N° D'ORDRE : 03/2007-D/IN

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique

Thèse

Présentée pour l'obtention du diplôme de DOCTORAT

EN : INFORMATIQUE

par :

Nadjet KAMEL

sujet :

Un cadre formel générique pour la modélisation d'IHM Multi-modales. Cas de la multi-modalité en entrée

Soutenue le 28/04/2007, devant le jury composé de :

Mohamed AHMED NACER	Professeur, LSI, Université des Sciences et de la Technologie Houari Boumedien, Alger	**Président**
Yamine AIT AMEUR	Professeur, LISI/ENSMA, Université de Poitiers	**Directeur de thèse**
Habiba DRIAS	Professeur, LRIA, Université des Sciences et de la Technologie Houari Boumedien, Alger	**Co-Directeur de thèse**
Malika BOUKALA	Professeur, LSI, Université des Sciences et de la Technologie Houari Boumedien, Alger	**Examinateur**
Patrick GIRARD	Professeur, LISI/ENSMA, Université de Poitiers	**Examinateur**
Guy PIERRA	Professeur, LISI/ENSMA, Université de Poitiers	**Examinateur**
Zoheir ZEMIRLI	Maître de conférences, LCSI, Institut National de formation en Informatique, Alger	**Examinateur**

A mes parents
A tous ceux qui m'aiment
A tous ceux que j'aime

Je tiens à remercier :

Yamine Ait Ameur, professeur et directeur adjoint du LISI, pour m'avoir encadrée, beaucoup conseillée, aidée et pour avoir oeuvré pour que ce travail aboutisse. Je lui exprime toute ma gratitude ;

Habiba Drias, professeur et Directrice Générale de l'Institut National de formation en Informatique à Alger (INI), pour la confiance qu'elle a toujours eue en moi, pour m'avoir associée au projet de l'accord programme CMEP, entre le laboratoire LRIA de l'USTHB et le LISI de l'ENSMA, qui a financé une partie de mes séjours au LISI, pour ses encouragements et pour avoir co-encadré les travaux de cette thèse ;

Mohamed Ahmed Nacer, professeur au département d'informatique de l'USTHB, pour avoir accepté de présider le jury de cette thèse ;

Guy Pierra, professeur et directeur du laboratoire de recherche LISI, pour m'avoir accueillie et prise en charge au sein de son laboratoire et pour avoir accepté de participer au jury de thèse ;

Malika Boukala, professeur et directrice du département d'informatique à l'USTHB, Patrick Girard, professeur et directeur de l'équipe " modélisation des Interfaces Homme-Machine et Programmation sur l'Exemple " au LISI, et Zoheir Zemirli, maître de conférences à l'INI, pour avoir accepté de participer au jury de cette thèse ;

Tous les collègues du département d'Informatique de l'USTHB pour leurs soutiens et encouragements. Je remercie particulièrement Saliha Aouat, Nassima Alleb, Ahlem Benchennaf et Saliha Bouagar pour m'avoir remplacé dans mes enseignements à l'USTHB durant mes séjours au LISI ;

Tous les membres du LISI pour l'ambiance sympathique qu'ils ont su créer durant mes séjours au LISI. Je remercie particulièrement Idir Ait Sadoune, Youcef Aklouf, Mounira Bachir, Mickael Baron, Ladjel Bellatreche, Frédéric Carreau, Sofiane Challal, Hondjack Dehainsala, Mourad El Hadj Mimmoune, Nicolas Guibert, Stephane Jean, Jean-Marc Mota, Dung Nguyen Xuan, Claudine Rault et Loe Sanou pour leurs services et aides.

Résumé

Nos travaux proposent un cadre méthodologique formel générique permettant la conception formelle d'IHM multi-modales et la formalisation de l'expression et de la validation de propriétés d'utilisabilité associées. Ce cadre s'appuie sur un même modèle formel unifié autour des systèmes de transitions. Dans une seconde partie, nous montrons comment les modèles issus du cadre générique sont mis en oeuvre dans des techniques formelles particulières et hétérogènes du point de vue de la sémantique. Deux catégories de techniques sont abordées. La première fondée sur le model-checking avec deux approches, l'une à base de variables d'états et de la logique temporelle arborescente CTL avec l'outil SMV et l'autre à base d'actions et de la logique temporelle linéaire LTL avec l'outil Promela/Spin. La seconde catégorie de technique mise en oeuvre est fondée sur la preuve interactive et sur le raffinement. Elle met en oeuvre la méthode B dans sa version événementielle.

Abstract

Our work proposes a generic formal methodological framework allowing the formal design of multimodal HCI and the formalisation of the expression and the validation of associated usability properties. This framework is based on the same formal model based on the transition systems. In a second part of the work, we show how the models issued from the generic framework are implemented in particular and heterogeneous (from the semantics point of view) formal techniques. Two categories of formal techniques are put into practice. The first one is oriented towards model-checking with two kinds of temporal logics, one uses state variables and the branching temporal logic CTL with SMV while the other is based on actions and uses linear temporal logic LTL with the Promela/Spin. The second category of used formal technique is based on proof and refinement within the event B method.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Le développement de nouveaux dispositifs de communication homme-machine (systèmes de reconnaissance et de synthèse de parole, écrans tactiles, gants numériques, etc.) a contribué à l'émergence de nouvelles techniques d'interaction avec les systèmes, ce qui a donné naissance à de nouveaux types d'interfaces homme-machine dites : Interface Homme-Machine Multimodales (IHM3). Un système est dit multimodal s'il dispose d'une IHM3 qui offre plus d'une modalité d'interaction en entrée (de l'utilisateur vers le système) et/ou en sortie (du système vers l'utilisateur).

Ces IHM3 ont pour objectif d'approcher le mode d'interaction existant chez l'homme. Elles tentent de fournir une interaction et une communication proche de celle pratiquée entre les humains. Elles permettent ainsi d'offrir à l'utilisateur une interaction assez intuitive avec le système. L'utilisateur peut utiliser la parole, la souris, le geste etc. pour communiquer avec le système. En conséquence, ces IHM permettent de s'adapter plus facilement aux utilisateurs handicapés moteurs en leur permettant de communiquer avec la parole. Elles sont mieux adaptées aux systèmes de sécurité en permettant la continuité de l'interaction en cas de panne d'un dispositif d'interaction.

Ainsi, du fait de la richesse des modes d'interaction fournis et les dispositifs associés, les caractéristiques de ce type d'IHM3 posent de nouveaux problèmes de conception et de développement de systèmes interactifs multimodaux. Ce type de systèmes interactifs devient de plus en plus complexe et donc plus difficile à concevoir et à valider. En effet, contrairement aux systèmes interactifs de type WIMP (Windows, Icon, Menus, Pointing devices) qui possèdent un caractère séquentiel, les IHM3 utilisent des modes de communication mettant en oeuvre la séquence, le parallélisme et la synchronisation de différentes interactions.

Nos travaux s'inscrivent dans ce contexte. Ils visent à fournir un cadre méthodologique générique et rigoureux permettant d'exprimer et de valider des systèmes interactifs multimodaux en entrée tout en conservant les notations et le savoir faire des concepteurs traditionnels d'IHM3. L'objectif est de fournir une assistance rigoureuse et à priori permettre à ces mêmes développeurs de mieux concevoir ces systèmes interactifs et de reproduire ces développements pour d'autres systèmes.

Plusieurs travaux ont été effectués dans ce domaine. Les premiers travaux se sont intéressés à préciser et à fixer les concepts de ce type de systèmes. Ensuite, d'autres

travaux ont permis de définir et de catégoriser les types d'interfaces multimodales ainsi que les propriétés d'utilisabilité qui les caractérisent. En parallèle, d'autres travaux ont traité du développement et du test des différents usages de multimodalité identifiés par les concepteurs et les utilisateurs au fur et à mesure que les nouveaux dispositifs d'interaction voyaient le jour.

Ces travaux ont permis de réaliser plusieurs IHM3 et ont montré l'efficacité de telles interfaces. Cependant le développement est souvent empirique et ne suit que rarement une méthode ou un processus identifiés. En effet, les travaux réalisés par la communauté des IHM dans le domaine particulier de la multimodalité, manquent de rigueur, de formalismes, de modèles permettant de raisonner sur la description des systèmes multimodaux et sur leurs propriétés et la validité de ces propriétés. Ainsi, le développement de ces IHM3 n'est pas reproductible ni réutilisable contrairement aux savoir-faire acquis par la communauté des IHM dans le cas des interfaces homme-machine classiques de type WIMP.

Dans d'autres domaines de l'informatique comme les télécommunications ou le développement de systèmes embarqués où la description de systèmes complexes mettant en jeu la séquence, le parallélisme et la synchronisation ont été étudiés et plusieurs travaux en modélisation formelle dans ces domaines ont été réalisés. Ils mettent en oeuvre différentes modélisations sémantiques (synchrone/asynchrone, systèmes à états ou à événements, flot de données ou flots de contrôle, etc.). Ces travaux ont permis de proposer des modélisations formelles qui ont permis la conception et la validation de plusieurs systèmes informatiques dans ces domaines.

Notre démarche propose de tirer le bénéfice de ces travaux afin de reproduire le savoir-faire acquis dans ces domaines au profit de la conception et de la validation formelles de systèmes interactifs multimodaux en entrée. Nous nous proposons d'utiliser différents concepts de la modélisation formelle mis en oeuvre dans ces domaines pour traiter des problèmes de modélisation des IHM3.

Ainsi, notre travail consiste à définir un modèle formel de conception de l'interaction multimodale en entrée et un modèle formel d'expression des propriétés d'utilisabilité de ces systèmes interactifs en utilisant les techniques de modélisation formelles classiquement utilisées dans d'autres secteurs de l'ingénierie du logiciel. La figure 1 positionne notre modèle formel de conception de l'interaction multimodale en entrée, par rapport à un système multimodal.

Dans ce contexte, deux démarches de conception peuvent être envisagées : utilisation d'une unique technique formelle, utilisation de plusieurs techniques formelles chacune étant mise en oeuvre dans son domaine d'efficience.

Contrairement aux autres travaux qui utilisent plusieurs formalismes, nous proposons de définir puis d'utiliser un cadre méthodologique formel générique permettant de définir le modèle du système de l'interaction multimodale ainsi que les propriétés d'utilisabilité associées.

L'intérêt d'un tel cadre est double. D'une part, il offre un aspect homogène

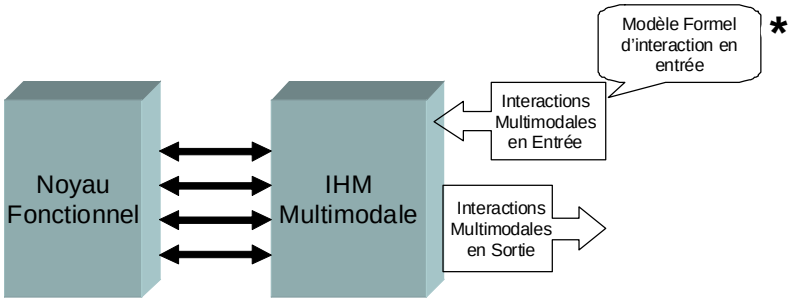

Fig. 1 – Système multimodal

de description de modèles utilisés pour le système et d'expression des propriétés. Ce cadre est fondé sur la définition du système, de ses propriétés et des tâches utilisateurs associées le tout ayant pour base les systèmes de transitions étiquetées. D'autre part, il offre la possibilité de mettre en oeuvre plusieurs techniques formelles différentes codant les descriptions issues du cadre générique. Les modèles que nous proposons sont génériques et indépendants de toute technique formelle. Ils exploitent les travaux réalisés par les chercheurs de la communauté des interfaces multimodales.

Pour la modélisation de l'interaction, nous avons choisi une des classifications des IHM multimodales définies pour offrir un espace de conception. Pour les propriétés d'utilisabilité, nous avons repris les propriétés CARE (Complémentarité, Assignation, Redondance et Equivalence). Notre objectif n'est pas de définir de nouveaux modèles pour la multimodalité, mais de proposer un cadre générique et opérationnel de formalisation des modèles proposés par les chercheurs du domaine de la multimodalité. Le résultat d'un tel travail est l'apport de plus de rigueur aux modèles et au développement des IHM multimodales due à la formalisation. Il apporte également un cadre pour la réutilisation et l'abstraction.

Afin de justifier notre démarche, nous montrons également comment les modèles issus du cadre générique sont mis en oeuvre dans différentes techniques formelles. La première technique formelle utilisée est celle du model-checking. Nous effectuons deux mises en oeuvre dans cette technique. La première utilise une modélisation à base de variables d'états et de la logique temporelle arborescente CTL pour l'expression des modèles de propriétés en utilisant l'outil SMV. La seconde mise en oeuvre utilise une modélisation à base d'actions et d'une logique temporelle linéaire LTL. Le système est exprimé en langage Promela et l'outil SPIN est utilisé pour la vérification des propriétés. Le modèle de l'interaction est exprimé dans le langage d'entrée correspondant à chacun des outils alors que les propriétés sont exprimées par des expressions de logique temporelle.

La seconde catégorie de technique est fondée sur la preuve interactive et sur le raffinement. Elle met en oeuvre la méthode B dans sa version événementielle. Dans ce cas le modèle est exprimé par un ensemble d'événements exprimant des changements d'états et les propriétés sont validées suivant une approche opérationnelle de

description.

Au travers de ces différentes utilisations, nous montrons comment le cadre méthodologique formel et générique proposé permet de transposer aussi bien les modélisations de systèmes multimodaux que les propriétés d'utilisabilité et tâches utilisateurs associées. Nous nous efforçons de démontrer l'intérêt d'un tel cadre comme moyen d'unifier les modélisations de systèmes multimodaux même si différentes techniques formelles sont mises en oeuvre.

Plan du mémoire

Nous avons structuré notre mémoire de thèse en cinq chapitres. Nous présentons dans le premier chapitre, une description des interfaces multimodales qui constituent le type de systèmes que nous abordons dans notre travail. Après une présentation des concepts liés à la multimodalité, nous présentons les principales classifications des IHM multimodales définies par les chercheurs de ce domaine. Nous présentons également les propriétés d'utilisabilité caractérisant ce type d'interface. Les différents modèles intervenant dans la conception et le développement des systèmes interactifs sont également abordés dans ce chapitre. Nous mettons en évidence l'absence de formalisme rigoureux pour le développement et la prise en charge des caractéristiques des systèmes interactifs multimodaux.

Dans le deuxième chapitre nous effectuons un panorama de techniques formelles et de leurs applications dans le développement de systèmes interactifs. Nous mettons en évidence que peu de travaux ont été réalisés pour le développement de systèmes interactifs multimodaux. Nous relevons également que le peu de travaux disponibles manquent de généricité et ne prennent pas en compte les modèles informels, tels que les espaces de conception, déjà définis par les chercheurs dans le domaine des IHM multimodales.

Notre contribution est abordée dans le troisième chapitre. Nous présentons un cadre méthodologique formel et générique de conception de l'interaction multimodale en entrée en respectant les classes d'IHM multimodales définies dans les espaces de conception proposés par la communauté des chercheurs de ce domaine. Nous présentons également un cadre méthodologique formel générique pour les propriétés d'utilisabilité définies par les chercheurs dans ce domaine. Dans les deux cas, ce cadre est formel et générique et indépendant de toute technique formelle.

Dans le quatrième chapitre, nous présentons une mise en oeuvre du cadre méthodologique proposé dans la technique de model-checking. Les modèles de propriétés d'utilisabilité CARE sont exprimés dans une logique temporelle, et le modèle de l'interaction multimodale est décrit dans un langage d'entrée permettant de décrire les systèmes de transitions. Nous avons exprimé nos modèles dans les deux outils SPIN et SMV. Dans ces deux outils, les propriétés sont exprimées dans une logique

temporelle et les systèmes de transitions du modèle de l'interaction sont décrits dans un langage d'entrée. Un sous ensemble des propriétés définies dans le modèle formel du chapitre 3 est exprimé dans les deux logiques temporelles associées à chacun de ces outils.

Enfin, le cinquième chapitre décrit la mise en oeuvre du cadre méthodologique proposé dans une technique de preuve. Le système de transitions décrivant l'interaction multimodale ainsi que celui des propriétés sont décrits dans le langage B évènementiel. La vérification est réalisée à l'aide des raffinements de la méthode B en utilisant une méthode opérationnelle de description des propriétés.

Nous terminons ce mémoire de thèse par une conclusion et une présentation des perspectives dégagées à l'issue de nos travaux.

Chapitre 1

Les Interfaces Homme-Machine Multimodales

1.1 Introduction

Un système est qualifié d'interactif lorsqu'il interagit avec son utilisateur en entrée et/ou en sortie. Il est doté en général d'une interface qui permet de gérer les interactions de l'utilisateur vers le système et/ou les interactions venant du système vers l'utilisateur. Les applications informatiques sont dans la plupart des cas (97% selon [Myers, 1995]) considérés comme des systèmes interactifs qui communiquent avec l'utilisateur à travers une interface homme-machine (IHM). Les fonctionnalités du système constituent dans ce cas le noyau fonctionnel.

A l'origine, les IHM étaient basées sur un dialogue question-réponse avec un langage de commande, puis des interfaces avec des grilles de saisie. Ensuite, il y'a eu apparition de l'approche multifenêtrage avec une manipulation d'objets représentés graphiquement. Ces dernières sont aussi dites interfaces WIMP (Windows, Icons, menus, Poiting devices). Ces interfaces ont offert beaucoup de flexibilité à l'utilisateur et sont utilisées dans la majorité des applications. Ensuite, et avec le développement de domaines de recherches tels que la synthèse de la parole, la vision par ordinateur et la synthèse d'image, les nouveaux dispositifs d'interaction issus de ces domaines ont conduit à la naissance des IHM multimodales. L'objectif de ces interfaces est d'approcher la qualité de communication entre l'utilisateur et la machine du niveau de la communication humaine en offrant plus de choix dans l'interaction homme-machine. Ceci permettra le developpement de systèmes plus adaptés aux personnes handicapées moteurs. De même, ce type d'IHM est trés utile pour les systèmes de sécurité, en permettant la continuité de l'interaction en cas de panne d'un dispositif d'interaction.

Le mot "multimodalité" est composé du préfixe "multi" qui désigne la multiplicité et le mot "modalité" qui désigne une technique ou une manière d'interagir

avec la machine. Cette multiplicité est liée à la variété des canaux d'interaction mis à la disposition de l'utilisateur et aux différentes manières d'utiliser ces canaux. La multimodalité est un thème de recherche au sein du domaine de l'interaction Homme-Machine assez récent. Richard Bolt [Bolt, 1980] a été le premier à proposer l'intégration de deux modalités d'interaction, la parole et le geste avec le paradigme "met ça là". Depuis, plusieurs travaux ont succédé couvrant plusieurs aspects tels que la fixation des concepts et le développement de spécifications, d'outils logiciels et les méthodologies d'évaluation appropriées.

Un des objectifs des recherches dans le domaine des IHM multimodales est d'augmenter les capacités communicatives de l'ordinateur, en étudiant et en s'inspirant de la multimodalité de la communication humaine pour améliorer la communication entre les utilisateurs et les ordinateurs. De même, plusieurs avancés conceptuelles ont été réalisées, que ce soit sur la caractérisation de l'usage des modalités ou sur la spécification de mécanismes de fusion de données issues de modalités distinctes.

