DU MODÈLE DE TÂCHES AU MODÈLE DE DIALOGUE DES APPLICATIONS INTERACTIVES

Sybille Caffiau
LISI-ENSMA
1, Avenue Clément ADER
86960 FUTUROSCOPE CEDEX
(+33/0)5 49 49 80 70
sybille.caffiau@ensma.fr

RÉSUMÉ
Afin de recueillir les besoins et de les formaliser, les modèles de tâches ont été développés. Malheureusement, ils ne sont que très rarement utilisés dans les outils de modélisation et de développement des applications interactives bien qu'ils aient prouvé leur utilité dans les phases de conception et de validation de ces applications.

Un moyen d'intégrer ces modèles de tâches lors du développement du logiciel est de les relier à la dynamique de l'application : le dialogue. En effet, les modèles de tâches et de dialogue sont deux manières différentes de décrire la même application.

Nous avons dans un premier temps étudié les liens entre ces modèles. Pour cela, nous avons essayé de déduire le modèle de dialogue à partir du modèle de tâches. Nous montrerons, en nous basant sur des exemples, comment passer directement du modèle de tâches au modèle de dialogue des applications interactives. Nous en étudierons les limites, et esquisserons des pistes de travail visant à permettre une généralisation.

MOTS CLÉS
Modèles de tâches, Modèles de dialogue, Machine à états

ABSTRACT
Task models were built to collect and formalize needs. Unfortunately, although their utility has been proven in design and validation steps, they are rarely used in modeling tools and in interactive application design.

A mean to add those task models during the development of software is to link them to application dynamic: the dialog. Indeed, task models and dialog are two different ways to describe the same application.

We studied the links between those models. In order to realize this study, we tried to deduce the dialog model from the task model. In this paper, we will show, using some examples, how to go from the task model of interactive applications to the dialog model. Finally, we will study the limits and outline the ways aiming to permit a generalization.

Categories and Subject Descriptors
H.5.2 User Interface : Theory and Method
I.6.4 Model Validation and Analysis
D.2.2 Design Tools and Techniques

General Terms
Design, Reliability, Languages, Theory, Verification.

Keywords
Task models, Dialogue Models, Interactive Systems Design and Verification, State Machine

1. INTRODUCTION
Les modèles de tâches et de dialogue sont deux manières différentes de décrire la même application, c'est pourquoi on souhaite pouvoir passer de l'un à l'autre de ces modèles. Une première étape est d'essayer d'obtenir les informations nécessaires à la réalisation du modèle de dialogue à partir du modèle de tâches.

Afin d'étudier ce passage, nous nous sommes appuyés sur plusieurs applications : deux convertisseurs de monnaies (l'un réalisant la conversion après validation des données et l'autre, dés qu'une des données a été modifiée), un jeu de mastermind avec deux modes de jeu (un joueur ou deux), une application multi-fenêtres de gestion administrative. Nous avons choisi des applications de types très différents afin d'essayer de balayer le champ des applications possibles. Cependant, cette approche n'est pas exhaustive, certains types d'applications n'ont pas été traités, comme les éditeurs par exemple. Pour chacune de ces applications, nous avons réalisé un modèle de tâches, une modélisation du dialogue et nous les avons comparés.

Nous avons choisi d'utiliser le formalisme ConcurrentTaskTree (CTT)[10], pour décrire le modèle de tâches. Ce choix a été motivé par le fait que cette notation est très largement utilisée et que de nombreux travaux partent de cette notation pour obtenir le dialogue d' applications [5, 8, 12]. Le dialogue, quant à lui, a été modélisé par des machines à états. Cette formalisation nous permet d'exprimer le dialogue d'une application de manière complète et d'obtenir le code correspondant en utilisant les machines à états de la bibliothèque SwingStates[1].

Nous allons, dans un premier temps décrire les éléments nécessaires à l'expression du dialogue pour une application interactive. L'étude des différentes applications-exemples nous a permis d'établir les éléments d'un modèle de dialogue qui peuvent être obtenus à partir du modèle de tâches CTT, et ceux qui ne le sont pas. Nous allons les présenter avant de conclure.

