interactives : nous supposons qu'elles sont modélisées par des machines d'états finis à E/S fournies en même temps que l'arbre des tâches.

Formellement, pour une tâche interactive T, on définit la machine d'états finis à E/S $M_T = (Q_T, qi_T, qf_T, I_T, O_T, trans_T)$ où :

- Q_T est un ensemble d'états.
- qi_T est l'état initial. C'est un état source.
- qf_T est l'état final. C'est un état puits.
- I_T est l'ensemble des entrées de la tâche T.
- O_T est l'ensemble des sorties de la tâche T.
- $trans_T \subseteq Q_T \times (I_T \cup \{\mu\}) \times O_T \times Q_T$ est l'ensemble de transitions de la tâche T. Si $(q_T, a, b, s_T) \in trans_T$, on écrit $q_T \xrightarrow{a/b} s_T$. En omettant l'entrée et la sortie de la transition, on note : $q_T \xrightarrow{c} p_T$ pour $c = a/b$.

Enfin, une tâche abstraite est composée de sous-tâches liées par des opérateurs CTT. Sa machine à E/S associée résulte de la combinaison des machines de ses sous-tâches [14, 13] (cf. 2 pour l'exemple de Memo).

Nous avons également proposé d'enrichir l'arbre de tâches avec des probabilités d'occurrence [15], ce qui permet de spécifier des profils opérationnels et de générer des tests conformes à ces derniers.

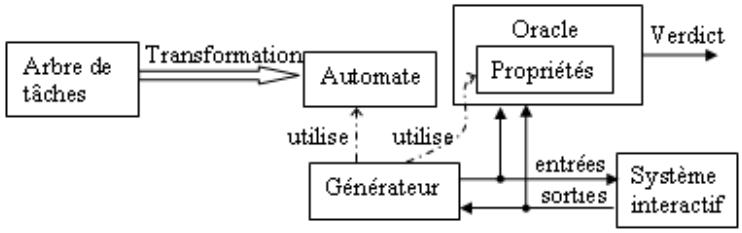

FIG. 3 – Processus du test de systèmes interactif

Automatisation du test

A partir de l'automate représentant l'arbre des tâches, la génération de tests pour l'application interactive peut s'effectuer "à la volée", par simulation. Il est ainsi possible de décrire des scénarios abstraits d'interaction, sous forme d'arbres de tâches et de profils opérationnels, et d'observer le comportement de l'application interactive en vérifiant, par exemple, sa conformité aux scénarios, comme le montrent des expérimentations que nous avons menées avec l'outil Lutess [5]. De même, on peut envisager de guider automatiquement la génération de tests vers des violations potentielles des propriétés de l'application, à condition que ces dernières puissent s'exprimer dans le modèle formel considéré (voir la figure 3).

La vérification de propriétés ergonomiques pourrait être également envisagée si leur expression était possible (ou bien si des scénarios permettant leur validation étaient exprimables) dans ce même modèle formel. Cette problématique est étudiée dans la suite de l'article.

3 Propriétés ergonomiques

En interaction homme-machine, l'ergonomie caractérise l'utilisabilité d'un système interactif. Elle définit des recommandations qui doivent permettre de rendre le système plus facilement utilisable par les utilisateurs. De nombreux catalogues de prescriptions ergonomiques existent. Certains sont spécifiques à un type de système interactif comme les sites web [11] ou les interfaces de réalité virtuelle [11] ; d'autres sont plus globaux et traitent du problème de l'utilisabilité quelque soit le système [9, 20]. Le niveau d'abstraction de ces propositions varie aussi : ainsi dans [11], des recommandations très concrètes telles que "évitez les phrases trop courtes" sont données alors que [9] reste abstrait avec des propriétés comme l'observabilité ou l'atteignabilité. Dans

[20] différents niveaux d'abstraction sont proposés avec des critères tels que le guidage (moyens mis en oeuvre pour conseiller, orienter, informer et conduire l'utilisateur) qui sont ensuite raffinés par des recommandations concrètes telles que le groupement par localisation des items appartenant à une famille de concepts.