1.2 Concepts de base

Plusieurs concepts liés à la multimodalité ont été définis par la communauté des IHM, mais un consensus sur leur définition reste absent. Ces notions sont soit abordées par rapport à l'individu et dans ce cas elles sont dites centrées utilisateurs, soit par rapport à la technologie, et dans ce cas elles sont dites centrées système. On trouve une synthèse des différentes définitions dans [Nigay & Coutaz, 1996]. Nous reprenons ici quelques définitions caractérisant ce type d'IHM en se basant sur cette synthèse et sur les travaux de [Bellik, 1995].

1.2.1 Média

En général le terme média désigne un support technique de l'information. De là, plusieurs définitions ont été données selon plusieurs niveaux d'abstraction allant du niveau purement physique à des niveaux logiques plus élevés. Pour certains, le média est vu comme un dispositif physique ayant le rôle de capteur ou effecteur dans un système informatique. Pour d'autres [Blattner & Dannenberg, 1990], un média n'est pas nécessairement lié uniquement au matériel mais peut être réalisé par un logiciel tel qu'un message électronique par exemple. Pour [Frohlich, 1991], un média est un système représentationnel nécessitant une interprétation et une compréhension tel qu'un graphe ou un texte en langage naturel.

1.2.2 Mode

Le concept de mode est lié au concept de modalité. En effet, le mode désigne la manière avec laquelle se fait une action et la modalité désigne sa forme. Dans

le domaine des IHM, le mode correspond à l'état dans lequel se trouve le système interactif à un instant donné. Il correspond au contexte qui détermine l'interprétation des actions.

Foley et al.[Foley *et al.*, 1984] définissent le mode par un état ou une collection d'états dans lesquels seul un sous-ensemble de toutes les tâches interactives possibles est disponible.

[Frohlich, 1991] définit les modes de l'interface par des états à travers lesquels différentes actions de l'utilisateur peuvent produire les mêmes effets. Ainsi, il définit deux modes : langage et Action. Dans le mode action les opérations sont effectuées en s'engageant dans des actions physiques tandis que dans le mode langage, les opérations sont effectuées en utilisant des activités langagières.

1.2.3 Canal

Le concept de canal est lié au concept de média. Il est défini dans [Foley *et al.*, 1984] par "une interface qui opère une transformation d'énergie". Il fait, ainsi, correspondre à chaque capteur ou effecteur définissant un canal de communication humain, les dispositifs physiques représentant les canaux de l'interface du système chargés de la transformation d'énergie. Par exemple, pour le canal de l'interface 'audio', le dispositif correspondant est un microphone.

1.2.4 Modalité

Une modalité est une manière d'interagir avec le système selon une technique. Comme pour le concept de *média*, la modalité est définie selon plusieurs approches : approche orientée utilisateur, approche système ou approche qui mixe les deux.

- Pour l'approche utilisateur, Bersen [Bersen, 1994] la définit comme un système représentationnel et la confond ainsi avec le concept média tel qu'il est défini par Alty [Alty, 1991].
- Dans l'approche système nous trouvons les définitions de [Martin, 1995], de [Nigay & Coutaz, 1996] et [Bourguet, 1992]. Ces définitions lient la modalité au contenu et à la nature des informations que le système est capable de traiter.
- L'approche mixte est représentée par le modèle 'syndésique' de Duke [Duke *et al.*, 1994]. Il définit une modalité par les capacités sensorielles et les dispositifs physiques ou logiques engagés dans l'interaction.

Par exemple, [Martin, 1995] définit une modalité comme un processus informatique d'analyse ou de synthèse qui fait coopérer plusieurs processus d'analyse et/ou plusieurs processus de synthèse. Pour [Nigay & Coutaz, 1996], une modalité est définie par le couple (d, l) où d désigne le dispositif physique et l, un langage d'interaction. Un langage d'interaction est défini par un vocabulaire d'éléments terminaux et une grammaire. Les éléments terminaux sont produits ou captés par les dispositifs

d'entrée/sortie. C'est une définition qui caractérise les échanges entre le système et l'utilisateur car elle identifie et met en relation deux niveaux d'abstraction : le niveau physique (dispositif) et le niveau logique (langage d'interaction).

1.2.5 Fusion et Fission

La *fusion* consiste à combiner les différentes informations provenant de plusieurs médias ou modalités pour former de nouvelles unités d'information. La fusion peut intervenir à différents niveaux d'abstraction. L'exemple le plus illustratif est la fusion de la commande orale 'met ça là' avec les clics de la souris.

La *fission* est l'opération inverse de la fusion. Elle consiste à éclater une commande en plusieurs commandes. Un exemple qui explique la fission est la commande 'dessiner un cercle dans une fenêtre'. La réalisation de cette commande nécessite son éclatement en deux commandes. La première commande est 'créer fenêtre', et elle est destinée au gestionnaire des fenêtres. La deuxième commande est 'créer cercle', et elle est destinée à l'éditeur graphique.

En général, l'opération de fusion concerne les interactions en entrée, tandis que l'opération de fission peut concerner aussi bien les interactions en entrée qu'en sortie.

1.2.6 Enoncé

Un énoncé est constitué d'une suite séquentielle d'événements élémentaires multimodaux intervenant dans la réalisation d'une même commande. Une commande est une tâche élémentaire que l'utilisateur veut réaliser. Un énoncé est dit "multimodal" s'il contient au moins un événement multimodal et au moins deux événements élémentaires provenant de deux médias distincts. Un événement multimodal est constitué d'un ensemble d'événements élémentaires provenant de médias distincts, produits dans des voisinages temporels proches et intervenant dans la réalisation d'une même commande. Un événement élémentaire est l'information de base produite par l'interface logicielle associée à un média. La phrase "met ça là" avec un clic de la souris constitue un énoncé multimodal.

1.3 Classes de systèmes multimodaux

La présence de plusieurs modalités permet à l'utilisateur d'interagir de différentes manières avec le système. Il peut utiliser ces modalités en séquence ou en parallèle. Selon les possibilités permises pour combiner ces modalités, plusieurs classes d'IHM sont identifiées. Pour cela, plusieurs classifications ont été proposées par la communauté des IHM. Ces classifications offrent des espaces de conception aux concepteurs sur lesquels ils peuvent se baser pour concevoir des IHM de tel ou tel autre type. Ils

définissent les contraintes à respecter pour l'utilisation des modalités. Nous présentons ici l'espace CASE, raffiné par la suite dans [Bellik, 1995] et que nous utilisons comme base pour notre modèle formel de conception de l'interaction multimodale présenté dans le chapitre 3.

1.3.1 Espace CASE

Certains travaux ont établi les classes d'IHM selon les types de multimodalité qu'elles permettent de traiter. [Caelen & Coutaz, 1991] et [Nigay & Coutaz, 1993a] distinguent quatre classes en se basant sur les propriétés CASE (Concurrente, Alternée, Synergique et Exclusive). [Caelen & Coutaz, 1991] déduit ces classes de la combinaison des deux critères :

1. *l'usage des médias* qui peut être *séquentiel* ou *parallèle*.
2. *la manière avec laquelle les données sont interprétées*, qui peut être *indépendante* ou *combinée*.

[Nigay & Coutaz, 1993a] utilise trois critères.

1. *Fusion :* deux valeurs sont possibles pour ce critère. Dans le cas où l'opération de fusion n'est pas indispensable on dit que les modalités sont indépendantes, et dans le cas où l'opération de fusion est nécessaire, on dit que les modalités sont combinés.
2. *Usage des modalités* : ce critère désigne l'usage des modalités dans le temps. En effet, elle peuvent être utilisées de manière séquentielle ou parallèle (en même temps).
3. *Niveau d'abstraction* : deux niveaux d'abstraction ont été retenus : capacité du système à extraire du sens et est appelé 'sens' et son inverse est appelé 'pas de sens'. Les systèmes multimodaux sont des systèmes ayant la capacité à extraire du sens.

La combinaison de ces critères a donné naissance à quatre classes d'IHM multimodales ou de multimodalité : La figure 1.1 montre l'espace CASE. Il correspond aux classes d'IHM multimodales ce qui signifie que la valeur du critère niveau d'abstraction est égale à "sens".

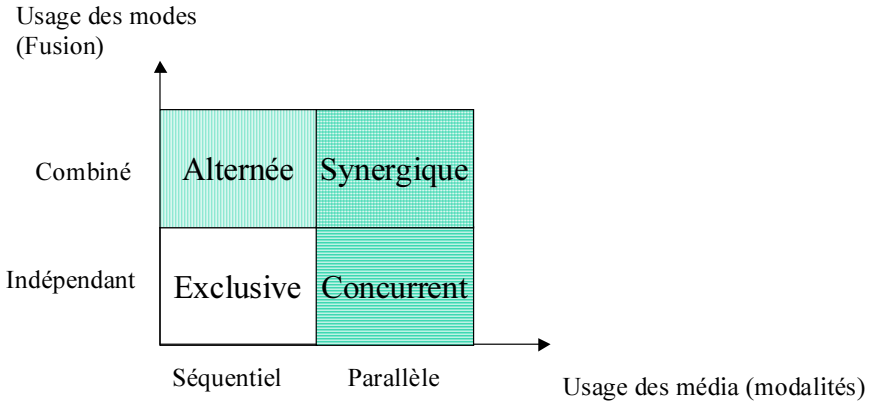

Fig. 1.1 – Espace CASE

1. **La multimodalité concurrente :** les énoncés sont produits en même temps en utilisant des modalités différentes.
2. **La multimodalité alternée :** plusieurs modalités peuvent être utilisées pour réaliser une tâche donnée, mais à un instant donné, une seule modalité est utilisée.
3. **La multimodalité synergique :** plusieurs modalités sont utilisées pour un même énoncé et en même temps.
4. **La multimodalité exclusive :** une seule modalité est utilisée par tâche, et à une instant donné, une seule modalité est utilisée.

1.3.2 Espace CASE raffiné

L'espace proposé dans [Bellik, 1995] raffine l'espace précédent en distinguant le parallélisme entre les médias et le parallélisme dans l'expression des énoncés. Ainsi, il repose sur trois critères.

1. *Production des énoncés :* la production des énoncés peut être en séquence ou d'une manière indépendante et parallèle.
2. *L'usage des médias :* les médias peuvent être utilisés de manière exclusive, et dans ce cas un seul média est actif à un instant donné, ou bien d'une manière parallèle et dans ce cas plusieurs médias peuvent être actifs en même temps.
3. *Le nombre de média par énoncé :* un énoncé peut être produit par un seul média ou plusieurs médias. Dans le cas où plusieurs médias sont utilisés en entrée, il est nécessaire de procéder à la fusion des différentes informations provenant des différents médias.

Les combinaisons de ces trois paramètres ont donné naissance à sept classes de multimodalité. Certaines classes sont omises car impossibles à réaliser. Par exemple,

le cas où la production des énoncés est séquentielle et l'usage des médias est simultané.

Nous illustrons chaque classe par un graphique représentant en abscisse le temps, et en ordonnée la modalité utilisée parmi la *parole*, la *manipulation directe* et le *geste*. Les valeurs correspondent aux interactions réalisées. Dans les définitions suivantes nous confondons média et modalité.

1. **La multimodalité exclusive :** les énoncés sont produits de manière séquentielle, et chaque énoncé est exprimé à l'aide d'un seul média (voir figure 1.2).

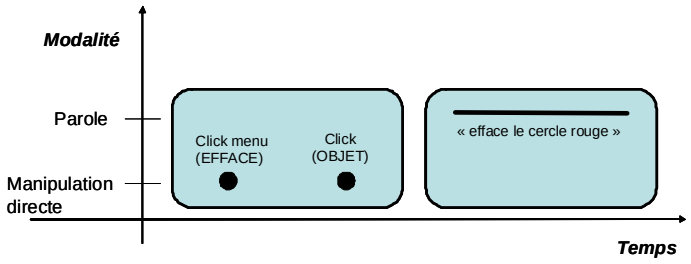

Fig. 1.2 – Exemple de multimodalité exclusive

2. **La multimodalité alternée :** les énoncés sont produits de manière séquentielle, et chaque énoncé peut être produit par plusieurs médias. Les médias dans ce cas sont utilisés de manière alternée (voir figure 1.3).

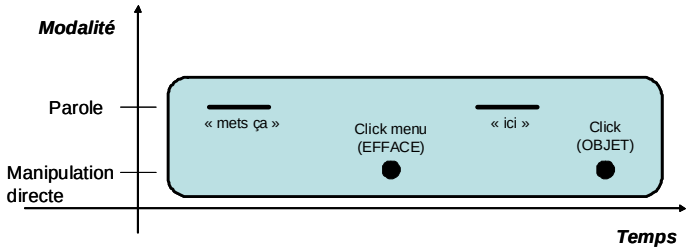

Fig. 1.3 – Exemple de multimodalité alternée

3. **La multimodalité synergique :** les énoncés sont produits de manière séquentielle, et plusieurs médias peuvent participer dans la production d'un énoncé. Les médias dans ce cas peuvent être utilisés simultanément (voir figure 1.4).

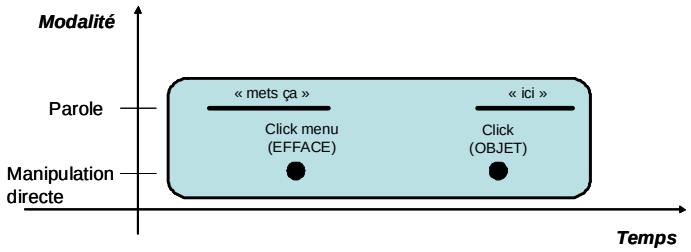

Fig. 1.4 – Exemple de multimodalité synergique

4. **La multimodalité parallèle exclusive :** les énoncés peuvent être produits en parallèle, mais chaque énoncé ne peut être produit qu'avec un seul média. A un instant donné, un seul média est utilisé exclusivement (voir figure 1.5).

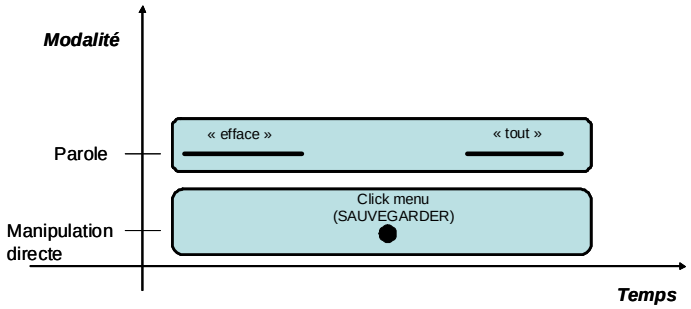

Fig. 1.5 – Exemple de multimodalité parallèle exclusive

5. **La multimodalité parallèle simultanée :** les énoncés peuvent être produits de manière parallèle, et chaque énoncé ne peut être produit que par un seul média. Contrairement à la multimodalité parallèle exclusive, plusieurs médias peuvent être utilisés simultanément (voir figure 1.6).

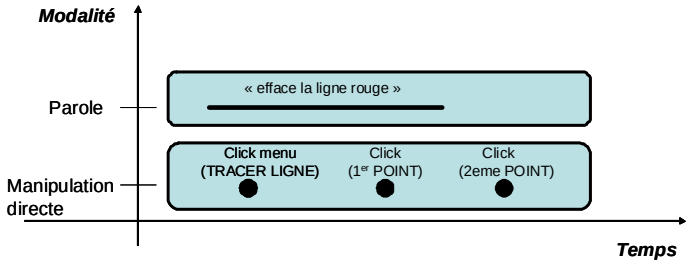

Fig. 1.6 – Exemple de multimodalité parallèle simultanée

6. **La multimodalité parallèle alternée :** les énoncés peuvent être produits en parallèle. Chaque énoncé peut être produit par plusieurs médias, mais à un instant donné un seul média est utilisé (voir figure 1.7).

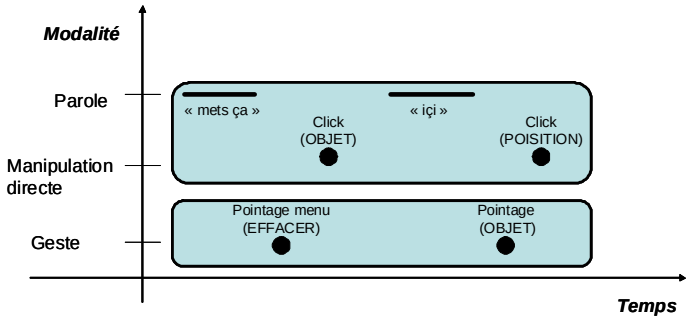

Fig. 1.7 – Exemple de multimodalité parallèle alternée

7. **La multimoldalité parallèle synergique :** les énoncés peuvent être produits en parallèle. Chaque énoncé peut être produit par plusieurs médias, et plusieurs médias peuvent être utilisés en même temps (voir figure 1.8).

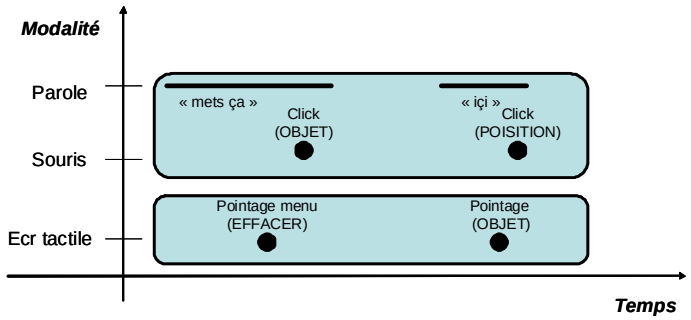

Fig. 1.8 – Exemple de multimodalité parallèle synergique

Nous avons choisi cette dernière classification comme modèle de base pour la proposition de notre modèle formel de conception de l'interaction multimodale présenté dans le chapitre 3. En effet, le degré de raffinement du parallélisme dans cette classification permet de distinguer plusieurs niveaux de parallélisme : parallélisme entre les énoncés et parallélisme entre les tâches. Ceci permet plus de généricité du modèle du moment que les classes CASE y sont incluses. De plus, l'expression et la validation de propriétés sur ce type de classification deviennent possibles.

1.4 Modèles de développement des systèmes interactifs

Une modélisation du processus de développement de logiciels est nécessaire pour offrir aux concepteurs d'IHM un schéma générique sur lequel ils se basent pour procéder au développement des systèmes informatiques. Les systèmes multimodaux sont des systèmes interactifs, et de ce fait il est important d'étudier les modèles de développement pour ce type de systèmes. Actuellement, il n'existe aucun modèle

spécifique pour ce type de système, et les modèles utilisés sont des adaptations de ceux développés pour les systèmes interactifs en général.

Les modèles de développement de logiciels ont été définis et étendus pour prendre en charge les systèmes interactifs. Le modèle en *cascade* fut le premier modèle à être utilisé par Royce [Royce, 1970]. Il fut raffiné par la suite sous diverses formes avec le modèle en *V* [McDermid & Ripkin, 1984] puis, le modèle en *spirale* [Boehm, 1988].

Nous avons choisi de présenter le modèle en V (voir figure 1.9) [Balbo, 1994] qui explicite et structure les activités de tests. Parmi ces activités, la validation et la vérification sont différenciées [Boehm, 1981] :

- La *validation* s'interroge sur l'adéquation du logiciel produit : "construisons-nous le bon produit ?". L'adéquation n'est pas une caractéristique mesurable mais relève d'un jugement subjectif. Le génie logiciel rejoint ici les constatations de l'ergonomie sur la pertinence des données qualitatives.
- La *vérification* concerne la conformité du produit avec une description de référence : "construisons-nous le produit correctement ?". La vérification n'a de sens que s'il existe un document de référence, par exemple, un dossier de spécification détaillée ou cahier des charges.