2. LES ÉLÉMENTS NÉCESSAIRES À L'EXPRESSION DU DIALOGUE
Les modèles de dialogue sont utilisés pour spécifier la dynamique du dialogue entre l'homme et la machine. En référence au modèle de Seeheim [11], ils sont décrits dans [3] comme une double liaison :

[1] http://www.lri.fr/~appert/SwingStates/index.html

- d'une part, une liaison avec la sémantique du système interactif pour que celui-ci sache ce qu'il doit faire,
- d'autre part, une liaison avec la présentation pour donner la visualisation de ce système.

Pour satisfaire cette définition, le dialogue d'une application doit accomplir certaines fonctions. Il doit pouvoir identifier les entrées de l'utilisateur et pour chacune d'entre elles, définir le traitement à appliquer. Afin de réaliser ces traitement, il doit solliciter les services du noyau fonctionnel de l'application [9].

Afin de concrétiser notre démarche, nous avons choisi d'exprimer notre dialogue à l'aide d'un modèle opérationnel déjà cité, SwingStates. De ce fait, la machine à états représentant le dialogue doit respecter la syntaxe définie par cette bibliothèque. Pour chaque état, il faut identifier les transitions sortantes. Une transition est définie par l'action qui est réalisée lors de son exécution, son état d'arrivée (désigné par son nom) et par le type d'interaction qui la déclenche. Cet élément est nécessaire lors de la création d'une transition afin d'utiliser le constructeur de la bonne classe-fille de la classe Transition.

3. LES ÉLÉMENTS IDENTIFIABLES À PARTIR DU MODÈLE DE TÂCHES CTT

À partir des observations faites sur les applications-exemples décrites dans l'introduction, nous proposons de réaliser un modèle de dialogue (sous forme de machine à états) fidèle au modèle de tâches en parcourant l'arbre CTT correspondant, afin de savoir si le modèle de dialogue est déductible du modèle de tâches.

Pour cela, à chaque tâche du modèle de tâches, nous associons trois ensembles d'états : l'ensemble des *états initiaux*, l'ensemble des *états transitoires* et l'ensemble des *états finaux*.

Les *états initiaux* sont les états dans lesquels se trouve le système avant la réalisation de la tâche. Pendant son exécution, les éventuels états atteints sont nommés états transitoires. Une fois que la tâche a été totalement exécutée, le système se trouve dans l'un des *états finaux* possibles pour cette tâche.

La nature, les caractéristiques et l'organisation temporelle des tâches entre elles sont autant de paramètres qui influent sur ces ensembles d'états.

3.1 Grâce à la présentation hiérarchique et les types de tâches

La notation CTT organise hiérarchiquement les tâches dans un arbre. Dans cette notation, les tâches peuvent être de quatre types différents. Les tâches « utilisateur » sont des tâches réalisées entièrement par l'utilisateur. Les tâches « système » sont uniquement exécutées par le système. Lorsqu'une tâche est une action d'un utilisateur sur le système, alors c'est une tâche « interactive ». Si une tâche est constituée d'un ensemble de tâches de type différent, c'est une tâche « abstraite ». La place des tâches dans cet arbre implique des caractéristiques sur les ensembles d'états qui lui sont associés. Afin de les présenter, nous allons considérer une tâche T appartenant à l'arbre CTT. T peut donc être : la tâche racine, un nœud ou une feuille.

Si T est la tâche racine, alors l'ensemble de ses *états initiaux* est constitué d'un seul état, l'état initial du système (l'état dans lequel se trouve le système à l'ouverture de l'application). De même l'ensemble de ses *états finaux* est constitué d'un seul état, l'état dans lequel se trouve le système une fois l'application fermée.

Si T est un nœud, alors l'ensemble de ses *états initiaux* est le même que celui de sa première tâche-fille. De même, son ensemble *d'états* finaux est le même que celui de sa dernière tâche-fille. De plus, l'ensemble des *états transitoires* de T est l'ensemble de tous les états dans lesquels peut se trouver le système pendant la réalisation de ses tâche-filles, exception faite de l'ensemble *d'états* initiaux de sa première tâche-fille et l'ensemble des *états finaux* de sa dernière tâche-fille.