Dans notre travail de validation de systèmes interactifs, nous avons choisi de nous baser sur des règles 1) de haut niveau, utilisables pour tous types d'interfaces pour rendre nos propositions les plus larges possibles ; 2) exprimées de préférence du point de vue du système car elles sont plus facilement exprimables indépendamment de l'utilisateur. Les propriétés ergonomiques que nous cherchons à valider sont donc celles de [9].

Dans cette approche, l'utilisabilité d'un système interactif caractérise sa capacité de permettre à l'utilisateur d'atteindre ses objectifs (obtenir un résultat correct ayant une qualité donnée) avec efficacité, en tout confort et sécurité. Elle est déterminée par deux facteurs de qualité qui sont définis par plusieurs propriétés ergonomiques :

- La *souplesse* exprime l'éventail des choix pour l'utilisateur et le système. Par exemple, offrir la possibilté de défaire une action est un élément important pour la souplesse du système. La souplesse comprend, entre autres, 1) la propriété d'atteignabilité ; 2) celle de non-préemption et 3) l'interaction multi-filaire. L'atteignabilité est la capacité du système à offrir à l'utilisateur la possibilité de naviguer dans l'ensemble des états observables du système. Ainsi dans l'application Memo, l'atteignabilité est respectée si l'utilisateur peut réaliser toutes les tâches prévues i.e. couvrir le terrain et manipuler un mémo. La non-préemption qui note la capacité pour l'utilisateur d'atteindre directement son prochain but souhaité est vérifiée, par exemple, si l'utilisateur souhaite quitter l'application et que cette sortie est réalisable en une seule action sur l'interface. La souplesse comprend aussi l'interaction multi-filaire qui définit la capacité du système à permettre la réalisation de plusieurs tâches (par exemple, dans Memo, la capacité de gérer plusieurs notes virtuelles à la fois).
- La *robustesse* vise à la prévention des erreurs, l'augmentation des chances de succès de l'utilisateur. La robustesse est définie par un ensemble de propriétés telles que l'observabilité c'est-à-dire la capacité du système à rendre perceptible son état pertinent à l'utilisateur. Par exemple, l'utilisateur doit pouvoir comprendre si la note virtuelle qu'il manipule est visible ou portée. L'observabilité implique par conséquent la capacité pour l'utilisateur d'évaluer l'état du système. La robustesse se caractérise aussi par la tolérance du rythme qui laisse le choix à

l'utilisateur plutôt qu'au système. Ainsi dans Memo, c'est toujours l'utilisateur qui décide de supprimer une note virtuelle et non pas le système au bout d'un certain délai.

Certaines propriétés ergonomiques orientées système peuvent dépendre du point de vue de l'utilisateur : en effet, lui seul peut évaluer la pertinence d'un état (cf. observabilité). Ce type de propriétés ne peut être formalisé et validé qu'une fois appliqué à un système et avec l'intervention des utilisateurs pour expliciter leurs attentes. Nous ne nous intéressons donc pas à ces règles et nous nous focalisons sur des propriétés portant sur des caractéristiques de l'interface indépendantes des utilisateurs. Parmi ces dernières, nous nous limitons à celles qui peuvent être vérifiées à l'aide d'automates.

4 Formalisation des propriétés ergonomiques

4.1 Insuffisances du modèle formel actuel

Pour formaliser les propriétés ergonomiques, nous avons besoin d'enrichir le modèle décrit dans le paragraphe 2.2. En effet, l'atteignabilité est définie par rapport à des *états observables* du système. Un état observable, du point de vue de l'utilisateur, est un état de l'interface du système. Le modèle formel actuel (machine à états finis), ne fournit pas d'information sur l'interface ce qui empêche de faire la correspondance entre ses états et les états observables.