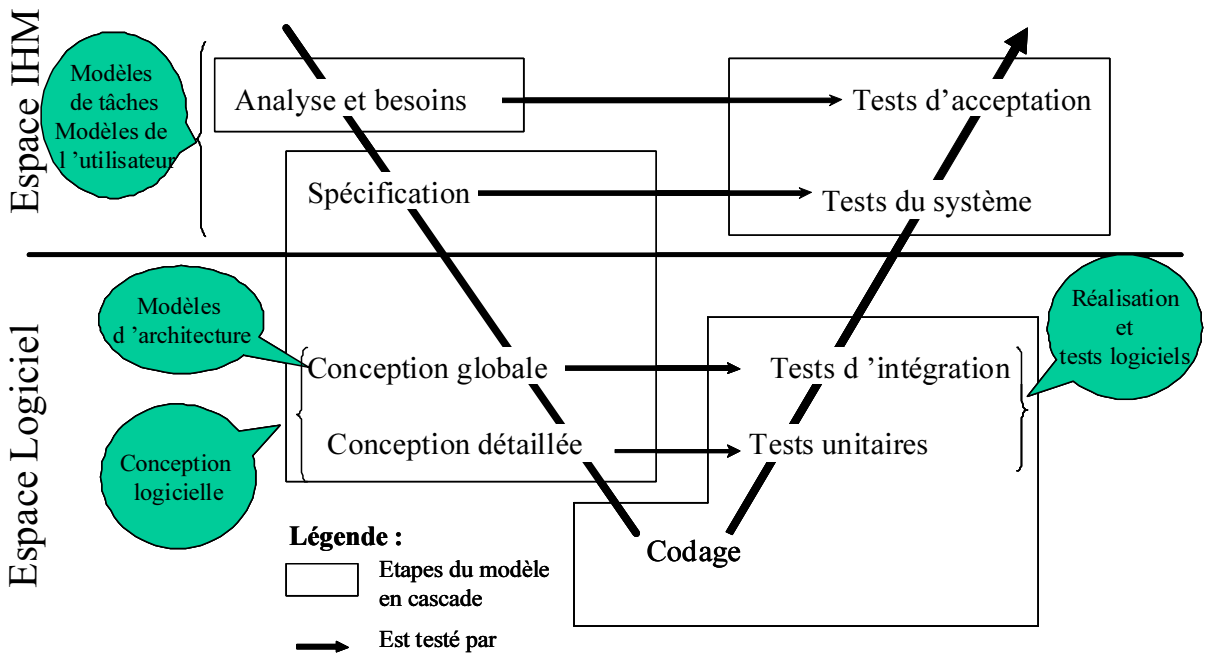

Fig. 1.9 – Le modèle en V

La pente descendante du modèle V (figure 1.9) comprend les étapes : Analyse des besoins, conception, conception globale, ensuite conception détaillée avant le codage. A chaque étape correspond un ensemble de tests qui permettent de vérifier

et/ou valider l'étape. En général, les tests sont effectués en séquence une fois le codage réalisé, mais ils doivent être spécifiés en même temps que leur correspondant dans la pente descendante.

Sur la figure 1.9, on constate que les tests d'acceptation correspondent à l'étape de définition et de spécifications des besoins, que les tests du système doivent être définis en même temps que la conception, etc.

- Les *tests unitaires* permettent de vérifier que les composants modulaires du système répondent chacun à leur spécification.
- Les *tests d'intégration* servent à vérifier que les modules réalisés indépendamment interagissent correctement.
- Les *tests du système* servent à vérifier que les éléments de la solution exprimés dans le dossier de spécifications externes sont présents.
- Les *tests d'acceptation* servent à vérifier que les besoins exprimés dans le cahier des charges du logiciel sont couverts. C'est le dernier contrôle avant la livraison du produit.

Le modèle en V permet de prendre en considération les exigences de l'utilisateur dès les premières phases de conception du logiciel et en particulier de l'IHM. Sur la figure 1.9 représentant le modèle V, les étapes du processus de développement sont réparties en deux espaces : l'espace IHM et l'espace logiciel. Le premier se caractérise par la priorité qu'il accorde aux aspects ergonomiques, le second par l'accent mis sur les techniques d'implémentation logicielles.

L'espace logiciel laisse la place aux compétences informatiques avec les conceptions globales et détaillées, le codage et les tests unitaires et d'intégration.

L'espace IHM inclut l'analyse des besoins, la spécification et les tests du système et d'acceptation. De nombreux modèles et notations sont définis pour chacune de ces étapes. Au niveau de cet espace, les modèles de tâches et les modèles de l'utilisateur interviennent. A l'origine, seuls les aspects techniques du système étaient considérés et faisaient que les utilisateurs avaient des problèmes pour utiliser ces systèmes afin de réaliser leurs tâches. C'est pourquoi, les modèles de tâches ont été définis dans le but de modéliser les tâches utilisateurs afin de pouvoir les valider lors de la conception du système. Ces modèles viennent en complément des modèles de développement du génie logiciel traditionnel.

Plusieurs travaux ont été réalisés par la communauté des ergonomes et des psychologues pour l'analyse des tâches et la modélisation de l'utilisateur à cette étape (analyse des besoins et spécification). L'utilisabilité du système est liée à cet espace (espace IHM) où les vérifications restent basées sur l'expérimentation et l'utilisation des méthodes empiriques

Au niveau conception globale, les modèles d'architectures offrent un schéma global des différents composants du système interactif.

Nous présentons dans ce qui suit une description des modèles de tâches et d'architectures utilisés dans la conception des systèmes interactifs. Nous signalons l'absence quasi totale de modèles spécifiques aux systèmes multimodaux en dehors de la tentative de [Rousseau *et al.*, 2004]. Les travaux actuels utilisent les mêmes modèles développés et utilisés pour les systèmes interactifs avec des IHM de type WIMP.

Nous nous focalisons sur les modèles des tâches et d'architecture car ces modèles ont fait l'objet de nombreuses propositions pour le développement d'IHM. Des modèles spécifiques prenant en compte les spécificités des IHM ont été définis. Ils complètent et enrichissent les modèles traditionnellement mis en oeuvre en génie logiciel.

1.5 Modèles d'architectures pour les systèmes interactifs

Les modèles d'architecture définissent l'organisation logicielle d'un système logiciel. Ils définissent les éléments directeurs et structurants de l'organisation d'un logiciel. Dans le cas des systèmes interactifs, tous les modèles ont comme principe, la séparation entre le noyau fonctionnel qui implémente les concepts propres à un domaine d'application particulier et l'interface qui présente ces concepts à l'utilisateur et qui lui permet de les manipuler et d'interagir avec le système. L'objectif est de permettre la modification de l'interface sans affecter le noyau fonctionnel et inversement. Les modèles d'architecture permettent non seulement de structurer le système interactif mais de disposer d'une meilleure modularité. Cette modularité facilite la réutilisation logicielle des composants, sa maintenance, son adaptation au contexte et son évolution. Un modèle d'architecture doit :

- préconiser une séparation fonctionnelle entre les services de l'application et ceux de l'interface ;
- définir une répartition des services de l'interface, qui se traduit par un ensemble de composants logiciels ;
- définir un protocole d'échange entre les constituants logiciels qu'il identifie.

Au cours du temps, différents modèles d'architectures ont émergé. Ils ont souvent été classés en trois classes dans la littérature.

1.5.1 Les modèles globaux

Ils définissent le nombre, la nature et l'organisation des différents modules constituant le modèle, sans cependant discuter de leur structure interne. Ils sont souvent nommés Macro-Modèle, modèles centralisés ou modèle généraux [Fekete, 1996]. Nous citons dans cette classe le modèle Seeheim proposé au cours du séminaire sur les systèmes de gestion utilisateur en 1983 à Seeheim (R.F.A.) [Pfaff, 1985], le modèle

ARCH une extension du modèle Seeheim défini au cours des quatre séminaires regroupant des développeurs de gestionnaires d'interface utilisateur [Bass *et al.*, 1991] [Paterno *et al.*, 1992]. Ce sont des modèles conceptuels : ils identifient les modules abstraits devant apparaître dans toute application, mais ne définissent ni la structure interne de ces modules, ni les interfaces d'échange.

1.5.2 Les modèles multi-agent

A l'inverse des modèles généraux, les modèles multi-agent définissent de manière précise les composants de base de l'application ainsi que la communication entre composants, mais sans préciser leur nombre ou leur organisation. Ces modèles sont également nommés modèles génériques [Fekete, 1996]. Nous citons dans cette classe le modèle MVC utilisé dans l'architecture de Smalltalk [Goldberg, 1984] et le modèle PAC proposé par J. Coutaz [Coutaz, 1987]. Les modèles multi-agents permettent de représenter, à un niveau de granularité beaucoup plus faible que les modèles globaux, la séparation entre les fonctions identifiées et la présentation.

1.5.3 Les modèles à base d'interacteurs

Les modèles à base d'interacteurs sont des modèles multi-agents qui en plus des services offerts par ce type de modèles, offrent des descriptions précises des différents événements manipulés et de leur composition. De plus, ces modèles ont été formalisés dans différentes techniques formelles. Nous citons les modèles de York [Duke & Harrison, 1993], de Pise [Paterno & Faconti, 1992], de Cnuce [Faconti & Paterno, 1990] et d'ADC [Markopoulos, 1995].

1.5.4 Les Modèles hybrides

Ils utilisent les modèles globaux pour déterminer les composants et les modules, et les modèles multi-agents pour détailler la structure interne des modules. Nous citons principalement le modèle PAC-Amodeus [Coutaz, 1990] qui adopte les mêmes composants que le modèle ARCH, mais détaille le composant contrôleur de dialogue. Ce dernier est décomposé en un ensemble d'agents coopératifs de type PAC. La structure des agents autorise l'emploi de techniques de fusion/fission pour les dialogues multimodaux [Nigay & Coutaz, 1993b], tel qu'implémenté dans Matis [Nigay, 1994].

1.6 Modèles de tâches pour les systèmes interactifs

On définit par le mot *tâche*, un travail déterminé qu'on doit réaliser [Robert, 1992]. Dans le cas de l'interaction Homme-Machine, on peut définir une *tâche* comme un objectif à atteindre par l'utilisateur à l'aide d'un système interactif [Normand, 1992].

Un ensemble de concepts liés à la définition d'une tâche ont été définis et sont résumés comme suit [Balbo, 1994] :

- une *tâche* représente un but que l'utilisateur veut atteindre avec une procédure décrivant les moyens et la manière de l'atteindre ;
- un *but* est un état du système que l'utilisateur souhaite obtenir. Dans le cas où l'interface du système n'est pas honnête, l'état réel du système risque d'être différent de l'état perçu par l'utilisateur ;
- une *procédure* est définie par un ensemble d'opérations reliées par des relations temporelles et structurées ;
- une *action* désigne une opération terminale ; elle intervient dans l'accomplissement d'un but terminal ;
- une *opération* est une action ou une tâche ;
- un *but terminal* et les actions qui permettent de l'atteindre définissent une tâche élémentaire ;
- une *procédure élémentaire* est la procédure associée à une tâche élémentaire ;
- les *relations temporelles* entre les opérations d'une procédure traduisent la séquentialité, l'interruptabilité ou le parallélisme ;
- les *relations structurelles* servent à exprimer la composition logique des opérations ou la possibilité de choix ;
- une *tâche composée* est une tâche non élémentaire. Elle inclut sans sa description des sous tâches.

Reprise de [Balbo, 1994], la figure 1.10 représente l'ensemble de ces concepts ainsi que les relations qui les lient.

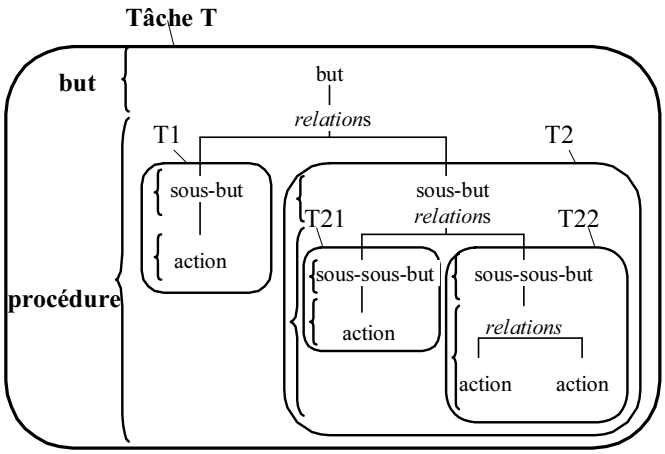

Fig. 1.10 – Décompositions de tâches

Un rectangle encapsule une tâche **T** définie par un but et une procédure. Les relations sont représentées par une structure arborescente exprimant une décomposition. Dans l'exemple, le but est atteint par l'accomplissement de deux sous tâches **T1** et **T2**. A **T1** correspond un sous-but atteint par une action. **T2** correspond à un

sous-but décomposé en deux autres sous-buts **T21** et **T22**. **T1**, **T21** et **T22** sont des tâches élémentaires. **T** et **T2** sont des tâches composées. Plusieurs modèles et notations ont été développés pour mettre en oeuvre ces concepts et permettre de raisonner sur les tâches utilisateurs et prendre en compte les besoins utilisateurs.

Le niveau de granularité de la tâche est laissé à l'appréciation du concepteur. Ce dernier peut définir les tâches élémentaires entre le niveau de granularité des fonctions des systèmes jusqu'à celui des actions physiques. Cette différence a engendré des points de vue différents chez les développeurs d'IHM. En effet, selon les modèles, méthodes et cycle de vie adoptés, les niveaux de granularité différent. Pour nos travaux, nous ne précisons pas un niveau de granularité particulier, notre proposition est en mesure d'exprimer tous les niveaux.

Plusieurs modèles de tâche ont été définis par la communauté des chercheurs en IHM. Parmi eux, nous pouvons citer UAN [Hartson & Gray, 1992], MAD [Scapin & Pierret-Golbreich, 1989] et CTT et son environnement CTTE [Paternò *et al.*, 2001]. Ces modèles peuvent être utilisés de deux manières.

1. **En guise de modèle de spécification :** où les fonctionnalités d'une IHM sont vues comme un ensemble de tâches que l'utilisateur peut effectuer. Le concepteur commence dans ce cas par spécifier l'IHM par une tâche principale qui sera décomposée en un ensemble de sous tâches qui seront décomposées à leur tour, jusqu'à atteindre les tâches élémentaires réalisées par les actions interactives de base selon le niveau de granularité défini par le concepteur. La spécification est obtenue en composant toutes les tâches principales grâce aux opérateurs de composition disponibles dans le modèle utilisé (ex : composition parallèle, séquentielle ou choix).
2. **En guise de modèle de validation :** où la tâche est modélisée pour vérifier si le système permet de la réaliser ou pas. Le système est représenté d'une part par son modèle éventuellement à partir d'un autre modèle de tâche, et la tâche est modélisée par le modèle de tâche. La validation d'une tâche T consiste à vérifier si son modèle est pris en compte dans le modèle de tâche associé au système représentant l'IHM ou le système interactif. Notons que la validation suivant cette démarche peut porter sur la réfutation d'une tâche. Dans ce cas, la validation porte sur le rejet d'une tâche par l'IHM à valider. On peut alors s'intéresser à l'analyse des conséquences de ce rejet. Ce type de validation est important pour garantir la sécurité dans l'utilisation d'une IHM.

1.6.1 MAD

MAD (Méthode Analytique de Description) [Scapin & Pierret-Golbreich, 1989] est une méthode de conception basée sur une approche psycho-ergonomique. Elle dispose d'une notation graphique permettant de décrire les tâches à l'aide d'une décomposition hiérarchique en utilisant des constructeurs qui jouent le rôle de liens

temporels : la séquence, l'alternative, le parallèle et la simultanéité. C'est une notation qui associe également aux tâches un ensemble d'attributs, de pré et de post-conditions.

1.6.2 UAN

UAN (User Action Notation) est une notation qui permet d'exprimer les spécifications des interfaces à manipulation directe. Elle décrit dans un tableau, d'une manière textuelle : les actions physiques que l'utilisateur exécute (clic de la souris), les retours d'informations fournis par le système et finalement l'état de l'interface. Elle impose au concepteur de se préoccuper de l'interface plutôt que des éléments du noyau fonctionnel. Le tableau 1.1 présente un exemple de description UAN.

Tâche : Déplacer un fichier		
Action Utilisateur	**Retour Information**	**Etat Interface**
$\sim [file\_icon]$ Mv $M\hat{}$	$file\_icon!$	$selected = file$

Tab. 1.1 – Déplacement d'une icône de fichier en UAN

Les actions de l'utilisateur "$\sim [file\_icon]$" et "Mv" signifient respectivement *"déplacer le curseur sur une icône de fichier non sélectionné"* et *"presser le bouton de la souris"*. L'icône de fichier apparaît en surbrillance "$file\_icon!$" et la variable *"selected"* prend la valeur *"file"*. L'action de l'utilisateur "$M\hat{}$" signifie *"relâcher le bouton de la souris"* et complète la tâche.

Cette notation a été utilisée dans les travaux [Coutaz *et al.*, 1993b] pour spécifier l'interface en entrée du système multimodal MATIS.

1.6.3 CTT : ConcurTaskTrees

CTT [Paterno, 2001] est une représentation de l'activité utilisateur. Elle classe les tâches utilisateurs en quatre types (abstraite, utilisateur, interaction et application) et utilise un ensemble d'opérateurs temporels empruntés de l'algèbre de processus Lotos (interruption, concurrence, ...) pour leur composition.

CTT utilise une représentation graphique et propose un outil constituant un environnement d'édition, de simulation, et génération de scenarii de tâches, appelé CTTE (CTTEnvironment) [Paternò *et al.*, 2001].

La figure 1.11 représente le modèle de tâche d'un *distributeur automatique de billet* (DAB). Dans ce modèle l'utilisateur doit insérer sa carte, ensuite (>>) saisir son code PIN pour avoir l'autorisation d'accéder à son compte. La tâche *Accès* peut être effectuée plusieurs fois (*) et désactivée ([>) à tout moment par la tâche *Terminer Accès*. L'utilisateur choisit ([]) ensuite de retirer de l'argent (*Retirer Argent*), de

déposer de l'argent (*Déposer Argent*) ou bien de consulter son compte (*Interrogation Compte*). Seule la tâche *Retirer Argent* est détaillée sur la figure 1.11.