Si T est une feuille, alors c'est une tâche atomique, elle ne possède pas d'*états transitoires*. Cependant, elle peut être représentée dans la machine à états par une transition, cela dépend du type de la tâche. Une tâche utilisateur étant une tâche cognitive entièrement réalisée par l'utilisateur, elle ne modifie pas le modèle de dialogue, c'est pourquoi une tâche feuille de type utilisateur n'est pas représentée sur la machine à états. Elles pourraient cependant être utilisées pour apporter des informations supplémentaires, comme par exemple savoir si un feedback est nécessaire ou si une tâche est exécutée conformément au souhait de l'utilisateur.

Au contraire, les tâches interactives sont toutes des transitions puisqu'elles représentent l'action des utilisateurs sur le système donc, une modification du système par l'utilisateur.

Les tâches systèmes, quant à elles, sont divisées en deux cas. D'une part, certaines sont uniquement des conséquences de l'exécution d'autres tâches, c'est le cas des tâches d'impression par exemple, ce ne sont donc pas des transitions dans la machine à états. D'autre part, les tâches ayant leur propre existence, telles que le déclenchement d'un chronomètre par exemple, modifient le dialogue du système et donc sont des transitions. En effet, si une action doit être réalisée en un laps de temps fixé, à la fin de cette durée le système doit se trouver dans un état qui ne permet pas l'exécution de ladite action.

Comme nous l'avons dit précédemment, le type des tâches des tâches atomiques a des conséquences sur la machine à états du modèle de dialogue. Si les tâches interactives et cognitives se traitent toutes de la même manière, les tâches systèmes peuvent être traités de deux manières différentes. D'une part, certaines tâches système qui n'ont pas de conséquences sur le modèle de dialogue, qui sont des tâches qui peuvent être exécutées pendant la transition d'une tâche à une autre. Les tâches de feedback sont dans ce cas. D'autre part, les tâches systèmes qui ont un rôle dans le modèle de dialogue, comme les tâches qui en interrompent d'autre. Par exemple, dans une application pour laquelle une tâche ne peut être réalisée que pendant une durée déterminée, l'horloge interrompt le dialogue quand le temps est écoulé. Dans ce cas, la tâche système est une transition. Cette division des tâches systèmes nous permet d'obtenir toutes les transitions utiles mais seulement celles-là.

3.2 Grâce aux opérateurs temporels

L'utilisation des différents opérateurs temporels proposés par CTT pour organiser les tâches entre elles implique des conséquences sur les ensembles d'états. Cependant, les conséquences ne sont pas influencées par le passage d'information, c'est pourquoi nous avons traité en même temps les opérateurs d'activation (>> et []>>) et de concurrence (||| et |[]|) avec et sans passage d'information.

3.2.1.1 Les opérateurs temporels basiques

Selon Baron [1], il existe deux types d'opérateurs temporels, les opérateurs basiques et ceux qui peuvent s'exprimer en fonction de ces premiers. Les opérateurs basiques sont les opérateurs d'activation, de choix, de concurrence, l'itération et l'itération finie.

La liaison de deux tâches par l'un de ces opérateurs implique des conséquences sur la manière dont elles se succéderont et donc la manière dont elles sont exprimées dans la machine à états.

L'opérateur d'activation entre deux tâches T1 et T2 (T1>>T2 et T1[]>>T2) rend obligatoire l'exécution totale de la tâche T1 avant celle de la tâche T2. Donc, l'ensemble des états qui permettent l'exécution de T2 (les *états initiaux* de T2), est donc l'ensemble des états dans lesquels se trouve le système une fois T1 exécutée (les *états finaux* de T1).

Si T1 et T2 sont liées par l'opérateur de choix (T1[]T2), alors ces deux tâches doivent pouvoir être exécutées à partir des mêmes états, elles ont donc le même ensemble d'*états initiaux*.