Par ailleurs, un état du modèle peut correspondre à plusieurs états observables du système. Par exemple, dans Memo (c.f. figure 2), le nombre global de notes dans l'espace n'est pas le même chaque fois que l'on passe par l'état q_1 et dépend du nombre de "get", "set" ou "remove" effectués. De même, dans le cas de "l'interaction multifilaire" qui caractérise la capacité du système à réaliser plusieurs tâches, les tâches sont assimilées à celles définies dans le modèle d'interaction CTT. Dans le modèle formel correspondant (machine d'états finis) il n'y a aucune information qui indique qu'une transition correspond à une action de telle ou telle tâche abstraite (du modèle CTT). Il est donc difficile de vérifier sur ce modèle si on peut, ou non, exécuter plusieurs tâches. Nous en déduisons qu'une information concernant la *tâche d'origine* doit être conservée dans le modèle.

En conséquence, le nouveau modèle formel proposé contient deux différences par rapport à la machine d'états finis définie au paragraphe 2.2.

- A chaque transition est associé l'*ensemble de tâches* (abstraites, application ou interactives) qui peuvent activer la transition. De cette manière l'information contenue dans l'arbre de tâches reliant l'action élémentaire à une tâche plus abstraite est gardée. Par exemple, dans le

cas d'une interaction multifilaire comportant l'édition de deux documents d'une manière entrelacée, et qui permet l'exécution de la tâche "sauvegarder", il est intéressant d'indiquer quel document est sauvegardé.

- *Un vecteur d'état* est ajouté décrivant l'état de l'interface. Chaque variable de ce vecteur correspond à un composant de l'interface de l'application (bouton, zone de texte,...). La valeur de ce vecteur d'état varie chaque fois que l'interface du système change. Chaque valeur du vecteur d'état est alors un état observable du système. Par exemple, dans Memo, si l'évolution du nombre total de notes doit être observable (pour une propriété comme l'atteignabilité), alors il faudra avoir une variable d'état indiquant ce nombre.

Dans la suite, nous présentons le modèle utilisé pour exprimer les propriétés (paragraphe 4.2), puis nous montrons comment des propriétés peuvent être formalisées sur ce modèle (c.f. paragraphe 4.3).

4.2 Nouveau modèle

Le modèle est composé d'une machine d'états finis à variables $MEFV$ décrivant l'interaction avec le système et un vecteur V de variables d'état. Il est formellement défini par :

- Q est un ensemble d'états.
- q_{init} et q_{fin} sont respectivement l'état initial et l'état final.
- I est l'ensemble des entrées de l'application.
- O est l'ensemble des sorties de l'application.
- N est l'ensemble des noms de toutes les tâches dans le modèle CTT.
- $V = (V_{s1}, V_{s2}, ..., V_{sn})$ est un vecteur d'état, ou n est le nombre des variables d'états représentant l'état actuel de n composants de l'interface et V_{si} est l'ensemble de valeurs possibles de la variable d'état si. Si $v \in V_{s1} \times V_{s2} \times ... \times V_{sn}$ est une valeur du vecteur d'état, on note par $valeur(si, v)$ la valeur de la variable si dans v.
- $v_{init} \in V_{s1} \times V_{s2} \times ... \times V_{sn}$ est la valeur initiale du vecteur d'état.
- $trans$ est l'ensemble de transitions définies sur $Q \times P(N) \times (I \cup \{\mu\}) \times O \times Q$. Chaque transition (q, ns, i, o, p) (également notée $q \xrightarrow{i/o}_{(ns)} p$) est identifiée par un état source q, un ensemble de noms de tâches ns (impliquées dans la transition), une entrée i (qui peut être l'entrée vide μ), une sortie o et un état destination p. Parfois on omet les entrées, les sorties et/ou les noms de tâches (on notera $q \longrightarrow p$ au lieu

de $q \xrightarrow{i/o}_{(ns)} p$). Nous utilisons également la notation $q \xRightarrow{i/o} p$ pour définir :

– soit $q \xrightarrow{i/o} p$;