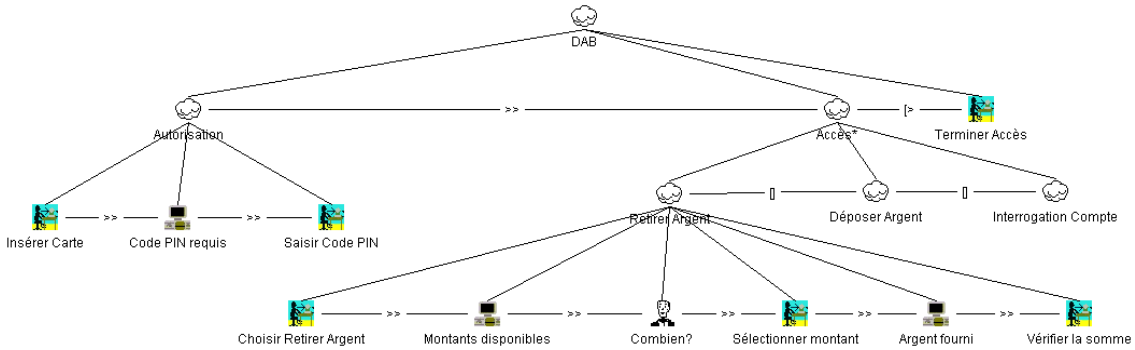

Fig. 1.11 – Exemple de tâche en CTT

Nous remarquons que les modèles de tâches manquent de formalismes et de notations rigoureuses, ce qui provoque souvent une ambiguïté d'interprétation et donc des difficultés de communication entre le concepteur et le développeur du système interactif. Un autre inconvénient est que ces modèles sont indépendants de toute architecture de système interactif ce qui rend difficile leur intégration dans un processus de développement. Le manque de formalisme fait que l'évaluation et la validation des tâches par ces modèles restent empiriques. Le test et l'expérimentation constituent les moyens de valider les tâches utilisateurs.

Enfin, notons que le modèle proposé permet d'intégrer une large catégorie des différentes notations proposées et leur utilisation aussi bien en mode spécification qu'en mode validation.

1.7 Propriétés des IHM

En plus des propriétés classiques de logiciels abordées en génie logiciel, les concepteurs d'IHM traitent de l'utilisabilité. Pour évaluer l'utilisabilité d'une interface homme-machine, un ensemble de propriétés ont été définies. Certaines concernent le fonctionnement du système telles que la propriété de sûreté et d'équité et d'autres sont liées aux exigences des ergonomes et des psychologues. Ces propriétés définissent la capacité d'un système à permettre à l'utilisateur d'effectuer ses tâches avec efficacité, confort et sécurité.

Plusieurs classifications de ces propriétés ont été proposées par les chercheurs de la communauté des IHM [Dix *et al.*, 1993][Gram & Cockton, 1996][Roche, 1998] [Duke & Harrison, 1995]. Nous reprenons la classification présentée dans [Roche, 1998] qui recense deux grandes classes :

- les propriétés de validité concernent le fonctionnement attendu et voulu par un utilisateur ;

- les propriétés de robustesse sont liées à la sûreté du fonctionnement du système.

1.7.1 Propriétés de validité

Dans cette catégorie figurent les propriétés de complétude, de flexibilité pour la représentation de l'information, pour le déroulement des tâches et pour l'adaptation du dialogue avec l'utilisateur.

Propriétés de complétude. [Gram & Cockton, 1996] différencie la complétude des traitements de la complétude des objectifs. Dans le premier cas, on traite de la complétude des traitements si le système est à même d'autoriser l'exécution correcte des traitements qui le composent. On parle cependant de complétude des objectifs si l'utilisateur peut atteindre un objectif donné au moyen du système.

Propriétés de flexibilité. Ces propriétés caractérisent la manière avec laquelle l'utilisateur et le système échangent des informations au moment de l'exécution d'une tâche. Selon [Gram & Cockton, 1996], les propriétés de flexibilité peuvent être classées suivant trois sous catégories : la représentation de l'information, l'adaptation du dialogue et le déroulement des tâches.

Représentation de l'information. Cette sous-catégorie englobe les propriétés liées à la multiplicité des périphériques, à la multiplicité de la représentation et à la réutilisabilité des données d'entrée et de sortie.

- La multiplicité des périphériques est la capacité du système à offrir plusieurs périphériques d'entrées et de sortie pour la communication.
- La multiplicité de la représentation est la capacité du système à offrir plusieurs représentation d'un même concept. Dans le cas d'une horloge, il existe plusieurs formes différentes de représentation : numérique/analogique.
- La réutilisabilité des données d'entrée et de sortie est la capacité du système à autoriser l'usage des entrées et des sorties précédentes comme entrées futures. Dans le cas des sorties du système, la technique du couper-coller est un concept utilisable comme données d'entrée. A l'inverse, les valeurs par défaut comme entrées de l'utilisateur sont réutilisables par le système en sortie.

Adaptation du dialogue. Cette catégorie englobe principalement les propriétés liées à l'adaptativité et à l'adaptabilité.

- L'*adaptativité* est la capacité du système à s'adapter à l'utilisateur sans intervention explicite de la part de ce dernier.
- L'*adaptabilité* ou la reconfigurabilité est la capacité du système à supporter sa personnalisation de la part de l'utilisateur.

Déroulement des tâches. Cette dernière catégorie englobe les propriétés liées à l'atteignabilité, la non-préemption et l'interaction de plusieurs fils.

- L'*atteignabilité* est la capacité du système à permettre à l'utilisateur d'atteindre un état désiré du système à partir de l'état actuel.
- La *non préemption* est la capacité du système à fournir directement le prochain but à l'utilisateur.
- L'*interaction à plusieurs fils* est la capacité du système à gérer plusieurs processus entrelacés, ce qui permet à l'utilisateur de lancer plusieurs traitements en même temps.

1.7.2 Propriétés de robustesse

Les propriétés de robustesse englobent les propriétés liées à la visualisation du système comme l'observabilité, l'insistance et l'honnêteté, et les propriétés liées à la gestion des erreurs comme la prédictibilité et la tolérance aux écarts.

Visualisation du système. Les propriétés liées à la visualisation du système permettent d'assurer une représentation correcte de l'état du système interactif via l'interface.

- L'*observabilité* est la capacité pour l'utilisateur à évaluer l'état interne du système. Le système fait en sorte que toutes les informations disponibles soient visualisées à l'écran.
- L'*insistance* est la capacité du système à forcer la perception de l'état du système. Le système fait en sorte que les informations nécessaires à l'utilisateur soient affichées à l'écran.
- L'*honnêteté* est la capacité à rendre conforme l'état interne du système aux yeux de l'utilisateur.

Gestion des erreurs. Ce sont des propriétés qui caractérisent la capacité du système à recouvrer ou prévenir les erreurs des utilisateurs.

- La *prédictibilité* est la capacité pour l'utilisateur de prévoir les états accessibles du système à partir d'un état courant observable.
- La *tolérance* aux écarts est la capacité du système à aider l'utilisateur lorsqu'une ou plusieurs erreurs se produisent.

La vérification de ces propriétés permet de garantir l'*utilisabilité* du système interactif.

1.7.3 Propriétés d'utilisabilité des IHM multimodales : CARE

Nous assistons ces dernières années à l'apparition de nouveaux types d'interfaces plus complexes en termes d'interaction et de présentation. Pour cette catégorie d'IHM, en plus des propriétés évoquées en sections 1.7.1 et 1.7.2, des propriétés permettant de les caractériser ont été définies.

Les propriétés CARE [Coutaz *et al.*, 1995a], définies initialement dans l'espace TYCOON [Martin & Beroule,] [Martin, 1997], ont servi de base pour définir les propriétés d'utilisabilité des interfaces multimodales. Elles définissent les différentes manières avec lesquelles l'utilisateur peut combiner les modalités pour réaliser ses tâches. Elles sont définies comme suit :

1. **Complémentarité :** désigne l'usage de plusieurs modalités pour la réalisation d'un but.
2. **Assignation :** désigne l'usage exclusif d'une seule modalité pour la réalisation d'un but, et aucune autre modalité ne permet de le réaliser.
3. **Equivalence :** désigne le choix offert à l'utilisateur pour réaliser son but avec une modalité de son choix parmi un ensemble de modalités permettant de le réaliser.
4. **Redondance :** désigne l'usage de plusieurs modalités pour le même but, et d'une manière parallèle.

1.7.3.1 Eléments de base

Dans [Coutaz *et al.*, 1995a], les auteurs définissent les propriétés CARE et donnent une expression formelle pour chacune d'elles. Cette expression formelle repose sur les concepts d'état, de but, de modalité, et de relation temporelle qu'ils définissent comme suit :

– Un *état* est un ensemble d'attributs mesurables à un instant donné et qui caractérisent une situation.

– Un *but* est un état qu'un agent veut atteindre. Un agent peut être le système ou l'utilisateur.

– Une *modalité* est une méthode d'interaction qu'un agent utilise pour atteindre un but. La fonction $Reach(s, m, s')$ exprime la capacité de la modalité m de permettre à un agent d'atteindre l'état s' à partir d'un état s en une seule étape. Une suite d'états successifs est appelé $trajectoire\ d'interaction$.

– Une *relation temporelle* caractérise l'utilisation des modalités à travers le temps. L'utilisation de ces modalités peut être simultanée ou en séquence dans une fenêtre temporelle définie par un intervalle de temps.

1.7.3.2 Comportements séquentiel et parallèle

A partir des notions définies dans la section précédente, Il est possible de définir les comportements parallèle et séquentiel des modalités permettant de décrire les propriétés CARE d'un système multimodal à états.

Soit M un ensemble de modalités. Ces modalités peuvent être utilisées en parallèle si, dans une fenêtre temporelle, elles sont actives en même temps. L'utilisation parallèle des modalités d'un ensemble M dans une fenêtre temporelle tw est définie par :

$$\begin{array}{|l} Parallel(M, tw) \Leftrightarrow \\ (\text{Card(M)}{>}1) \wedge (Duration(tw) \neq \infty) \wedge (\exists t \in tw, \forall m \in M \ Active(m, t)) \end{array}$$

Avec :
- $Active(m, t)$ est un prédicat qui exprime que m est active à l'instant t.
- $Card(M)$ est le nombre de modalités dans l'ensemble M ;
- $Duration(tw)$ est la durée de l'intervalle du temps tw.

Les modalités de l'ensemble M sont utilisées en séquence dans une fenêtre temporelle tw si au plus, une seule modalité est active à un instant donné et si toutes les modalités de l'ensemble M sont utilisées dans la fenêtre temporelle tw. L'utilisation séquentielle des modalités d'un ensemble M dans une fenêtre temporelle tw est définie par :

$$\begin{array}{|l} Sequential(M, tw) \Leftrightarrow \\ (\text{Card(M)}{>}1) \wedge (Duration(tw) \neq \infty) \wedge \\ (\forall t \in tw(\forall m,\ m' \in M \ Active(m, t) \Rightarrow \neg Active(m', t)) \wedge (\forall m \in M, \exists t \in tw \\ \text{Active(m,t))} \end{array}$$

1.7.3.3 Définition formelle des propriétés CARE

Les propriétés CARE peuvent caractériser la relation entre les états et les modalités en quatre types :

Equivalence. Les modalités de l'ensemble M sont dites équivalentes pour atteindre l'état s' à partir de l'état s, s'il est nécessaire et suffisant d'utiliser une de ces modalités. L'ensemble M doit contenir au moins deux modalités. Cette définition est donnée formellement par :

$$\begin{array}{|l} Equivalence(s, M, s') \Leftrightarrow \\ (\text{card(M)}{>}1) \wedge (\forall m \in M \ Reach(s, m, s')) \end{array}$$

Cette propriété exprime la possibilité de choisir entre plusieurs modalités sans aucune contrainte temporelles entre elles.

Assignation. Une modalité m est dite assignée à l'état s pour atteindre l'état s', si aucune autre modalité ne permet d'atteindre l'état s' à partir de s. Contrairement à l'équivalence, l'assignation exprime l'absence de choix. L'équivalence

$$\boxed{\begin{array}{l} Assignement(s, m, s') \Leftrightarrow \\ \text{Reach(s,m,s')} \wedge (\forall m' \in M Reach(s, m, s') \Rightarrow m' = m) \end{array}}$$

et l'assignation expriment le choix ou l'absence de choix de modalités pour atteindre un but. L'assignation ne dépend pas de la fenêtre temporelle.

Redondance. Les modalités de l'ensemble M sont utilisées de manière redondante pour atteindre l'état s' à partir de l'état s, si elles sont équivalentes et elles sont utilisées toutes dans la même fenêtre temporelle. Elles sont décrites formellement par :

$$\boxed{\begin{array}{l} Redundancy(s, M, s', tw) \Leftrightarrow \\ \text{Equivalence(s,M,s')} \wedge (Sequential(M, tw) \vee Parallel(M, tw)) \end{array}}$$

Complémentarité. Les modalités de l'ensemble M sont dites complémentaires pour atteindre l'état s' à partir de l'état s dans une fenêtre temporelle tw, si elles sont toutes utilisées pour atteindre l'état s' à partir de s et aucune modalité ne permet à elle seule d'atteindre s'.

$$\boxed{\begin{array}{l} Complementarity(s, M, s', tw) \Leftrightarrow \\ (\text{Card(M)}{>}1) \wedge (Duration(tw) \neq \infty) \wedge \\ (\forall M' \in P(M)(M' \neq M \Rightarrow \neg Reach(s, M', s'))) \wedge Reach(s, M, s') \wedge \\ (\text{Sequential(M,tw)} \vee Parallel(M, tw)) \end{array}}$$

$P(M)$ désigne l'ensemble de partitions de M.
$Reach(s, M, s') \Leftrightarrow \forall m \in M, Reach(s, m, s')$, ce qui signifie que l'état s' peut être atteint à partir de l'état s en utilisant les modalités de l'ensemble M.

Ces propriétés permettent de mesurer la flexibilité et la robustesse des IHM multimodales. En effet, l'équivalence est une propriété qui définit aussi bien la flexibilité en offrant le choix à l'utilisateur que la robustesse de l'IHM. En cas de panne d'un dispositif l'interface continue à être utilisable. L'assignation, et contrairement à l'équivalence est restrictive. Elle ne permet de réaliser une tâche donnée qu'avec la modalité qui lui est assignée ce qui engendre la non utilisabilité de l'interface dans le cas où le média correspondant est en panne. La redondance permet d'augmenter la robustesse de l'IHM, mais charge le système. La complémentarité permet la flexibilité de l'interaction mais risque de générer des problèmes de synchronisation et d'interprétation cognitives.

1.7.3.4 Définition paramétrée des propriétés CARE

Les propriétés CARE peuvent être définies à différents niveaux de raffinement et peuvent être appliquées aussi bien du coté utilisateur pour les interactions en entrée, que du coté système pour les interactions en sortie.

Les propriétés CARE ont été définies également dans [Nigay & Coutaz, 1997] en prenant en compte les dispositifs physiques et les langages d'interaction associés. Cette définition permet de préciser les définitions précédentes car elles sont paramétrées par les langages autorisés.

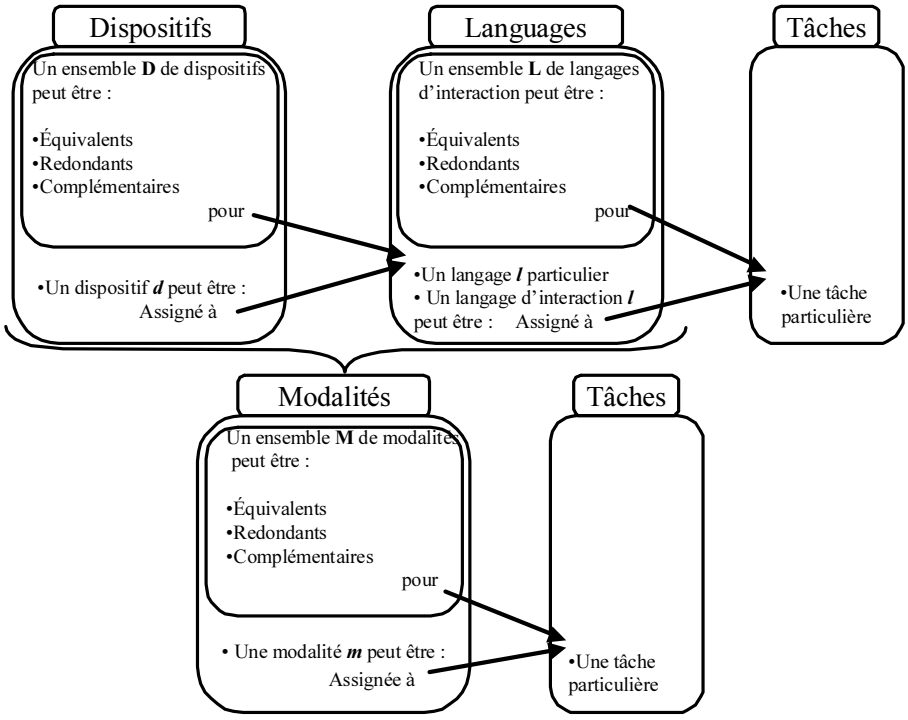

Fig. 1.12 – Les propriétés CARE par rapport aux dispositifs, langages et modalités

Equivalence de langage : L-Equivalence(L,s,T). Les langages d'interaction d'un ensemble L sont équivalents par rapport à un état s et un ensemble non vide T de tâches, si toutes les tâches dans l'ensemble T peuvent être exprimées en utilisant un langage de l'ensemble L.

Assignation de langage : L-Assignation(l,s,T). Un langage d'interaction l est assigné à l'ensemble des tâches T, dans l'état s, s'il n'existe aucun autre langage équivalent à l réalisant les tâches T dans l'état s.

Redondance de langage : L-redondance(L,s,t). Les langages d'interaction d'un ensemble L sont dits redondants pour la tâche t dans l'état s, s'ils sont équivalents pour réaliser la tâche t dans l'état s et sont utilisés simultanément (en parallèle).

Complémentarité des langages : L-Complémentarité(L,s,T). Les langages d'interaction de l'ensemble L sont dits complémentaires pour s et T, si l'ensemble T est partitionné de sorte que pour chaque partition Tp de T, il existe un langage l de L assigné à Tp dans l'état s.

Ces propriétés sont définies de la même manière pour les dispositifs physiques.

Equivalence de dispositifs : D-Equivalence(D,s,E,l). Les dispositifs de l'ensemble D sont dits équivalents dans l'état s et pour un ensemble E d'expressions d'un langage d'interaction l, si toutes les expressions de l'ensemble E peuvent être définies en utilisant n'importe quel dispositif de l'ensemble D.

Assignation de dispositifs : D-Assignation(d,s,E,l). Un dispositif d de l'ensemble D est dit assigné dans l'état s pour un ensemble d'expressions E d'un langage d'interaction l, si toutes les expressions de l'ensemble E ne peuvent être formulées qu'en utilisent le dispositif d.

Redondance de dispositifs : D-Redondance(D,s,E,l). Les dispositifs de l'ensemble D sont dits redondants dans l'état s pour un ensemble d'expression E d'un langage d'interaction l, s'ils sont équivalents pour la formulation des expressions E et sont utilisées en parallèle (simultanément).

Complémentarité des dispositifs : D-Complémentarité (D,s,E,l). Les dispositifs de l'ensemble D sont dits complémentaires dans l'état s pour l'ensemble des expressions E d'un langage d'interaction l, si l'ensemble E est partitionné de sorte que pour chaque partition Ep de E, il existe un dispositif d dans D assigné à Ep dans l'état s.

Ces propriétés sont dites permanentes dans le cas où elles sont vérifiées dans tous les états du système, et temporaires dans le cas contraire. Elles sont dites totales dans le cas où elles sont vérifiées pour toutes les tâches du système, et partielles dans le cas contraire.