Si une tâche T est composée par deux tâches T1 et T2 liées par l'opérateur temporel de concurrence (T1 ||| T2 et T1 |[]| T2) alors T1 et T2 sont toutes deux réalisables à partir de l'ensemble des *états initiaux* de T. De plus, après l'exécution de l'une des deux sous-tâches, l'autre est réalisable. Donc les ensembles des états finaux de T1 et T2 sont les ensembles des états initiaux de T2 et T1.

Une tâche itérative T* doit être capable de se ré-exécuter. Donc, l'état dans lequel se trouve le système après l'exécution de T doit être dans l'ensemble de ces *états initiaux*. C'est pourquoi les ensembles des *états initiaux* et des *états finaux* de T sont les mêmes.

Une tâche qui est itérée un nombre de fois fini (T(N)) doit respecter la règle de l'itération un nombre de fois fini N.

3.2.1.2 Les opérateurs temporels non basiques

Quelques opérateurs temporels peuvent être exprimés en fonction des opérateurs basiques, c'est le cas de l'ordre indépendant, de la désactivation et de la suspension.

L'exécution d'une tâche T composée des sous-tâches T1 et T2 réalisées dans un ordre indépendant (T1 |=| T2) est équivalent à l'exécution de T1 puis de T2 ou l'exécution de T2 puis de T1 ((T1>>T2) [] (T2>>T1)). C'est pourquoi il faut respecter la règle d'activation entre T1 et T2 d'une part et entre T2 et T1 d'une autre part. L'ensemble des *états finaux* de T1 et celui des *états initiaux* de T2 sont donc les mêmes. De même pour l'ensemble des *états finaux* de T2 et des *états initiaux* de T1. De plus, l'opérateur de choix entre les deux expressions (T1>>T2) et (T2>>T1) impose que les tâches T1 et T2 puissent être exécutées en premier, il faut donc qu'ils aient le même ensemble d'*états initiaux*.

On peut obtenir deux réécritures différentes pour l'utilisation de l'opérateur de désactivation (T1[>T2), selon si la tâche T1 est atomique ou non.

Si T1 est une tâche atomique, alors soit elle est exécutée entièrement, soit elle ne l'est pas du tout donc, T2 interrompt T1 est équivalent à T1 ou T2 (T1[]T2). Tout comme pour l'utilisation de l'opérateur de choix, il faut donc que les ensembles des *états initiaux* soient identiques.

Si T1 n'est pas une tâche atomique alors elle est composée d'un ensemble de tâches $T1_1$, $T1_2$, …, $T1_n$ liées entre elles par un ensemble d'opérateurs op_1, op_2, … op_{k-1}. T1 est partiellement exécutée avant d'être interrompue par T2 signifie que T1 n'est pas du tout exécuté ou seul $T1_1$ est exécuté ou $T1_1$ op_1 $T1_2$ est exécuté…avant l'exécution de T2. Donc, T2 doit pouvoir débuter son exécution à partir de n'importe quels états de $T1_1$, $T1_2$, …C'est-à-dire, à partir de n'importe quel état (initial, transitoire) de T1.

Dans le cas où la tâche T1 est une tâche itérative (T1*), les deux cas vus précédemment (si T1 est atomique ou non) s'appliquent également.

Si la tâche T1 est atomique, elle peut être exécutée N (N pouvant être égal à 0) fois avant que T2 ne s'exécute. T1*[>T2 peut donc se réécrire T1(N)>>T2, c'est donc équivalent à une itération finie appliquée à une tâche qui permet l'activation d'une autre. Nous avons donc, les *états finaux* de T1 équivalents à ses *états initiaux* pendant ses N exécutions, puis identiques aux *états initiaux* de T2.

Dans le cas où T1 n'est pas atomique alors, elle est totalement exécutée un nombre de fois fini et une fois partiellement, c'est pourquoi tous les états de T1 doivent permettre l'exécution de T2.

L'utilisation de l'opérateur de suspension pour lier deux tâches (T1 |> T2) impose qu'à tout moment de l'exécution de la première, la seconde peut s'exécuter et qu'une fois l'executionde celle-ci terminée, T1 reprend là où elle s'était arrêtée. Tous les ensembles d'états de T1 sont donc des *états initiaux* de T2 et pour permettre la reprise de l'exécution de T1, tous les ensembles d'états de T2 sont les *états initiaux* de T1.