– soit $\exists q_1, ..., q_m$ tel que $q \xrightarrow{\mu/o_1} q_1 \xrightarrow{\mu/o_2} ... \xrightarrow{\mu/o_m} q_m \xrightarrow{i/o} p$

On note par $Action : O \times V_{s1} \times V_{s2} \times ... \times V_{sn} \rightarrow V_{s1} \times V_{s2} \times ... \times V_{sn}$ l'application reliant chaque sortie s du système interactif et une valeur de la variable d'état avec une nouvelle valeur de cette variable. Chaque fois que la sortie o est produite, V est modifié pour traduire le changement de l'interface. Pour l'exemple de Memo, on peut utiliser une seule variable d'état Nb_{notes} qui représente le nombre global de notes (portées et sur le terrain). La valeur de cette variable est modifiée uniquement par la sortie *memoRemoved* :

$$Action(memoRemoved, Nb_{notes}) = Nb_{notes} - 1$$

Notons que chaque transition $q \xrightarrow{i/o}_{(ns)} q'$ modifie le vecteur d'état d'une valeur v à une autre valeur v'. Cela est noté comme suit $(q, v) \xrightarrow{i/o}_{(ns)} (q', v')$.

4.3 Expression des propriétés

Atteignabilité Rappelons que l'atteignabilité est la capacité du système à permettre à l'utilisateur la possibilité de naviguer dans l'ensemble des états observables (i.e. états de l'interface du système). En supposant qu'un état observable de l'application interactive est une valeur du vecteur d'état V, vérifier qu'un état observable v' est atteignable revient à observer une transition (ou suite de transitions), dans le modèle, qui modifie la valeur du vecteur d'état à v'. Autrement dit, un état observable v' n'est pas atteignable si aucun chemin dans le modèle partant de l'état actuel ne peut mettre le vecteur d'état à v'.

Plus précisément, l'état observable v' est atteignable à partir de l'état initial, s'il existe dans $MEFV$ un état q' où le vecteur d'état a la valeur v' et un chemin exécutable à partir de l'état initial (q_{init}, v_{init}) vers l'état (q', v') $(\exists (q_1, v_1), (q_2, v_2), ..., (q_m, v_m)$ tel que $(q_{init}, v_{init}) \longrightarrow (q_1, v_1) \longrightarrow (q_2, v_2) \longrightarrow ... \longrightarrow (q_m, v_m) \longrightarrow (q', v'))$.

L'état observable v'est atteignable à partir d'un autre état v si :

– v est atteignable à partir de l'état initial (supposons à l'état (q, v)).

– $\exists q' \in Q$ et $\exists (q_1, v_1), (q_2, v_2), ..., (q_m, v_m)$ tel que $(q, v) \longrightarrow (q_1, v_1) \longrightarrow (q_2, v_2) \longrightarrow ... \longrightarrow (q_m, v_m) \longrightarrow (q', v')$.

Par exemple, dans Memo, supposons qu'une variable d'état Nb_{notes} indique le nombre total de notes. Si à l'état initial $Nb_{notes} = 2$, alors l'état $Nb_{notes} = 0$

est atteignable en effectuant deux fois la commande "remove".

Non-préemption Cette propriété caractérise le fait que l'utilisateur atteint directement son prochain but et est définie par rapport à une opération précise, telle que la suppression d'un fichier, l'ajout d'un utilisateur, le login, etc. En général, les opérations de suppression de fichiers ne doivent pas vérifier cette propriété (une confirmation de la suppression doit être demandée, ce qui signifie que le but, qui est la suppression du fichier, n'est pas directement atteint).

Nous assimilons le but à un état observable du système. Il est donc représenté par une valeur du vecteur d'état. Notons $G = (v_{sl}, ..., v_{sm})$, le but à atteindre par l'utilisateur. Dans le but G, il n'est pas nécessaire de spécifier les valeurs de toutes les variables d'état. Pour atteindre le but, l'utilisateur doit exécuter une séquence d'entrée du système $A = (a_i, ..., a_m)$ où $a_i, ..., a_m \in I$. Cette séquence est atomique ; c'est-à-dire que l'utilisateur l'exécute dans l'ordre et n'y ajoute aucune autre entrée.