1.7.4 Mise en oeuvre des propriétés CARE

Les propriétés CARE ont servi de base pour la définition de la plateforme de conception d'IHM multimodale ICARE [Bouchet *et al.*, 2004]. Cette plateforme se base sur un modèle de composants conceptuels qui décrivent les composants programme manipulés. La plateforme distingue deux types de composants conceptuels :

1. composants élémentaires : ils désignent les modalités ;
2. composants de composition : ils se basent sur les quatre propriétés CARE pour définir les opérations de composition des modalités.

Le concepteur choisit les composants élémentaires et les compose en utilisant les composants de composition afin d'obtenir des interactions plus complexes, vérifiant

les propriétés CARE qu'il aurait choisies. La méthodologie développée consiste à associer un composant de composants ICARE à chaque tâche élémentaire apparaissant dans les feuilles d'un arbre de tâche.

1.7.5 Relation entre l'espace CASE et les propriétés CARE

Après avoir présenté l'espace de conception CASE et les propriétés CARE, nous pouvons déduire que les deux modèles traitent les différentes manières dont les modalités sont combinées, mais de deux points de vue différents. L'espace CASE définit les différentes manières avec lesquelles les modalités peuvent être combinées lors de la conception du système, il est donc centré système. Les propriétés CARE définissent les différentes manières avec lesquelles les modalités peuvent être utilisées par l'utilisateur, elles sont donc centrées utilisateurs. Le choix de l'espace de conception CASE influe sur les propriétés CARE vérifiées par le système. Il est évident qu'une IHM3 de type exclusif ne vérifie pas la propriété de complémentarité sur ses énoncés, car chaque énoncé est produit par une seule modalité.

1.7.6 Vérification des propriétés des systèmes interactifs

Les propriétés des systèmes interactifs sont en général établies de trois manières :

1. par le test tel que mis en oeuvre en génie logiciel. Ceci concerne en général les propriétés liées au système. Cette manière d'établir les propriétés n'est pas sûre car les jeux de test ne sont pas exhaustifs et ne peuvent pas confirmer l'établissement des propriétés pour tous les cas possibles, sauf pour le cas des systèmes finis ;
2. par l'expérimentation en observant les utilisateurs. Ceci concerne les propriétés d'utilisabilité. Dans ce cas plusieurs méthodes techniques ont été définies par les ergonomes et psychologues. Comme pour les tests effectués pour valider le système, l'observation ne permet pas de confirmer dans l'absolu la vérification des propriétés du moment que l'observation ne concerne qu'une catégorie d'utilisateurs.
3. par la preuve et la vérification formelle en utilisant des modèles formels pour l'expression du système et des propriétés. Ces techniques peuvent concerner aussi bien les propriétés du système que certaines propriétés liées à l'utilisabilité de ce dernier. En utilisant des modèles et des procédés formels se basant sur des concepts mathématiques rigoureux, ces méthodes permettent de confirmer la vérification des propriétés d'une manière formelle avant et/ou après l'implémentation du système.

Nous ne pensons pas qu'une manière est meilleure que l'autre ou doit être utilisée exclusivement. Ces différentes manières sont complémentaires. Elles doivent être

utilisées conjointement dans un cadre de méthodologie à définir. Ce point sort du cadre de notre travail ; en effet, nous nous intéressons à l'approche formelle.

1.7.7 Vérification et validation de propriétés CARE

La vérification et validation de propriétés CARE est réalisée par une approche empirique soit par le test et l'experimentation ou bien par construction comme sur la plate forme ICARE. Mais dans ce dernier cas, se pose le problème de correction des constructions, composants et opérateurs de composition que fournit cette plateforme.

Par ailleurs, les expressions de propriétés fournies dans cette section sous entendent la définition d'états, de systèmes, de langages, de notions de parallélisme et de séquence, d'expressions, etc., sans en donner de sémantique formelle (précise et non ambiguë). De plus, aucun moyen ou technique permettant d'établir ces propriétés (i.e les propriétés CARE) n'est fourni. Ces expressions restent au stade de la notation, il est nécessaire de fournir un cadre opérationnel et méthodologique pour répondre à ces interrogations. C'est ce que nous nous efforçons de proposer dans nos travaux par la définition d'un cadre général permettant d'exprimer des modèles IHM3 et d'en établir les propriétés.

1.8 Le système multimodal Matis : étude de cas

Les systèmes multimodaux sont mis en oeuvre dans divers domaines d'application. Certains systèmes ont été développés pour des besoins réels tel que le système [Merloz *et al.*, 2000] qui permet d'aider les chirurgiens à placer une vis dans le pédicule d'une vertèbre en utilisant deux modalités en entrée, et le système VICO (Virtual Intelligent CO-Driver)[Bernsen & Dybkjaer, 2001] utilisé pour l'aide à la conduite d'automobile. Ce dernier utilise deux modalités en entrée : le geste à l'aide de l'écran tactile et la parole.

D'autres systèmes ont été développés dans le but d'étudier l'interaction multimodale tel que [Juster & Roy, 2004] et [Nedel *et al.*, 2003] et d'autres sont développés dans un cadre expérimental dans des laboratoires de recherche tel que le système MATIS [Nigay, 1994] que nous avons choisi comme étude de cas pour l'expression et la validation de nos modèles. C'est un système multimodal qui permet d'interroger une base de données sur les programmes de vol à partir d'une ville de départ vers une ville de destination. Plusieurs possibilités sont offertes à l'utilisateur pour formuler ses requêtes. Trois modalités sont possibles : manipulation directe avec la souris, saisie de la requête dans un formulaire ou prononciation d'une phrase analysée par l'analyseur de langage naturel Sphinx, développé à l'université de Carnegie Mellon. Ces trois modalités peuvent être utilisées d'une manière indépendante ou synergique.

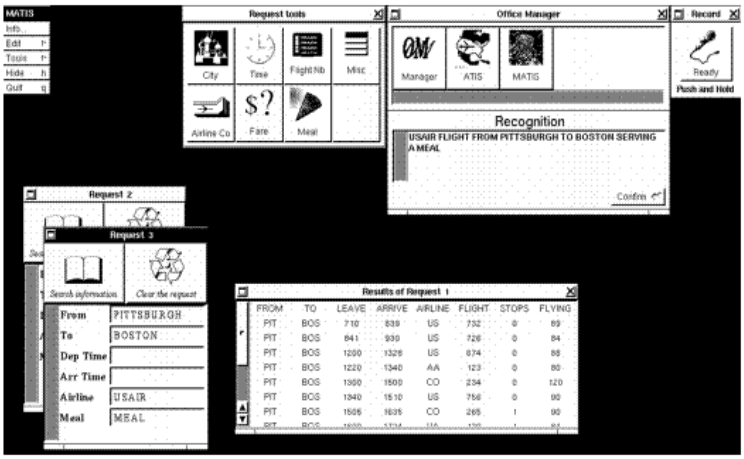

Fig. 1.13 – Interface de Matis

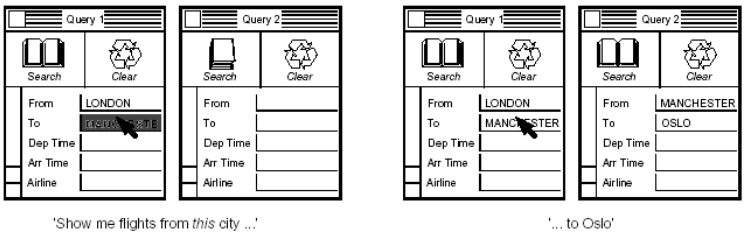

Fig. 1.14 – Requête avec parole et manipulation directe

Le système MATIS permet aussi un dialogue à plusieurs fils. L'utilisateur peut formuler deux requêtes en parallèle. À chaque requête correspond une fenêtre pour sa saisie. L'utilisateur peut interrompre l'édition d'une requête et basculer vers une autre en sélectionnant la fenêtre correspondante. La figure 1.13 montre l'interface du système MATIS.

La figure 1.14 montre la coopération entre le langage naturel et la manipulation directe avec la souris pour la formulation d'une requête. L'utilisateur prononce la phrase *"Show me flights from this city"* ensuite sélectionne la ville de destination en cliquant par la souris sur la ville dans le formulaire de la première requête, ensuite prononce la phrase *"to Oslo"*.

Pour appliquer notre modèle, nous avons choisi Matis comme étude de cas. Notre choix est purement pédagogique. En effet, les requêtes sont des tâches assez simples qui peuvent être formulées par une ou plusieurs modalités.

1.9 Conclusion

Nous notons en conclusion deux éléments : le premier élément est lié aux IHM, et le deuxième aux IHM multimodales, notre domaine d'étude.

1. Toutes les notations et modèles définis ont permis de mieux raisonner sur les interfaces homme-machine et d'en augmenter la qualité. Néanmoins, ces modèles et notations manquent beaucoup de précision et de formalisme rigoureux ce qui empêche un raisonnement automatique sur ce type de systèmes. Pour cela beaucoup d'efforts ont été notés avec les tentatives de formaliser certains modèles. Nous notons en particulier les travaux relatifs aux modèles d'architecture et que nous avons présentés dans la section 1.5 et aux modèles de tâches tel que CTT. D'autres travaux traitant de l'utilisation des approches formelles sont présentés dans le chapitre suivant.
2. Le domaine des IHM multimodales est un domaine en pleine expansion. La complexité du parallélisme dans les interactions, la fusion des données provenant de plusieurs modalités, etc., sont des problèmes à prendre en compte dans les modèles proposés pour concevoir et vérifier ce type de systèmes. La plupart des travaux actuels se concentrent au niveau de la définition des concepts, le développement d'applications multimodales expérimentales ou bien par la proposition de plateforme de développement de travaux matures concernant la formalisation de modèles de développement, que ce soit au niveau architecture, tâche, etc. La vérification et la validation restent dans la majorité des cas fondées sur le test et l'expérimentation.

Notre travail se base sur les travaux réalisés par les experts des IHM3. Tout en les conservant, nous proposons de les réutiliser en les formalisant dans un modèle formel générique que nous proposons dans le chapitre 3. Nous ne proposons pas de nouvelles notations ni espace de conception mais nous visons à apporter plus de rigueur dans le développement des IHM3.

Chapitre 2

Développements formels de systèmes interactifs

2.1 Introduction

L'utilisation des méthodes formelles dans le développement des systèmes garantit un raisonnement rigoureux permettant de comprendre et d'appréhender leurs comportements et propriétés. Les premiers travaux proposant de formaliser le développement de logiciel ont été réalisés pas Hoare [Hoare, 1969]. Ensuite, plusieurs techniques et méthodes formelles ont été définies pour traiter tout ou partie du cycle de vie du logiciel.

La modélisation formelle consiste en la description d'un système et de ses propriétés en utilisant un langage avec une syntaxe et une sémantique définies à base de techniques mathématiques. Les propriétés peuvent servir à spécifier la statique ou la dynamique du système. Selon le niveau de détail utilisé dans la modélisation du système, on peut parler de spécification, conception ou implantation du système.

En général, le développement formel d'un système nécessite un modèle formel pour la représentation du système en question, un modèle formel pour l'expression des propriétés à vérifier et un algorithme ou une procédure de vérification de ces propriétés par le système. Afin de représenter le système ainsi que ses propriétés, plusieurs modèles formels ont été définis. Certains sont utilisés pour représenter uniquement le système, d'autres peuvent représenter uniquement les propriétés et d'autres peuvent représenter aussi bien le système que ses propriétés. Dans ce chapitre nous présentons les modèles qui interviennent aussi bien pour modéliser le système, les propriétés ainsi que les procédures implantant la vérification formelle. Nous nous intéressons particulièrement aux travaux appliquant les techniques formelles pour les IHM en général et les IHM multimodales en particulier.

2.2 Modélisation Formelle

Plusieurs approches de modélisation formelles ont été définies. Elles permettent de modéliser les systèmes, les propriétés ou les deux. On distingue trois types de modèles formels.

1. **Modèles Sémantiques :** décrivent aussi bien le système que les propriétés par l'expression de leur comportement. Ces modèles décrivent les comportements d'un système et les propriétés sont des comportements attendus du système. Les principaux modèles de cette classe sont les systèmes de transitions et les systèmes d'évènements. Plusieurs formalismes dont la sémantique est basée sur ces deux modèles ont été définis et utilisés pour spécifier les systèmes. Nous citons principalement les statecharts [Harel, 1987], les réseaux de pétri [Peterson, 1981], Lustre[Caspi & Girault, 1992] et les algèbres de processus telles que CCS [Milner, 1980], CSP [Hoare, 1985] et LOTOS [ISO8807, 1989].
2. **Modèle syntaxiques :** sont fondés sur deux modélisations complémentaires : une modélisation statique décrivant les entités constituant le système et les états pouvant leur être associés et une modélisation dynamique décrivant les changements d'états autorisés à l'aide d'actions définies sur des entités, et de propriétés devant être vérifiées avant l'action (précondition) ou après l'action (post-condition). Elles sont qualifiées de syntaxiques car elles utilisent une logique associée à un système de preuve fondée sur la réécriture d'expressions logiques comme support de preuve. Les méthodes Z [Spivey,], VDM [Jones, 1990] et B [Abrial, 1996a] font partie de cette classe.
3. **Modèles logiques :** décrivent les propriétés des systèmes sous forme de formule logique. Une procédure de vérification est mise en oeuvre pour la vérification de ces propriétés (model checking). Nous distinguons dans cette classe les logiques temporelles CTL, PLTL, ACTL, etc., et la logique du premier ordre.

2.3 Vérification formelle

On classe traditionnellement les techniques de vérification formelle selon deux grandes catégories : technique de vérification sur modèle et technique de preuve.

2.3.1 Technique de vérification sur modèle (model-checking)

La vérification sur modèle ou model-checking, est une technique basée sur l'énumération exhaustive de l'espace d'état d'un système. Il existe deux approches pour le model-checking.

2.3.1.1 Première approche

Cette approche a été développée par Clarke et Emerson [Clarke & Emerson, 1981] et Queille et Sifakis [Queille & Sifakis, 1981]. Elle consiste à modéliser le système par un système de transitions fini et à exprimer ses propriétés dans une logique temporelle [Manna & Pnueli, 1992]. Un algorithme vérifie si le système de transitions est un modèle pour la spécification exprimée par la propriété de logique temporelle.

2.3.1.2 Seconde approche

Cette approche consiste à modéliser non seulement le système mais aussi la propriété par un automate. Les deux automates, l'un décrivant le système et l'autre le comportement attendu (la propriété), sont comparés pour vérifier si le système est conforme à la spécification. Ici, la conformité est établie par une relation définie entre les deux automates. Cette relation peut être l'inclusion de langage [Kurshan, 1994], ou l'équivalence observationnelle [Cleaveland & Steffen, 1993].

Dans cette approche, le système ainsi que les propriétés sont exprimés dans le même formalisme. La vérification de la propriété consiste à vérifier une relation entre le modèle du système et celui de la propriété. Cette approche concerne les propriétés opérationnelles qui peuvent être exprimées par un système de transitions. La vérification de la propriété sur le système consiste à comparer deux ensembles de comportements.

Deux manières sont possibles pour représenter les comportements : linéaire avec les traces ou arborescente avec des arbres.

Approche linéaire : les comportements sont décrits par les traces. Si une propriété est représentée par un système de transitions B, et le système par un système de transitions A alors la vérification consiste à prouver que l'ensemble des traces du système B est inclu dans l'ensemble des traces du système A dans le cas d'une relation de pré-ordre, ou l'égalité des deux ensembles dans le cas d'une relation d'équivalence.

Approche arborescente : les comportements sont représentés par des arbres. Chaque branche représente une trace et chaque noeud représente un état du système qui est un point de choix et de synchronisation. La vérification d'une propriété dans ce cas consiste à établir une relation de pré-ordre définie par une relation de simulation ou une relation d'équivalence définie par une relation de bisimulation.

L'outil qui implémente la technique de model-checking est un model-checker (vérificateur su modèle). Plusieurs model-checkers ont été développés et ils différent par le langage de spécification du système et des propriétés, et par l'algorithme de vérification. SMV a été le premier à utiliser les BDD (Binary Decision Diagram) et SPIN utilise les ordres partiels pour parer à l'explosion d'états.

L'avantage de la vérification sur modèle est qu'elle est complètement automatisable. Elle est capable de fournir un contre exemple dans le cas où la propriété ne serait pas vérifiée, ce qui permet de mettre en évidence les erreurs et aider à la mise au point.

Par contre, son principal inconvénient est le problème de l'explosion combinatoire. En 1987, McMillan utilisa les BDD de Bryant [Bryant, 1990] pour représenter les systèmes de transitions de manière efficace, augmentant ainsi la taille des systèmes pouvant être vérifiés. D'autres approches prometteuses apparaissent telle que l'exploitation de l'information des ordres partiels, les réduction locales [Kurshan, 1994] ou encore les minimalisations sémantiques [Elseaidy *et al.*, 1997] pour éliminer les états inutiles du modèle d'un système.

Les model-checkers peuvent aujourd'hui aisément traiter des systèmes avec entre 100 et 200 variables d'états, des systèmes constitués de 10^{120} états atteignables [Burch *et al.*, 1994] peuvent être vérifiés, et en utilisant des techniques d'abstraction appropriées, ils peuvent même aborder les systèmes à états infinis [Clarke *et al.*,]. Ainsi, le model-checker est maintenant suffisamment puissant et il est largement utilisé dans l'industrie pour assister la conception des nouveaux systèmes. Dans ce qui suit, nous allons présenter les modèles utilisés dans les deux approches.

2.3.2 Technique de preuve

Avec la technique de preuve, le système ainsi que ses propriétés sont données par des formules exprimées dans une logique, un formalisme mathématique. Cette logique est donnée par un système formel définissant un ensemble d'axiomes et de règles d'inférence. La démonstration de théorème consiste à trouver la preuve d'une propriété en fonction des axiomes du système. Les étapes de la preuve font appel aux règles et aux axiomes et éventuellement à des lemmes intermédiaires. Plusieurs prouveurs automatiques existent actuellement. Ils sont assistés par ordinateurs. Ils sont utilisés de plus en plus dans la vérification de propriétés critiques de sécurité de systèmes hardware et software.

Les outils de preuve sont très variés ; ils vont du plus automatisé au plus interactif et du plus général au plus spécialisé. Certains sont guidés par une séquence de lemmes et de définitions, mais chaque théorème est prouvé automatiquement en utilisant des heuristiques pour l'induction, la réécriture et la simplification.

Il existe des outils combinant plusieurs techniques, tels que Analytica qui combine la démonstration de théorème avec l'algèbre symbolique de Mathematica. PVS et STep permettent d'utiliser le model-checking et la preuve. Enfin, l'Atelier B combine un développement par raffinement, un prouveur et un générateur de code. La méthode B sera présentée en détail dans le chapitre 5 ainsi que son utilisation pour mettre en oeuvre notre modèle formel d'interactions multimodales.

2.4 Modèles formels

Nous présentons dans ce qui suit, quelques modèles utilisés pour la spécification des systèmes et des propriétés. Nous avons choisi de présenter ceux que nous utiliserons dans la modélisation formelle que nous proposons dans le chapitre 3. En effet, les systèmes de transitions constituent une technique de modélisation générique adaptée à la modélisation de systèmes interactifs.