4. LES ÉLÉMENTS QUI MANQUENT

Nous avons appliqué les règles décrites ci-dessus sur les modèles de tâches de nos applications-exemples. Les machines à états que nous avons ainsi obtenues n'exprimaient pas le dialogue complet.

Certaines transitions étaient absentes. Nous avons identifié trois types d'absence différents. Certaines transitions, nommées « *transitions-retour* », sont des transitions qui ramènent le système dans un état précédent à la suite de « l'effacement » d'une précédente action. Par exemple, si un bouton est actif que lorsqu'un TextField n'est pas vide alors l'action d'effacer son contenu est traduite par l'exécution d'une transition pendant laquelle le bouton est inactivé. Cette transition ramène donc le système dans l'état où il était avant la saisie qui avait permis l'activation du bouton.

Il manquait également les transitions représentant les erreurs de l'utilisateur. Par exemple, pour une application de conversion, lors de la saisie d'une somme à convertir, dans un textField si l'utilisateur entre une lettre à la place d'un chiffre alors le dialogue du système doit pouvoir réagir en conséquence (en traitant l'erreur). Les modèles de tâches ne permettent que très rarement de spécifier le traitement des erreurs de l'utilisateur. Cependant, les modèles de tâches sont conçus pour spécifier les besoins des utilisateurs donc ce n'est pas leur rôle de prévoir les erreurs de l'utilisateur.

Le dernier type de transitions absentes étaient les transitions qui permettaient de changer de fenêtre, dans les applications multi-fenêtre. En effet, chacune des fenêtres permet la réalisation d'un ensemble de tâches, le système change d'état en fonction de la fenêtre active.

En plus de l'absence de certaines transitions, le manque d'information concernant la manière dont l'utilisateur réalise une tâche atomique rendait nécessaire l'ajout d'information. En effet, les tâches atomiques correspondent à des transitions dans le modèle de dialogue et le constructeur d'une transition dépend de la manière dont elle va être déclenchée.

5. CONCLUSION

Le dialogue obtenu à partir du modèle de tâches n'est que partiel. Certaines informations, impérativement présentes dans un dialogue totalement défini, sont impossibles à obtenir à partir

d'un modèle de tâches seul. Nous avons cherché à obtenir les dialogues d'applications-exemples à partir de leurs modèles de tâches. Nous avons alors constaté qu'à partir d'un diagramme CTT, certaines informations nécessaires au dialogue n'apparaissaient pas sur la machine à états que nous obtenions. L'utilisation des pré et post-conditions permettrait de combler certains de ces manques. La notation CTT met à disposition ces conditions mais sans réelle définition sémantique. Nous avons utilisé ce formalisme, mais une étude ultérieure d'un autre formalisme, incluant une sémantique des pré et post-conditions, permettrait d'obtenir un automate plus complet.

De même qu'une étude d'autres formalismes de modèles de tâches apporterait de nouvelles informations, d'autres formalismes de modèles de dialogue pourraient permettre une meilleure description du dialogue. Comme, par exemple, l'utilisation des *Statecharts* [4] qui permettrait d'obtenir un modèle de dialogue plus clairement compréhensible pour de grands dialogues et la conservation de la hiérarchie des tâches.

De plus, nos applications-exemples ne représentent pas l'ensemble des applications interactives possibles. En effet, nous n'avons pas traité de cas d'applications de type post-WIMP, ni d'exemple d'éditeurs, qui présentent la particularité d'avoir un modèle de tâches linéaires, puisque presque toutes les tâches peuvent être réalisées parallèlement. De ce fait, certains manques n'ont pas été identifiés.

Enfin, les modèles de tâches ont été conçus afin de permettre l'expression des différentes activités que l'utilisateur final souhaite pouvoir réaliser [2], ils ne spécifient donc pas les éventuelles erreurs de l'utilisateur final. Or, le dialogue d'une application interactive doit les prendre en compte afin de les traiter.