Plus formellement, l'objectif de l'utilisateur $G = (v_{sl}, ..., v_{sm})$ est directement atteint par la séquence d'entrée $A = (a_1, a_2, ..., a_k)$ si et seulement si :

$\forall (q, v) \in Q \times V_{s1} \times V_{s2} \times ... \times V_{sn} : \exists (q_1, v_1), (q_2, v_2), ..., (q_k, v_k)$ tel que $(q, v) \overset{a_1/o_1}{\Longrightarrow} (q_1, v_1) \overset{a_2/o_2}{\Longrightarrow} (q_2, v_2) \overset{a_3/o_3}{\Longrightarrow} ... \overset{a_k/o_k}{\Longrightarrow} (q_k, v_k)$ et $\forall si$ dont la valeur est spécifié dans G alors $valeur(si, v_k) = valeur(si, G)$.

Prenons un exemple d'une application de dessin. Pour afficher un segment de droite ($G = (afficheSeg)$), on doit sélectionner un bouton de dessin de lignes, cliquer sur la position du premier point, puis cliquer sur la position du deuxième point ($A = (selectBotton, clickFirstPos, clickSecondPos)$). Le but est directement atteint si on voit le segment s'afficher après avoir cliqué sur la position du deuxième point sans aucune autre intervention.

Tolérance du rythme L'utilisateur plutôt que le système décide quand il peut agir. Cette propriété n'est pas vérifiée si, par exemple, on utilise des enchaînements automatiques d'une page Web vers la suivante. Elle est par contre respectée en cas de séquence vidéo à laquelle on ajoute des moyens de contrôle du flux vidéo.

Cette propriété est violée par une séquence de transitions avec des entrées vides (l'utilisateur ne peut pas contrôler la situation). La figure 4 montre un exemple ou cette propriété est violée : sur le chemin $q_1 \overset{\mu/o_1}{\longrightarrow} q_2 \overset{\mu/o_2}{\longrightarrow} q_3 \overset{\mu/o_3}{\longrightarrow} q_4$, à l'état q_2, nous n'avons pas de transition avec une entrée de l'utilisateur.

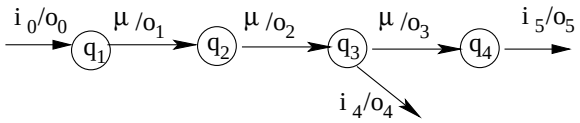

FIG. 4 – Exemple de violation de la tolérance au rythme.

Cette propriété est exprimée formellement comme suit (si dans l'automate il existe une suite de transitions avec des entrées vides, alors dans chaque état de cette suite l'utilisateur peut effecteur une action interrompant l'exécution de cette suite) :

si $\exists q_1, ..., q_m \in Q$ et $m \geqq 3$ tel que $q_1 \xrightarrow{\mu/o_2} q_2 \xrightarrow{\mu/o_3} q_3 .... \xrightarrow{\mu/o_m} q_m$, alors $\forall q_i, 1 < i < m : \exists t = q_i \xrightarrow{i/o} q'_i \wedge i \in I$

Interaction multifilaire Cette propriété peut concerner plusieurs caractéristiques du système, par exemple :

- On peut exécuter simultanément plusieurs tâches $T_1, T_2, T_3, ...$
- On ne peut pas exécuter T_1 alors qu'on est en train d'exécuter T_2 (on ne peut pas supprimer un fichier alors qu'on est en train de le modifier).