2.4.1 Systèmes de transitions

Un système de transitions étiquetées est un modèle formel utilisé pour modéliser le comportement dynamique des systèmes. Un système de transitions est décrit principalement par un ensemble d'états, un ensemble d'événements, un état initial et une relation de transition entre les états. Les événements produits permettent la transition, éventuelle, du système d'un état à un autre. Il est défini formellement par un tuple $A = (Q, E, T, q_0)$ où

- Q : ensemble des états du système
- E : ensemble d'étiquettes désignant les actions ou les évènements du système
- T : relation de transition $\subseteq Q \times E \times Q$
- q_0 : état initial du système, $q_0 \in Q$

Une transition t est donc un triplet $t = (p, a, q)$ $(a \in E)$ que l'on note $p \longrightarrow^a q$. Un système de transitions peut être représenté par un graphe où une transition (q_i, a, q_j) est représentée par

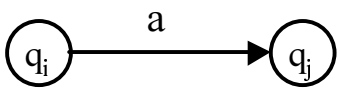

Fig. 2.1 – Représentation graphique d'une transition

L'*état initial* est représenté comme suit :

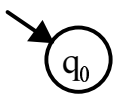

Fig. 2.2 – Représentation graphique de l'état initial

Une *trace* ou encore appelée *chemin* dans un système de transitions A est une suite σ, finie ou infinie, de transitions (q_i, a_i, q'_i) de A qui s'enchaînent, c'est-à-dire $q'_i = q_{i+1}$ pour tout i. On note souvent une telle suite sous la forme $q_1 \longrightarrow^{a1} q_2 \longrightarrow^{a2} q_3 \longrightarrow^{a3} q_3 \ldots$.

Un *comportement* est un chemin dans le graphe représentant le système de transitions.

2.4.2 Structure de Kripke

Les structures de Kripke sont des systèmes de transitions enrichis par une fonction qui associe à chaque état les propositions vérifiées dans cet état. Ils sont utilisés comme modèle sémantique pour certaines logiques temporelles. Soit $P = \{p_1, p_2, ...\}$ un ensemble de propositions décrivant des propriétés élémentaires. Une structure de Kripke est un modèle représenté par le tuple $K = (Q, E, T, q_0, \rightarrow, l)$ où :

- Q est une ensemble fini d'états,
- E est l'ensemble fini des étiquettes des transitions,
- T avec $T \subseteq Q \times Q$ est l'ensemble des transitions,
- q_0 est l'état initial du modèle,
- l est l'application qui associe à tout état de Q l'ensemble fini des propriétés élémentaires vérifiées dans cet état.

L'application l permet d'obtenir les différentes propriétés vérifiées dans un état donné. Elle permet entre autre de décrire les variables d'états qui caractérisent un système de processus et de les observer lors d'un changement d'état. Un chemin dans un modèle K est une suite σ finie ou infinie, de transitions (q_i, q_{i+1}) de K qui s'enchaînent, c'est-à-dire $q_i \xrightarrow{e_i} q_{i+1}$ pour tout i. On désigne également chemin par le mot *trace*.

En général, ce sont les structures de Kripke qui sont utilisées pour modéliser les systèmes à la place des systèmes de transitions.

2.4.3 Produit de systèmes de transitions

Lors de la description d'un système, on procède souvent par la description des différents sous systèmes qui le composent. Chaque système peut être décrit par un système de transitions. Le système complexe est décrit par l'ensemble des systèmes de transitions.

Le produit cartésien (produit libre) et produit synchronisé de systèmes de transitions permettent de décrire des systèmes par assemblage (opération de produit) des systèmes de transitions de base. Ces opérations de produit permettent d'obtenir un système de transitions qui décrit globalement le système. Généralement, le système de transitions obtenu possède un très grand nombre d'états, si bien qu'il est difficile voire impossible de le construire.

2.4.3.1 Produit cartésien ou produit libre.

Considérons une famille de n systèmes de transitions $A_i = (Q_i, E_i, T_i, q_{0,i}, l_i)$,- $i = 1 \ldots n$.

Soit '$-$' une nouvelle étiquette permettant d'exprimer l'action fictive. Cette action fictive servirait à décrire qu'un des sous-systèmes de transitions n'effectue aucune transition dans le système de transitions global (transition de bégaiement).

Le produit cartésien $A_1 \times A_2 \times \ldots \times A_n$ de ces systèmes de transitions, est le système $A = (Q, E, T, q_0, l)$ tel que :

- $Q = Q_1 \times Q_2 \times \ldots \times Q_n$
- $E = \prod_{i=1}^{n}(E_i \cup \{-\})$
- $T = \{((q_1, \ldots, q_n)(a_1, \ldots, a_n)(q'_1, \ldots, q'_n))|\forall i \in 1..n, (a_i =' -' \text{ et } q_i = q'_i) \text{ ou bien } (a_i \neq' -' \text{ et } (q_i, a_i, q'_i) \in T_i)\}$
- $q_0 = (q_{0,1}, \ldots, q_{0,n})$
- $l = \bigcup_{i=1}^{n}(l_i(q_i))$

Dans un produit cartésien, chaque système de transitions A_i peut, lors d'une transition, soit effectuer une transition locale, soit ne rien faire (action fictive). Il n'y a aucune obligation de synchronisation entre les différents systèmes de transitions. De plus, le produit cartésien permet des transitions où tous les systèmes de transitions ne font rien.

2.4.3.2 Produit synchronisé.

Pour synchroniser les différents systèmes de transitions d'un produit cartésien, il est nécessaire de restreindre les transitions possibles dans le système résultant du produit cartésien. Seules les transitions correspondant à des transitions de synchronisations acceptées seront conservées.

Considérons un ensemble de synchronisation $Sync$, ou encore appelé vecteur de synchronisation, défini par :

$$Sync \subseteq \prod_{i=1}^{n}(E_i \cup \{-\})$$

Le produit synchronisé $(A_1 \parallel A_2 \parallel \ldots \parallel A_n)_{Sync}$ des systèmes de transitions A_i, est l'automate $A = (Q, E, T, q_0, l)$ tel que :

- $Q = Q_1 \times Q_2 \times \ldots \times Q_n$
- $E = \prod_{i=1}^{n}(E_i \cup \{-\})$
- $T = \{((q_1, \ldots, q_n)(a_1, \ldots, a_n)(q'_1, \ldots, q'_n))|(a_1, \ldots, a_n) \in Sync \text{ et } \forall i \in 1..n, (a_i =' -' \text{ et } q_i = q'_i) \text{ ou bien } (a_i \neq' -' \text{ et } (q_i, a_i, q'_i) \in T_i)\}$
- $q_0 = (q_{0,1}, \ldots, q_{0,n})$
- $l = \bigcup_{i=1}^{n}(l_i(q_i))$

2.4.4 Logique temporelle

Les logiques temporelles sont des logiques qui permettent d'exprimer des propriétés dans le temps. Elles sont utilisées pour exprimer des propriétés comportementales des systèmes. L'avantage de l'utilisation des logiques temporelles pour l'expression

des propriétés est que leur vérification est automatique dans le cas de systèmes finis et ceci par model-checking, grâce à l'énumération de traces. Plusieurs logiques ont été définies. Elles se classent principalement en deux grandes classes : les logiques temporelles *linéaires* et les logiques temporelles *arborescentes*. Les logiques temporelles linéaires utilisent les traces comme modèle d'interprétation tandis que les logiques arborescentes utilisent les arbres comme modèle sur lesquels les propriétés sont vérifiées. Nous avons choisi de présenter les deux logiques temporelles que nous utilisons pour exprimer et pour vérifier les propriétés d'utilisabilité des IHM3 : une logique temporelle arborescente (CTL) et une logique temporelle linéaire (LTL).

2.4.4.1 Logique temporelle CTL*

La logique temporelle CTL* est une logique temporelle arborescente dont découlent les deux logiques CTL et LTL. Nous donnons la syntaxe ainsi que la sémantique de CTL*.

Syntaxe : la syntaxe de la logique temporelle est donnée par la grammaire suivante :

$$\begin{aligned}\phi, \psi ::= & \ p_1 \mid p_2 \mid ... \\ & \mid \neg\phi \mid \phi \wedge \psi \mid \phi \Rightarrow \psi \mid \phi \vee \psi \mid \\ & \mid X\phi \mid F\phi \mid G\phi \mid \phi \cup \psi \mid ... \\ & \mid E\phi \mid A\phi\end{aligned}$$

Les propositions atomiques sont utilisées pour caractériser ces états qui apparaissent dans les comportements. Ces propositions sont regroupées dans l'ensemble de propositions $P = \{p_1, p_2...\}$. Une proposition p_i est vraie dans l'état q d'une structure de Kripke si $p_i \in l(q)$. Les combinateurs booléens classiques de la logique sont indispensables pour construire des expressions logiques complexes.

Les combinateurs temporels **X**, **F**, et **G** permettent de décrire les enchaînements des états au sein des différents comportements. Soit p une proposition de l'état courant. Alors

- **X**p énonce que p est vraie dans l'état suivant ;
- **F**p énonce que p est vraie dans un futur (suivant l'état courant) sans préciser quel état. Cette expression peut se lire *"la propriété p sera vraie un jour"* ;
- **G**p énonce que tous les états futurs satisfont la propriété p. Cette expression peut se lire *"la propriété p sera toujours vraie"*.
- $\phi_1 \cup \phi_2$ énonce que ϕ_1 est vraie jusqu'à ce que ϕ_2 le soit. En d'autres termes, *"ϕ_2 sera vérifiée un jour, en attendant ϕ_1 restera vraie"*.

Le combinateur **G** est dual de **F**. En effet, $\mathbf{G}\phi \equiv \neg\mathbf{F}\neg\phi$. Le combinateur **F** est un cas particulier de $\cup$, dans la mesure où $\mathbf{F}\phi$ peut s'écrire $(true \cup \phi)$. La combinaison des combinateurs permet d'augmenter la puissance d'expression des propriétés en logique temporelle. La combinaison de **F** et **G** est souvent utilisée. Ainsi les combinateurs suivants sont autorisés :

- **GF**ϕ se lit *"toujours il y aura un jour un état tel que ϕ"*. L'expression ϕ sera vraie infiniment souvent ;
- **FG**ϕ se lit *"tout le temps à partir d'un certain moment ϕ sera vraie"*. L'expression ϕ sera vraie tout le temps sauf un nombre fini de fois.

A partir d'un état, plusieurs futurs sont possibles. Cela donne une représentation arborescente des comportements. Les chemins ou les comportements sont des parcours ou des branches particuliers dans l'arborescence des comportements. Lorsque les comportements sont arborescents, la logique temporelle introduit des quantificateurs de chemins **A** et **E** qui permettent de quantifier l'ensemble des exécutions sur l'ensemble des chemins.

- La formule $\mathbf{A}\phi$ énonce que toutes les exécutions partant de l'état courant satisfont la propriété ϕ.
- La formule $\mathbf{E}\phi$ énonce qu'il existe une exécution, partant de l'état courant, qui satisfait la propriété ϕ.

Sémantique : la sémantique des formules de CTL* est définie à l'aide d'une structure de Kripke $K = (Q, A, \rightarrow, q_0, L)$. Pour définir cette sémantique, il faut définir la relation de satisfaction $\models$ qui indique dans quelles conditions, une propriété exprimée en logique temporelle est satisfaite. Pour qu'une formule soit satisfaite, il faut se référer à :

- un automate ou modèle de Kripke,
- une trace ou une exécution σ,
- un temps i qui indique l'état courant dans l'exécution.

On écrit $K, \sigma, i \models \phi$ pour exprimer qu'au temps i de l'exécution σ de l'automate K, la formule ϕ est vraie. Dans plusieurs écritures, le contexte K est implicite, il représente le système étudié. Il est souvent omis.

La définition de $\sigma, i \models \phi$ se fait par induction sur la structure de la propriété ϕ. Elle est donnée par les expressions suivantes :

-$\sigma, i \models p$ ssi $p \in L(\sigma(i))$
-$\sigma, i \models \neg\varphi$ ssi il n'est pas vrai que $\sigma, i \models \phi$
-$\sigma, i \models \phi_1 \cup \phi_2$ ssi $\sigma, i \models \phi_1$ et $\sigma, i \models \phi_2$
-$\sigma, i \models X\phi$ ssi $i <\mid \sigma \mid$ et $\sigma, i+1 \models \phi$
-$\sigma, i \models F\phi$ ssi il existe j tel que $i \leq j \leq \mid \sigma \mid$ et $\sigma, j \models \phi$
-$\sigma, i \models G\phi$ ssi pour tout j tel que $i \leq j \leq \mid \sigma \mid$ on a $\sigma, j \models \phi$
-$\sigma, i \models \phi \cup \psi$ ssi il existe $j, i \leq j \leq \mid \sigma \mid$ tel que $\sigma, j \models \psi$
et pour tout k tel que $i \leq k < j$, on a $\sigma, k \models \phi$
-$\sigma, i \models E\phi$ ssi il existe un σ' tel que $\sigma(o)$,..., $\sigma(i) = \sigma'(0), ..., \sigma'(i) et \sigma', i \models \phi$
-$\sigma, i \models A\phi$ ssi pour tout σ' tel que $\sigma(o)$,...,$\sigma(i) = \sigma'(0)$,...,$\sigma'(i)$ on a σ', $i \models \phi$

$\mid \sigma \mid$ et $\sigma(i)$ représentent respectivement la longueur de la trace σ et le i-ème état de σ. On dit qu'un modèle K satisfait une formule ϕ notée $\models \phi$ par : $S \models$ ssi $\sigma, 0 \models \phi$ pour toute exécution σ

2.4.4.2 Logique temporelle linéaire LTL (Linear Temporal Logic)

LTL est dite logique temporelle linéaire. Elle est interprétée sur des chemins d'exécutions d'un système de transitions. Les quantificateurs **A** et **E** n'existent pas dans cette logique. Elle est obtenue à partir du langage de la logique CTL* en retirant les quantificateurs **A** et **E**.

Une formule de LTL ne peut pas examiner les chemins d'exécutions alternatives issues du choix non déterministe. LTL s'intéresse à toutes les exécutions prises d'une manière linéaire indépendamment les unes des autres. Toutes les formules de LTL sont des formules de chemin.

LTL ne permet pas d'exprimer qu'à un certain moment le long d'un chemin, il serait possible de prolonger l'exécution de telle ou telle manière. SPIN est un exemple de modèle-checker implémentant la vérification de formules de la logique temporelle LTL.

2.4.4.3 Logique temporelle arborescente CTL

CTL est une logique temporelle arborescente. Elle est obtenue à partir du langage de la logique CTL* en exigeant que chaque utilisation d'un combinateur temporel (X, F, G ou ∪, etc.) soit immédiatement sous la portée d'un quantificateur A et E. Cette restriction fait que les formules de CTL sont des formules d'états. Les propriétés exprimées dans cette logique peuvent être vérifiées en utilisant l'état courant et les états qui peuvent être atteints à partir de l'état courant dans l'automate, et non en utilisant une exécution courante.

Cette logique a été utilisée par Abowd et al. [Abowd *et al.*, 1995] pour vérifier des propriétés de dialogue dans les systèmes interactifs. SMV est un exemple de model-checker implémentant la vérification de formules de la logique CTL. Nous trouvons

également dans [Loer & Harrisson, 2001] des schémas génériques de propriétés d'utilisabilité des systèmes interactifs. Par exemple la propriété d'atteignabilité est exprimée par la formule **AG EF**(< *Configuration-cible* >) où $< configuration - cible >$ désigne l'état du système que nous voulons atteindre.

2.4.5 Relations d'équivalence

Ce sont des relations qui permettent de comparer deux systèmes de transitions. Elles sont utilisées pour vérifier si deux systèmes de transitions sont équivalents. Elles peuvent être utilisées également pour vérifier si le système décrivant une propriété du système est bien pris en charge par le système de transitions définissant le système à vérifier. Nous citons deux exemples de relations d'équivalence : relation de simulation et relation de bisimulation.

2.4.5.1 Relation de simulation

Une relation binaire $R \subseteq Q \times Q$ entre deux états est une simulation si $(p, q) \in R$ implique pour toute action $a \in A$:

Si $p \xrightarrow{a} p'$ alors pour tout état q', $q \xrightarrow{a} q'$ et $(p', q') \in R$

La relation de simulation lie un état à un autre, mais uniquement dans un sens. Deux systèmes sont similaires si leurs états initiaux sont similaires.

Le systèmes S2 simule le système S1 sur la figure 2.3.

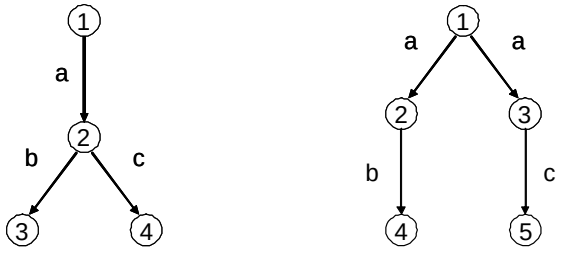

Fig. 2.3 – Exemple de systèmes similaires

2.4.5.2 Relation de bisimulation

Une relation de bisimulation est une relation de simulation dans les deux sens. Une relation binaire $R \subseteq q \times q$ entre deux états est une bisimulation si $(p, q) \in R$ implique pour toute action $a \in A$:

1. Si $p \xrightarrow{a} p'$ alors pour tout état q', $q \xrightarrow{a} q'$ et $(p', q') \in R$
2. Si $q \xrightarrow{a} q'$ alors pour tout état p', $p \xrightarrow{a} P'$ et $(p', q') \in R$

Cette relation se contente d'identifier les graphes ayant la même structure arborescente. La figure 2.4 montre deux systèmes bisimilaires S1 et S2.

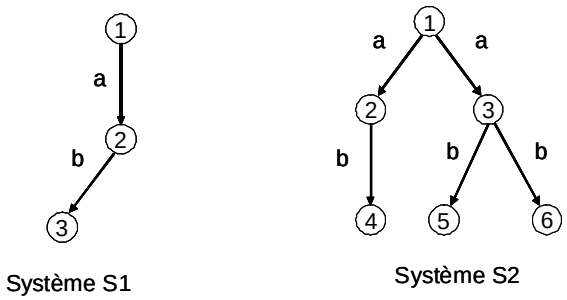

Fig. 2.4 – Exemple de systèmes bisimilaires

Ceci n'est pas le cas des systèmes présentés sur la figure 2.3. Ces deux systèmes différent par le fait que le premier ($S1$) ne peut être bloqué après avoir effectué l'action a que lorsque ni l'action b ni l'action c ne sont possibles. Par contre le systèmes S2, après avoir effectué l'action a, peut être bloqué parce que soit l'action b soit l'action c n'est pas possible.

2.5 Démarche de conception

Il existe principalement deux manières de procéder pour la modélisation d'un système :

Modélisation descendante ou par décomposition. Elle consiste à passer d'une spécification abstraite du système à une spécification concrète du système en plusieurs étapes de raffinements successifs. Le raffinement permet d'enrichir la spécification avec de nouveaux éléments de description de la spécification. Cette démarche permet de préciser les modèles de la spécification au fur et à mesure que le développement est réalisé. Les propriétés sont introduites et vérifiées à chaque étape du développement. Le raffinement permet de garantir que les propriétés vérifiées dans les étapes précédentes sont toujours conservées au niveau de la nouvelle d'étape de raffinement ce qui évite de les vérifier une nouvellr fois. Ceci constitue l'avantage majeur de cette démarche.

Modélisation ascendante ou par composition. Elle consiste à modéliser le système en composant des sous systèmes. C'est une démarche qui permet de réutiliser des systèmes déjà définis pour la conception de nouveaux systèmes. L'inconvénient de cette démarche est qu'elle ne garantit pas toujours la préservation des propriétés vérifiées sur les sous systèmes. Le concepteur est obligé de garantir la preservation des propriétés des sous systèmes sur le système global.