Un modèle de tâches n'est donc pas suffisant pour générer le dialogue d'une application interactive. Certains travaux réalisent une semi-génération du dialogue à partir du modèle de tâches. TERESA (Transformation Environment for inteRactivE Systems) [6, 7] est l'un d'eux. Les états sont calculés en fonction des tâches exécutables à un même instant. À chaque état correspond une *Presentation* dans laquelle sont positionnés les widgets nécessaires à l'exécution de l'une ou l'autre de ces tâches. Malheureusement, cet outil permet la réduction du nombre d'états à l'aide de l'application manuelle ou automatique d'heuristiques, ce qui a pour conséquence la perte de l'équivalence entre le modèle de tâches et le modèle de dialogue. En effet, du fait de la manière dont les fenêtres sont obtenues, le dialogue à l'intérieur de chacune d'entre elles n'est pas géré. Chaque *Presentation* est une boîte de dialogue dans laquelle toutes les tâches sont réalisables en parallèle.

Un autre outil, Dygimes framework [5], permet également la génération du dialogue à partir du modèle de tâches. TERESA et Dygimes framework sont deux approches très proches, qui utilisent tous deux un calcul de tâches actives pour définir un état.Cependant, Dygimes ne prend pas en compte le dialogue inter-fenêtre et ne se préoccupe pas de ce qui peut se produire dans une fenêtre (activation d'un bouton, par exemple).

Nous avons étudié le passage d'un modèle de tâches, sous forme d'un diagramme CTT, vers la machine à états représentant le dialogue d'une application interactive. Cette étude nous a permis d'établir les conséquences des éléments du premier modèle sur ceux du second. Une étude du passage inverse permettrait de connaître les implications du modèle de dialogue sur le modèle de tâches. De cette manière, nous pourrions comparer les deux modèles d'une même application afin de pouvoir vérifier leurs propriétés communes.

6. REFERENCES

[1] M. Baron, "Vers une approche sûre du développement des Interfaces Homme-Machine (Thesis)." Poitiers: Université de Poitiers, 2003, pp. 255.

[2] D. Diaper and N. A. Stanton, The Handbook of Task Analysis for Human Computer Interaction: Lawrence Erlbaum Associates, 2004.

[3] A. Dix, J. Finlay, G. Abowd, and R. Beale, Human-Computer Interaction: Prentice Hall, 1993.

[4] D. Harel, "Statecharts: a visual formalism for complex systems," Science of Computer Programming, vol. 8, pp. 231-274, 1987.

[5] K. Luyten, "Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development," in School of Information Technology. Diepenbeek, Belgium: University Limburg, 2004, pp. 194.

[6] G. Mori, F. Paternò, and C. Santoro, "Tool Support for Designing Nomadic Applications," presented at Intelligent User Interfaces (IUI'2003), Miami, Floride, 2003.

[7] G. Mori, F. Paternò, and C. Santoro, "Design and Development of Multidevice User Interfaces through MultipleLogical Descriptions " IEEE Transactions on Software Engineering, pp. 507-520, 2004.

[8] D. Navarre, P. Palanque, F. Paterno, C. Santoro, and R. Bastide, "A Tool Suite for Integrating Task and System Models through Scenarios," in Interactive Systems Design, Specification, and Verification (DSV-IS'01), SpringerComputerScience, C. Johnson, Ed. Glasgow, Scotland, UK: Springer-Verlag, 2001, pp. 88-113.

[9] D. R. Olsen, User Interface Management Systems: Models and Algorithms. San Mateo (CA), USA: Morgan Kaufmann Publisher, 1992.

[10] F. Paternò, Model-Based Design and Evaluation of Interactive Applications: Springer, 2001.

[11] G. E. Pfaff, "User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim," in Eurographic Seminars. Berlin: Springer-Verlag, 1985.

[12] C. Pribeanu and J. Vanderdonckt, "Exploring Design Heuristics for User Interface Derivation From Task and Domain Models," presented at Computer-Aided Design of User Interfaces (CADUI'2002), Valenciennes, France, 2002.