Ces propriétés sont formalisées de la manière suivante : chaque tâche concernée par cette propriété, qui est une tâche dans le modèle CTT, est représentée dans l'automate par l'ensemble des transitions qui lui appartiennent (ayant donc le nom de cette tâche dans ns). Vérifier l'exécution simultanée entre deux tâches T_1 et T_2, dont les transitions sont représentées respectivement par $trans_{T_1}$ et $trans_{T_2}$, revient à valider sur le modèle des chemins exécutant parfois des transitions de $trans_{T_1}$, parfois des transitions de $trans_{T_2}$ de façon entrelacée : $\exists q_{01} \longrightarrow_{ns_1} q_1 .... \longrightarrow_{ns_k} q_k$ tel que $\forall ns_i : T_1 \in ns_i \vee T_2 \in ns_i$ et que $\exists ns_i, ns_j : j \neq i \wedge T_1 \in ns_i \wedge T_2 \in ns_j$. Par exemple, prenons la figure 5 représentant un arbre de tâches et sa machine d'E/S associée. (D ||| C signifie l'entrelacement entre C et D). Le chemin d'exécution $q_1 \xrightarrow{a/s_a}_{\{A,D,T\}} q_2 \xrightarrow{c/s_c}_{\{C,T\}} q_5 \xrightarrow{b/s_b}_{\{B,D,T\}} q_6$ valide l'entrelacement entre D et C.

4.4 Limites de l'approche

Dans l'approche présentée dans ce papier, toutes les propriétés ergonomiques ne peuvent pas être formalisées à cause d'informations liées à ces propriétés et qui ne sont pas représentées dans le modèle, par exemple :

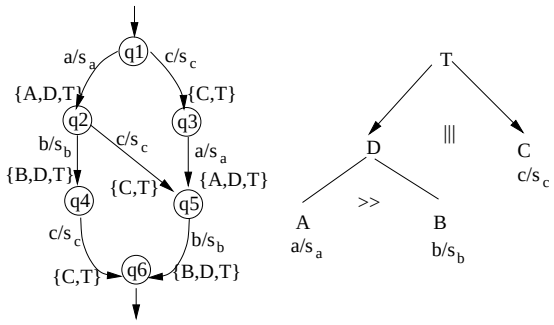

FIG. 5 – Un exemple de l'interaction multifilaire

- les propriétés dépendant de l'état interne de l'application comme l'observabilité (la capacité pour l'utilisateur d'évaluer l'état interne du système) et l'honnêteté (la capacité du système à rendre observable l'état interne du système sous une forme conforme à cet état) ;
- les propriétés liées à des attributs perceptuels tels que l'intensité sonore comme l'insistance (la capacité du système à forcer la perception de son état) et
- les propriétés concernées par le contexte d'utilisation comme l'adaptabilité (la personnalisation du système sans intervention explicite de l'utilisateur).

5 Conclusion et perspectives

L'extension du modèle formel permet désormais l'expression de certaines propriétés ou de scénarios qui ont trait aux aspects ergonomiques d'une application interactive. Cela permet d'envisager de définir des techniques de guidage de la génération de tests. Nous avons donc effectué un pas pour permettre, à terme, d'automatiser la validation de propriétés ergonomiques.

Néanmoins, ce travail a nécessité d'ajouter des informations (des probabilités, des variables d'états) sur un arbre CTT. Bien qu'a priori ces ajouts n'introduisent pas d'incohérence, ils n'ont, pour l'instant, pas été intégrés au niveau de la définition de la sémantique des arbres de tâches. Outre la sémantique, il apparaît aussi nécessaire de faire évoluer l'outil d'édition de modèles CTT ou de proposer des outils complémentaires pour permettre aux concepteurs d'ajouter les informations nécessaires au test.

Enfin il reste surtout à exploiter notre modèle formel pour valider des propriétés ergonomiques. Si notre travail a permis de préciser certaines pro-

priétés, la formalisation doit maintenant être utilisée pour générer des tests. Nous pourrons ainsi proposer aux concepteurs un outil de validation d'interfaces qui utiliserait à la fois la spécification fonctionnelle donnée par les arbres de tâches et les propriétés ergonomiques.

Références

[1] Gregory D. Abowd, Joëlle Coutaz, and Laurence Nigay. Structuring the space of interactive system properties. In James A. Larson and Claus Unger, editors, *Engineering for Human-Computer Interaction*, volume A-18 of *IFIP Transactions*, pages 113–129. North-Holland, 1992.