2.6 Utilisation des méthodes formelles pour les IHM

Plusieurs méthodes et techniques formelles ont été utilisées pour la conception et la vérification des IHM de type WIMP. Certaines sont basées sur la preuve et d'autres sont basées sur la vérification sur modèle. De même plusieurs modèles de spécification ont été utilisés : réseaux de pétri, algèbre de processus, etc.

Peu de travaux mettant en oeuvre des techniques formelles se sont intéressés à la modélisation, à la vérification et à la validation des IHM multimodales. La plupart de ces travaux étendent les formalismes déja utilisés pour les IHM de type WIMP afin de prendre en charge la multimodalité.

Nous passons en revue les travaux utilisant les techniques formelles pour le développement des IHM de type WIMP ainsi que les IHM multimodales.

2.6.1 Le model-checking : Utilisation de SMV

SMV a été utilisé par Abowd et al. [Abowd *et al.*, 1995] pour la vérification des propriétés du dialogue, d'une spécification du dialogue effectuée par le Simulateur d'Action (Action Simulator).

L'IHM est spécifiée à l'aide du simulateur d'action, ensuite, la spécification est traduite en langage d'entrée de SMV qui permet de vérifier des propriétés exprimées dans la logique temporelle CTL.

Dans ce travail, aucun modèle d'architecture n'est utilisé pour représenter l'IHM. Seul le dialogue est spécifié de manière tabulaire à l'aide du système de production propositionnel (PPS) [Olsen, 1990]. L'outil offre aussi la possibilité de simuler le dialogue.

Un ensemble de schémas génériques de propriétés de dialogue a été proposé. On y trouve le non blocage, la complétude des tâches, etc.

SMV a également fait l'objet du travail de [de Campos, 1999], où les interacteurs de York sont d'abord spécifiés dans un langage basé sur la logique modale MAL [Ryan *et al.*, 1991], ensuite la spécification est traduite vers le langage d'entrée de SMV où des propriétés exprimées en CTL sont vérifiées. Contrairement au travail précédent où seul le dialogue est modélisé, ici toute l'interface est spécifiée en utilisant l'architecture d'interacteur de York. Nous avons utilisé cette démarche, [Kamel, 2004] pour vérifier les propriétés d'utilisabilité des IHM multimodales, que nous présentons dans le chapitre 4.

2.6.2 Les algèbres de processus : utilisation de Lotos

Lotos est une algèbre de processus [ISO84, 1984]. Elle a été utilisée pour modéliser les interacteurs de Pise, Cnuce dans les travaux de Paterno et Faconti [Paterno, 1995] [Paterno & Faconti, 1992] et les interacteurs *ADC* dans les travaux de Markopoulos [Markopoulos *et al.*, 1996][Markopoulos, 1997]. Dans cette approche l'interface

est modélisée par un ensemble d'interacteurs interconnectés entre eux. Chaque interacteur est spécifié en Lotos. L'interconnexion entre les différents interacteurs est réalisée par le lien entre les ports de communication des différents processus lotos.

Paterno, démarre la conception de l'interface à partir d'une spécification des tâches qu'il traduit en une hiérarchie d'interacteurs. Ensuite, chaque interacteur est traduit en Full-lotos qui est un langage basé sur lotos de base pour sa partie de spécification du contrôle et la spécification algébrique ACT-ONE [Quemada *et al.*, 1993] pour sa partie de spécification des données. Enfin, la spécification lotos de l'interface est traduite en un système de transitions. Pour que cette traduction soit possible automatiquement, la spécification est traduite d'abord vers lotos de base. Cette traduction implique la perte des données et les paramètres ainsi que les gardes booléennes utilisées pour contraindre le comportement. Une fois que le système de transitions correspondant à la spécification de l'interface est obtenu, les propriétés sont spécifiées dans la logique ACTL et vérifiées à l'aide du model-checker de l'outil Lite. Un ensemble de schémas de propriétés vérifiées à l'aide de la logique temporelle ACTL sont définies dans [Paterno, 1995].

Dans le travail de Markolpoulos [Markopoulos *et al.*, 1996], les deux approches de model-checking ont été utilisées. Une approche identique à celle de Paterno qui consiste à traduire la spécification lotos en un système de transitions et la spécification des propriétés à l'aide de la logique temporelle ACTL. L'autre approche consiste à spécifier les propriétés en lotos et la vérification de la relation de conformité entre le système de transitions de l'interface et celui de la propriété est réalisée à l'aide de l'outil CAESAR/ALDEBARAN [Fernandez *et al.*, 1992].

L'avantage de Lotos est qu'il est structuré et permet une description formelle de l'architecture, mais le problème de l'approche Lotos est qu'au moment de la génération des systèmes de transitions, la spécification est traduite vers le Lotos basique ce qui implique la perte des données spécifiées. Le problème se pose essentiellement pour les évènements gardés. Avec la perte des gardes, le système obtenu n'est pas équivalent à celui d'origine. Pour éviter ce problème, il faut éviter l'utilisation des gardes dans la spécification ce qui limite le pouvoir d'expression du langage. Un autre inconvénient est que si le langage lotos est assez abordable par la communauté des non experts dans le domaine des techniques formelles, ce n'est pas le cas pour la spécification algébrique des données ACT-ONE. Ces travaux n'offrent pas une interface aidant le concepteur d'IHM à utiliser cette technique sans être un expert.

Lotos a été utilisé également pour les IHM multimodales. Les travaux [Coutaz *et al.*, 1993a] [Paterno & Mezzanotte, 1994] utilisent lotos pour décrire les interacteurs modélisant l'interface de l'application multimodale *Matis*. La modélisation de l'IHM3 démarre d'une modélisation des tâches, en utilisant la notation UAN. Cette modélisation est raffinée jusqu'au niveau des tâches interactives élémentaires. Les interacteurs ont été implémentés en Lotos. Un ensemble de propriétés d'utilisa-

bilité a été exprimé dans la logique temporelle ACTL et vérifié par le contrôleur sur modèle de l'outil Lite.

2.6.3 Approche synchrone à flots de données : Utilisation de Lustre

Lustre [Halbwachs *et al.*, 1991] est un langage synchrone à flot de données. Il a été utilisé dans les travaux de Pierre Roche [Roche, 1998] pour décrire des structures d'interacteurs adaptées de celle de York.

Chaque interacteur est modélisé par un noeud Lustre et l'ensemble de l'interface est conçu en interconnectant un ensemble de noeuds. Une description de l'interface est faite d'abord à l'aide d'Uil/x [HALL, 1991], un langage décrivant la hiérarchie des objets de la présentation. Ensuite, elle est traduite en noeuds lustre. Un interacteur est un noeud qui a en entrée/sortie les flots booléens correspondants aux signaux d'action, de réaction, d'activation, de représentation et aux états internes du modèle d'interacteur.

A tout objet interactif un interacteur Lustre peut être associé dont la structure générique est la suivante :

Node interacteur(liste flots *réaction*, liste flot *action* : bool)
returns (liste flots *activation*, liste flots *représentation* : bool) ;
var liste flots *internes* : bool ;
Let
- équations des flots internes ;
- équations des flots de sortie ;
Tel ;

Ainsi la modélisation en lustre de l'interacteur correspondant au Bouton à enfoncer est donné par :

enfonce, *relâche* et *dessus* sont des flots en entrée. *vidéo_inv* et *active* sont des flots en sortie. Le positionnement du curseur de la souris (*dessus*) sur le bouton affecte l'état *selecte* au bouton. L'enfoncement ($enfonce$) du bouton gauche de la souris affecte l'état *vidé_inv*. Son relâchement (*relâche*) provoque l'activation (*active*) du traitement attaché au bouton.

L'équation définissant le flot *vidéo_inve* signifie que le bouton est en vidéo inverse entre le moment où le bouton gauche de la souris est enfoncé (flot $enfonce$) et relâché (flot *relâche*).

Dans cette approche les propriétés sont exprimées dans le même formalisme que l'interface. Une propriété est spécifiée par un noeud Lustre et elle est vérifiée si le flot

```
Node Bouton (enfonce, relâche, dessus : bool)
        returns (vido_inv, active : bool);
            var inverse, sélecte : bool;
    Let
            inverse = dejusqua (enfonce, relâche) and dessus;
            sélecte = dejusqua (dessus, not dessus);
            active = sélecte and Top (not inverse);
            vido_inv = sélecte and inverse;
    Tel;
```

de sortie du noeud est vrai. La vérification est effectuée avec l'outil Lesar. Contrairement aux autres model-checker, dans l'approche lustre, l'automate représentant le système n'est pas généré avant le processus de vérification, mais pour chaque propriété un nouvel automate est généré.

Un des avantages du formalisme choisi, est que le système ainsi que ses propriétés sont exprimés dans le même formalisme, ce qui évite à l'utilisateur de maîtriser deux formalismes différents. Par contre les propriétés vérifiées par lustre sont limitées par rapport à celles exprimées en logiques temporelles. Seules les propriétés de sûreté et une forme restrictive de vivacité peuvent être vérifiées. Ainsi, la vérification de la réalisation d'une tâche utilisateur ne peut pas être automatisée.

D'autres travaux utilisent Lustre/Lutess pour effectuer les tests logiciels afin de vérifier si le système developpé vérifie les propriétés souhaitées. Les travaux [Jourde *et al.*, A paratre 2006] et [Bouchet *et al.*, 2006] utilisent l'outil de vérification de logiciel réactif synchrone Lutess [du Bousquet *et al.*, 1999] pour vérifier un système interactif multimodal développé avec la plateforme ICARE. Lutess construit automatiquement un générateur de données de test pour le programme sous test. Dans ces travaux, les propriétés CARE ont été spécifiées en Lustre, ensuite vérifiées à l'aide de l'outil Lutess. Ces travaux n'utilisent pas les méthodes formelles au cours de la conception du système mais y font appel à la fin du développement du système multimodal afin d'effectuer les tests logiciels.

2.6.4 Les réseaux de petri : Utilisation des ICO

Philippe Palanque et al. [Bastide & Palanque, 1990] [Palanque *et al.*, 1995] utilisent le formalisme des objets coopératifs et interactifs (ICOs) (Interactive Cooperative Objects). C'est une technique de description formelle dédiée à la spécification, modélisation et implémentation des systèmes interactifs. Elle utilise les concepts de l'approche orientée objet (instanciation dynamique, classification, encapsulation, héritage) pour la description de la partie statique du système et utilise les réseaux de pétri pour la description de sa partie dynamique ou comportement.

Dans cette approche le système est modélisé en général selon le modèle d'architecture de SEEHEIM. Une fois que l'interface est décrite dans le formalisme ICO, la description est ensuite traduite vers les réseaux de pétri standards. La vérification est effectuée par analyse du graphe du marquage du réseau de pétri obtenu.

Un ensemble de propriétés du dialogue a été vérifié telles que : l'absence de blocage, re-initialisabilité et les états atteignables.

La vérification effectuée peut concerner toute ou une partie du système, mais dans cette approche, la spécification des propriétés n'est pas assistée par un langage de description des propriétés d'interaction. De même, aucune validation de tâche n'est effectuée et seules les propriétés d'atteignabilité sont vérifiées.

Les travaux [Palanque & Schyn, 2003], [Schyn *et al.*, 2003], [Navarre *et al.*, 2005] et [Schyn, 2005] utilisent le modèle ICO pour représenter la fusion de deux modalités : le geste et la parole.

Pour prendre en compte les spécificités de l'interaction multimodale, le formalisme des ICO a été étendu avec un ensemble de mécanismes tel que le mécanisme de communication par production et consommation d'événements. Ce mécanisme permet de synchroniser un ensemble de transitions sur un événement, et permet d'émettre un événement lors du franchissement d'une transition. Des interactions d'une application de réalité virtuelle ont été présentées dans [Schyn *et al.*, 2003]. Les interactions sont modélisées à plusieurs niveaux d'abstraction dont le premier niveau concerne les actions physiques faisant intervenir les modalités. La figure 2.5 montre le fonctionnement de la souris. A chaque déplacement de la souris, un événement *Move* est généré par cet ICO, et ceci quelque soit l'état dans lequel il se trouve. Les événements *Pick* et *Drop* sont générés suite aux clics de la souris d'une manière alternative.

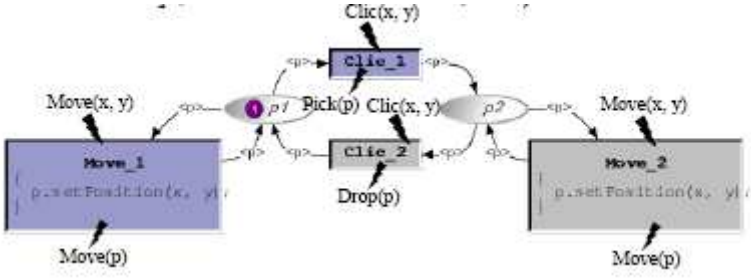

Fig. 2.5 – Comportement de la souris

Le contrôleur de dialogue est modélisé en se basant sur les événements provenant du bas niveau d'interaction. Le contrôleur de dialogue est conçu en fonction d'événements abstraits résultant de la fusion des événements de bas niveau. Ainsi, le changement de médias d'entrée ne perturbe pas le fonctionnement du contrôleur de dialogue. La figure 2.6 montre l'utilisation des événements *Move*, *Pick* et *Drop* pour la modélisation du dialogue. Le dialogue est ainsi modélisé indépendamment

du média d'entrée.

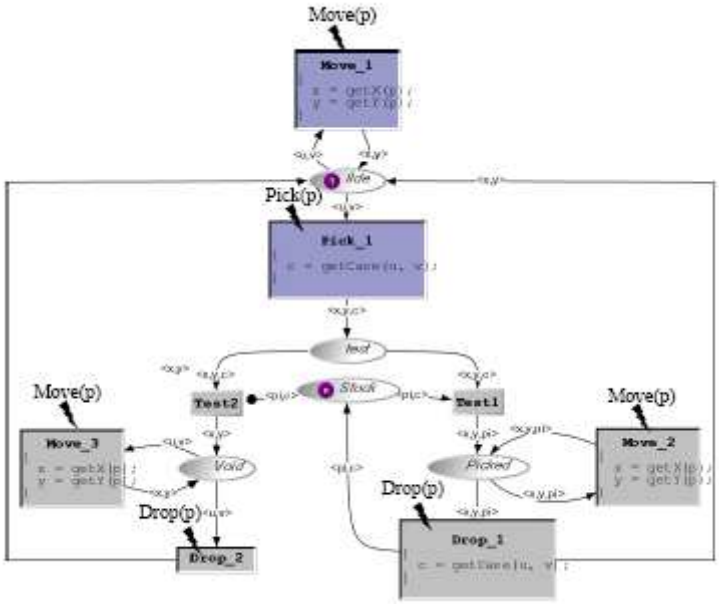

Fig. 2.6 – Comportement du controleur de dialogue

Ces travaux ne traitent pas les propriétés d'utilisabilités des IHM multimodales.

2.6.5 Technique de preuve : La méthode B

Plusieurs travaux utilisant la méthode B [Abrial, 1996a] ont été réalisés au sein de l'équipe d'IHM du LISI. Dans un premier temps [Ait-Ameur *et al.*, 1998a] et [Ait-Ameur *et al.*, 1998b] se sont intéressés à la formalisation des spécifications d'un système interactif opérationnel tout en assurant certaines propriétés ergonomiques. A partir d'études de cas, ces travaux ont permis de vérifier des propriétés comme l'observabilité, l'insistance ou la robustesse. Dans les travaux de [Ait-Ameur,], une application est implémentée par raffinement jusqu'à génération du code dans un langage de programmation. Un ensemble de propriétés ont été vérifiées. Dans [Ait-Ameur *et al.*, 2003], une expérience est réalisée en utilisant les deux techniques : preuve et model-checking.

Dans ces travaux plusieurs notations, souvent utilisées par les concepteurs d'IHM, ont été mises en oeuvre. Ces travaux utilisent le modèle ARCH comme architecture utilisée pour la description et la spécification des différents composant du logiciel.

Contrairement aux travaux basés sur la technique de model-checking, ces travaux échappent au problème d'explosion combinatoire grâce au processus de raffinement et l'outil Atelier B qui génère automatiquement toutes les obligations de preuves qui sont déchargés par le prouveur. Par contre, le processus de preuve n'est pas

complètement automatique. L'utilisateur peut avoir à conduire la preuve de manière semi-automatique.

D'autres travaux utilisant la même approche ont été réalisés, en s'intéressant à une architecture orientée objet, telle que MVC ou PAC, au lieu du modèle ARCH. Comme les contraintes du langage B ne permettaient pas d'utiliser directement les modèles à agents, une nouvelle architecture hybride appelée CAV [Jambon, 2002] a été développée. Ces travaux s'intéressent à la vérification des propriétés du noyau fonctionnel.

Dans les deux types de travaux, le contrôleur de dialogue n'est pas modélisé et les tâches ne sont pas validées. Dans [Baron, 2003], le B événementiel a été utilisé ce qui a permis, non seulement de vérifier des propriétés de l'IHM, mais de valider des tâches modélisées en CTT et de modéliser le contrôleur de dialogue. Nous citons également les travaux [Ait-Sadoune, 2005], [Ait-Ameur *et al.*, 2006a] et [Ait-Ameur *et al.*, 2006b] qui présentent l'utilisation de cette technique pour la vérification des IHM multimodales. Nous présentons dans le chapitre 5 l'utilisation de la méthode B pour la mise en oeuvre de notre modèle formel pour modéliser et vérifier les IHM multimodales.

2.6.6 Technique de preuve : Z

La méthode Z a été également utilisée, dans les travaux de Duke et Harisson, pour la description des IHM. Dans [Duke & Harisson, 1997] les interacteurs ont été modélisés en utilisant le langage Z. Une spécification en Z est décrite en plusieurs étapes de raffinement et les propriétés ont été vérifiées à l'aide d'invariants.

Les travaux [MacColl & Carrington, 1998] utilisent Z et CSP [Hoare, 1985] pour modéliser les systèmes multimodaux. Les auteurs utilisent Z pour modéliser l'aspect statique de l'interface et l'algèbre CSP pour l'aspect dynamique. Ils modélisent le système Matis en utilisant les interacteurs.

Ces travaux font abstraction des modalités et évitent ainsi de prendre en charge les spécificités des systèmes multimodaux. Les problèmes de fusion et fission des modalités ainsi que la vérification des propriétés ne sont pas abordés.

2.6.7 Les machines à états

Les travaux de [Bourguet, 2003] [Bourguet, 2002], montrent que les machines à états constituent un modèle adéquat pour modéliser l'interaction multimodale en entrée. Ils montrent que ce modèle formel permet aussi la modélisation des différentes contraintes temporelles et de synchronisation. La figure 2.7 montre une transition dans une machine à état. Une transition est caractérisée par un *état source* et *un état cible*. Lorsque la machine se trouve dans l'état source et l'*évènement* associé à la transition se produit, l'*action* associée à celle-ci est exécutée.