[2] Yamine Ait-Ameur and Mickael Baron. Bridging the gap between formal and experimental validation approaches in HCI systems design : use of the event b proof technique. In *ISOLA*, pages 74–81, 2004.

[3] Yamine Ait-Ameur, Mickael Baron, and Nadjet Kamel. Encoding a process algebra using the event b method. application to the validation of user interfaces. In *ISOLA*, page 17, 2005.

[4] Rémi Bastide, Philippe A. Palanque, Duc-Hoa Le, and Jaime Munoz. Integrating rendering specifications into a formalism for the design of interactive systems. In *DSV-IS*, pages 171–190, 1998.

[5] J. Bouchet, L. Madani, L. Nigay, C. Oriat, and I. Parissis. Formal testing of multimodal interactive systems. In *EIS'2007 Engineering Interactive Systems*, pages 36–52, Salamanca, Spain, March 2007.

[6] J. Bouchet and L. Nigay. Icare : a component-based approach for the design and development of multimodal interfaces. In *CHI Extended Abstracts*, pages 1325–1328, 2004.

[7] Bruno d'Ausbourg. Using model checking for the automatic validation of user interface systems. In *DSV-IS*, pages 242–260, 1998.

[8] A. Dittmar. More precise descriptions of temporal relations within task models. In *Interactive Systems : Design, Specification, and Verification, 7th International Workshop DSV-IS, Proceedings*, pages 151–168, Limerick, Ireland, 5-6 June 2000.

[9] Abowd G., Coutaz J., and Nigay L. Structuring the space of interactive system properties. In *Engineering for Human-Computer Interaction*, pages 113–129, 1992.

[10] Melody Y. Ivory and Marti A Hearst. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.*, 33(4) :470–516, 2001.

[11] Nogier J.-F. *Ergonomie du logiciel et design web - 2ème Edition.* Dunod, 2003.

[12] Francis Jambon, Patrick Girard, and Yohann Boisdron. Dialogue validation from task analysis. In David J. Duke and Angel R. Puerta, editors, *DSV-IS*, pages 205–224. Springer, 1999.

[13] Laya Madani. *Utilisation de la programmation synchrone pour la spécification et la validation de services interactifs.* PhD thesis, Université Joseph Fourrier, Grenoble, France, Octobre 2007.

[14] Laya Madani and Ioannis Parissis. Vers la génération automatique de tests à partir d'arbres de tâches. In *Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL)*, pages 125–141, Namur, Belgique, Juin 2007.

[15] Laya Madani and Ioannis Parissis. Automated test of interactive applications using task trees. In *4th Workshop on Advances in Model-based Testing (A-MOST)*, pages 13–22, Lillehammer, Norway, 2008.

[16] G. Mori, F. Paternò, and C. Santoro. CTTE : Support for developing and analyzing task models for interactive system design. *IEEE Transactions on Software Engineering (TSE)*, 28(8) :797–813, 2002.

[17] Giulio Mori, Fabio Paternò, and Carmen Santoro. Ctte : Support for developing and analyzing task models for interactive system design. *IEEE Trans. Software Eng.*, 28(8) :797–813, 2002.

[18] F. Paternò ; and G. Faconti. On the use of LOTOS to describe graphical interaction. In *HCI'92 : Proceedings of the conference on People and computers VII*, pages 155–173, New York, NY, USA, 1993. Cambridge University Press.

[19] Fabio Paternò. A formal approach to the evaluation of interactive systems. *SIGCHI Bull.*, 26(2) :69–73, 1994.

[20] Bastien J. M. C. Scapin D. Ergonomic criteria for evaluating the ergonomic quality of interactive systems. *Behaviour and Information Technology*, 16 :220–231, 1997.

[21] Harold Thimbleby, Paul Cairns, and Matt Jones. Usability analysis with markov models. *ACM Trans. Comput.-Hum. Interact.*, 8(2) :99–132, 2001.