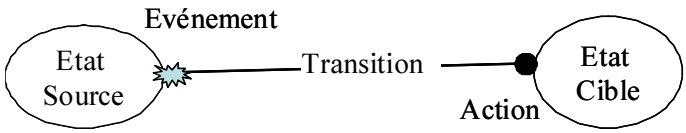

Fig. 2.7 – Une transition d'une machine à état

Pour spécifier les machines à états, l'outil IMBuilder a été développé. La figure 2.8 montre un exemple de machine à état conçu à l'aide de cet outil. Une liste d'entrées valides ainsi qu'une liste de commandes peuvent être créées ou importées d'un fichier XML.

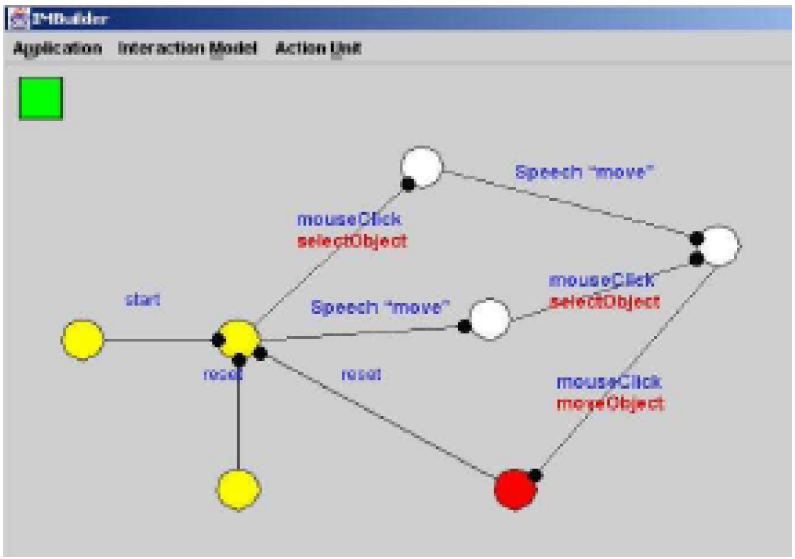

Fig. 2.8 – Machine à état modélisant la commande "déplacer un objet", construite avec IMBuilder

Ces travaux restent au niveau prototypage des interactions multimodales. Aucun modèle d'interaction n'est proposé et les propriétés d'utilisabilité ne sont pas modélisées ni vérifiées. Tous ces travaux ne traitent que partiellement la modélisation de l'interaction multimodale. Ils s'intéressent uniquement à prouver que telle ou telle technique ou modèle formel peut aider à modéliser l'interaction multimodale. Aucun modèle de conception n'est pris en compte, ni une modélisation formelle des propriétés CARE n'est proposée. Notre travail, par contre, est plus générique quant à l'utilisation des techniques formelles. Il propose un modèle générique pour le système ainsi que pour les propriétés.

2.7 Méthodologie et hétérogénéité

Comme les systèmes interactifs sont composés de deux parties : Noyau fonctionnel et IHM, nous décrivons les différentes démarches de conception pouvant être

envisagées lors de leur développement. Le noyau fonctionnel ainsi que l'IHM sont modélisés chacun par un système de transitions [Ait-Ameur *et al.*, 2006a].

Approche 1. Dans cette approche, le noyau fonctionnel du système est développé indépendamment de l'IHM. Ensuite, ce dernier est raffiné en ajoutant le modèle de L'IHM correspondant avec les interactions de l'utilisateur. Dans ce cas les propriétés déjà vérifiées et liées au noyau du système seront préservées et il n'est plus nécessaire de les vérifier de nouveau dans le système final.

Approche 2. Dans cette approche, l'IHM est développée en premier lieu sans tenir compte des aspects du noyau fonctionnel du système à développer. Ensuite, le système obtenu est raffiné avec les éléments du noyau fonctionnel. Dans ce cas, les propriétés de l'IHM déjà vérifiées seront préservées par le raffinement et ne seront pas obligées d'être vérifiées une seconde fois sur le système global obtenu.

Approche 3. Dans cette approche, l'IHM et le noyau fonctionnel sont développés chacun d'un coté. Le modèle du système est obtenu par composition du système décrivant le noyau fonctionnel et celui décrivant l'IHM. La composition est réalisée par le produit synchronisé des deux systèmes de transitions. Dans cette approche la revérification de toutes les propriétés peut être nécessaire car leur préservation par le produit synchrone n'est pas toujours garantie.

La première et la deuxième approchent peuvent être mises en oeuvre dans une technique permettant le raffinement telle que la méthode B. Dans ce cas, les systèmes de transitions sont décrits dans le langage B. La vérification des propriétés dans ce cas, consiste à vérifier que les tâches utilisateurs modélisées en CTT et codées dans le langage B raffinent bien le système interactif. Cette démarche est montrée dans le chapitre 5.

La troisième approche peut être mise en oeuvre dans une technique de model-checking. Deux exemples de ce type de techniques sont implémentés dans les model-checker SMV et SPIN. Dans Le cas de SMV, les propriétés sont exprimées dans une logique temporelle CTL, le système de transitions est décrit dans le langage d'entré de SMV et la vérification consiste à parcourir l'espace d'états du système pour vérifier si la formule logique exprimant la propriété est vérifiée. Un algorithme symbolique est utilisé dans ce cas. Il consiste à représenter l'espace d'états de façon à regrouper un ensemble d'états dans une seule représentation afin de minimiser la taille du système. Dans le cas de SPIN, les propriétés sont exprimées dans la logique temporelle LTL, le système est décrit dans le langage d'entrée de SPIN. La vérification est effectuée par l'algorithme de büchi qui consiste à vérifier que l'intersection du langage généré par l'automate correspondant au système de transitions de la négation de la propriété et le langage généré par l'automate correspondant au système interactif est un ensemble vide. Autrement dit, l'algorithme vérifie qu'aucun comportement non souhaité (négation de la propriété) n'est généré par le système

interactif.

Ces techniques utilisent des modèles hétérogènes, et diffèrent en puissance d'expression, complexité de modélisation, temps d'exécution, etc. Une modélisation générique pour le système et les propriétés, et indépendante de toutes ces techniques est nécessaire. C'est ce que nous proposons dans le chapitre 3. La généricité de cette démarche permet de passer vers l'une ou l'autre des approches citées plus haut. Ceci permet de faire coopérer plusieurs techniques formelles en profitant de la puissance de chacune d'elle. Mais dans ce cas, il faudrait définir les règles de passage du modèle générique vers la technique formelle et valider ces règles de manière à assurer que le modèle défini dans telle ou telle technique est bien celui du modèle générique.

2.8 Conclusion

Après avoir montré les différentes approches des méthodes formelles ainsi que leur application dans le domaine des systèmes interactifs, nous constatons qu'aucune méthode ou technique ne permet de couvrir les besoins de spécification, de modélisation et de vérification des systèmes interactifs et en particulier les systèmes multimodaux.

Nous constatons également qu'aucune de ces techniques ne s'est imposée dans ce domaine. Les techniques basées sur la vérification sur modèle sont complètement automatisables mais souffrent du problème de l'explosion combinatoire. Quant aux techniques basées sur la preuve, elles offrent une conception par raffinement du système mais sont semi automatiques et ne permettent pas de vérifier toutes les propriétés que les logiques temporelles peuvent exprimer. Toutes ces techniques ne couvrent que partiellement le cycle de vie du développement d'une IHM3.

Les interfaces multimodales sont caractérisées principalement par la concurrence des interactions. Les travaux utilisant les modèles formels pour spécifier l'interaction multimodales restent insuffisants pour couvrir tous les problèmes de conception de ce type d'IHM. Les quelques expériences citées dans la section 2.6 restent limitées. Aucune approche n'est générique, mais elles sont toutes liées à une technique donnée.

Chacune de ces techniques a été utilisée pour couvrir une partie du cycle du développement du système interactif.

De même, ces travaux n'utilisent pas un même modèle formel pour spécifier tous les composants du système de conception. Les travaux que nous avons étudiés ne prennent pas en compte les espaces de conception définis par la communauté des IHM3. De même, aucun travail n'a été fait pour modéliser formellement les propriétés CARE.

2.9 Notre proposition

Les travaux présentés précédemment

1. ne couvrent que partiellement le cycle de vie de développement des IHM3 ;
2. sont spécifiques à une technique formelle particulière. En général cette technique n'est efficace que pour une situation particulière comme l'expression du système ou la vérification de telle ou telle propriété ;
3. ne respectent que partiellement les modèles proposés par les concepteurs d'IHM3 comme les espaces de conception ou les propriétés CARE ;
4. ne proposent aucune démarche d'intégration des différentes techniques afin de tirer bénéfice des résultats obtenus à l'aide de chaque technique.

Nos travaux visent à répondre aux insuffisances constatées dans notre étude bibliographique des chapitres 1 et 2. Notre objectif est de proposer un modèle qui soit générique, dans le sens où il sera utilisé pour la modélisation de l'interaction multimodale ainsi que les propriétés d'utilisabilité CARE. Nous proposons un modèle générique pour :

- décrire et modéliser l'interaction multimodale en respectant les types de multimodalité définis par la communauté des chercheurs dans ce domaine ;
- modéliser les propriétés d'utilisabilité CARE et proposer des techniques de vérification ;
- les tâches utilisateurs afin de décrire les différents comportements attendus.

Ce modèle est formel et générique. Il se base sur les algèbres de processus dont la sémantique est définie par les systèmes de transitions.

Les modèles formels que nous proposons sont génériques dans le sens où ils ne sont pas liés à une technique formelle particulière mais peuvent être codés dans n'importe quelle technique supportant les systèmes de transitions comme modèle sémantique.

Nous montrerons qu'une fois le système modélisé, ainsi que ses propriétés décrites, il peut être prouvé selon les techniques choisies. Pour cela, nous l'avons mis en oeuvre dans trois outils différents : SMV, SPIN et B. Les deux outils SMV et SPIN se basent sur la technique du model-cheking où les propriétés sont exprimées, respectivement, dans les logiques temporelles CTL et LTL. L'outil B se base sur la technique de preuve. Notons que chaque technique n'est pas en mesure d'exprimer tous les besoins en conception apparaissant dans les IHM3. Mais, l'utilisation de différentes techniques formelles à partir d'un même modèle générique contribue à une meilleure intégration des modèles de conception.

Dans le chapitre suivant, nous présentons le modèle générique de conception de l'interaction multimodale ainsi que ceux exprimant les propriétés d'utilisabilité CARE.

Chapitre 3

Un modèle conceptuel formel générique pour les systèmes interactifs multimodaux. Cas de la modalité en entrée

3.1 Introduction

Dans ce chapitre, nous présentons notre modèle formel pour la conception de l'interaction en entrée pour les IHM multimodales ainsi que pour les propriétés d'utilisabilité CARE. Le modèle que nous proposons pour modéliser l'interaction multimodale est générique et peut être mis en oeuvre dans toute technique formelle supportant la description de modèles de systèmes de transitions que nous avons choisi comme modèle sémantique pour la description du comportement dynamique d'une IHM multimodale.

Nous proposons de modéliser l'interaction multimodale en entrée ainsi que les propriétés d'utilisabilité CARE. Pour cela nous définissons :

1. un modèle formel générique permettant de modéliser l'espace de conception présenté dans le chapitre 1. Nous avons repris les contraintes relevées et les différents types d'IHM multimodales qui en découlent. Ainsi, ces modèles permettent au concepteur de définir son IHM selon le type désiré et choisi. Le cadre numéro 5 sur la figure 3.1 positionne notre modèle par rapport à un système multimodal ;
2. une spécification formelle des propriétés CARE décrites sur le modèle générique précédent. Nous proposons deux types de spécification : l'un fondé sur un modèle opérationnel et l'autre fondé sur un modèle logique. Le modèle opérationnel décrit la propriété par un comportement modélisé par un système de transitions. Le modèle logique décrit la propriété par une formule logique qui

sera interprétée sur le modèle décrivant le système.

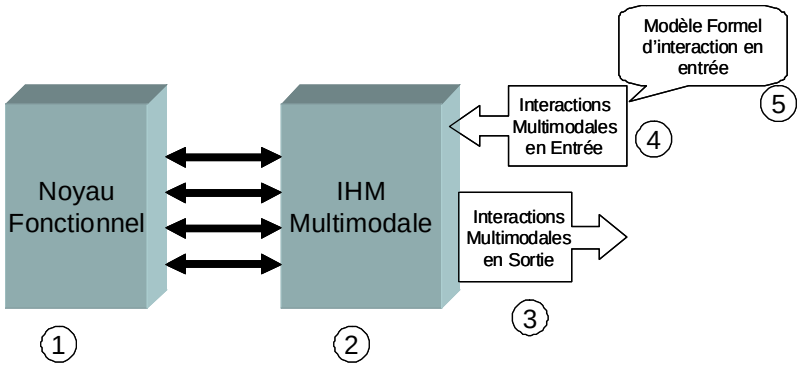

Fig. 3.1 – Système multimodal

3.2 Modélisation de l'interaction en entrée dans les IHM multimodales

L'IHM multimodale est caractérisée par la présence de plusieurs manières d'interagir avec l'utilisateur. Ce dernier peut formuler ses commandes en utilisant une ou plusieurs modalités. Elle est caractérisée également par la présence du parallélisme dans l'utilisation de ces modalités. Une modélisation de l'interaction avec ce type d'interface doit prendre en compte ces aspects et permettre leur expression.

Il existe deux types d'interactions : l'interaction en entrée (cadre numéro 4 sur la figure 3.1) provoquée par l'utilisateur pour communiquer avec le système, et l'interaction en sortie (cadre numéro 3 sur la figure 3.1) provoquée par le système pour communiquer avec l'utilisateur. Nous nous intéressons à la modélisation de l'interaction en entrée et à sa fusion. Le même raisonnement peut être fait pour l'interaction en sortie.

L'idée derrière notre travail est de proposer un modèle qui permet de capturer les concepts de modalités pour les interactions et le parallélisme présents au niveau des tâches d'interaction, des commandes de base (énoncés) et des interactions de base. Un modèle générique permet une conception indépendante de tout outil ou technique formelle pouvant l'implémenter. Pour cela, nous nous sommes inspirés des algèbres de processus pour proposer notre modèle de conception d'IHM multimodale [Ait-Ameur & Kamel, 2004]. La syntaxe du modèle est donnée sous forme de grammaire BNF augmentée. Elle se base sur un ensemble d'opérateurs de composition de base et un ensemble d'éléments syntaxiques caractérisant les interactions et leurs modalités. La sémantique du modèle est donnée à l'aide des systèmes de transitions qui permettent de modéliser le comportement dynamique de l'IHM.

Le concepteur décrit les différentes interactions multimodales en utilisant la grammaire du modèle. Un modèle sémantique peut être défini à partir de la description syntaxique. Une fois le modèle de l'IHM conçu, une mise en oeuvre peut être réalisée dans toute technique formelle supportant les systèmes de transitions afin de vérifier des propriétés, de valider des tâches de l'utilisateur, ou de concevoir le système interactif.

En premier lieu, nous proposons un modèle générique qui permet toutes les compositions (séquentielle, parallèle, etc.) de tout type de modalité à tous les niveaux d'abstraction, que ce soit au niveau des commandes de base ou des tâches plus complexes.

En seconde étape, nous proposons un modèle qui permet de concevoir une IHM multimodale selon l'espace de conception présenté dans le chapitre 1. Ceci est réalisé en imposant des contraintes sur l'utilisation des modalités et les opérateurs de composition. Ainsi, le concepteur peut modéliser une IHM selon le type de la multimodalité souhaité.

Avant de présenter la syntaxe et la sémantique de nos modèles, nous donnons quelques définitions précisant certains termes nécessaires pour leur définition.

3.2.1 Eléments de base

Les notions de *modalité*, *interaction*, *énoncé* et *tâche* sont utilisées dans la conception d'une IHM multimodale. Ces mêmes notions sont reprises dans notre modèle avec des définitions adaptées. Ceci est dû au caractère abstrait de notre modèle. Nous donnons dans ce qui suit nos définitions informelles de ces notions que nous définissons formellement par la suite dans les sections qui présenteront nos modèles.

3.2.1.1 Modalité

Nous avons vu dans les chapitres précédents la diversité des définitions données aux concepts liés aux systèmes multimodaux et en particulier les définitions données à la modalité. Nous proposons un modèle formel qui abstrait les détails du niveau physique mais doit capter le caractère de diversité de modes d'interactions. Les interactions peuvent provenir de plusieurs sources, ce qui explique leur caractère concurrent et parfois coopératif. Nous proposons ainsi une abstraction de la notion de modalité que nous ne définissons pas par rapport aux dispositifs physiques ni par rapport au processus de traitement des informations. Nous définissons une modalité comme un type d'actions interactives de base. Le concepteur distingue les actions de base et les classes selon les modalités arrêtées par ce dernier.

Ainsi, si le concepteur prend la définition qui associe une modalité à un dispositif physique, les actions seront typées par rapport aux dispositifs physiques du système. Dans le cas où il prend la définition qui associe la modalité au couple (dispositif, lan-

gage d'interaction) alors les actions sont typées par rapport à ce couple qui distingue les types d'actions interactives de base.

Cette définition garantit la généricité de notre modèle. Elle permet de modéliser l'interaction multimodale quelle que soit la définition prise pour la modalité.

3.2.1.2 Action interactive

Nous définissons par *interaction* ou *action interactive de base*, l'action de base que l'utilisateur réalise en communiquant avec l'IHM. Par exemple le clic de la souris sur un objet, une phrase prononcée et reconnue par l'analyseur du langage naturel, etc. Ce sont des actions non décomposables et chacune d'elle est produite par une seule modalité. Ce sont les actions de granularité la plus faible. Nous ne définissons pas la granularité des actions mais c'est au concepteur de l'IHM de définir cette granularité et qui permet de distinguer les actions de base des actions composées. C'est en fonction de ces actions interactives de base que le système de transitions qui décrit le comportement de l'IHM est défini.

3.2.1.3 Enoncé

Les actions interactives de base peuvent être composées d'une manière séquentielle ou parallèle pour participer à la réalisation d'un traitement élémentaire que nous définissons par *énoncé*. En d'autres termes, un énoncé est une commande qui appelle une fonction élémentaire du noyau fonctionnel. Une action interactive de base peut être suffisante pour appeler une fonction élémentaire du noyau fonctionnel, mais parfois il est nécessaire d'utiliser plusieurs actions de base pour formuler une commande simple.

Dans le cas où l'énoncé est produit par la composition d'un ensemble d'actions de même modalité on parle d'énoncé monomodal. Dans le cas où les actions sont issues de plusieurs modalités, on parle d'énoncé multimodal.

3.2.1.4 Tâche d'interaction

Une tâche d'interaction est un ensemble d'énoncés composés de manière séquentielle ou parallèle pour réaliser un traitement donné. Une tâche d'interaction est identifiée par un état initial est un état final et un ensembles de chemins ou traces d'actions interactives de base. La tâche d'interaction la plus élémentaire est celle définie par un énoncé.

3.2.2 Modèle générique de fusion d'interactions multimodales

Le modèle générique Mg, que nous présentons dans cette section, modélise la fusion en entrée de l'interaction multimodale sans contraintes sur les utilisations des

modalités. Toutes les combinaisons sont possibles, que ce soit pour les modalités ou la composition temporelle entre les modalités (séquentielle ou parallèle). Le modèle est inspiré des algèbres de processus telles que CCS et Lotos largement utilisées dans la modélisation des systèmes concurrents et des protocoles de communication.

3.2.2.1 Syntaxe

La syntaxe du modèle est donnée à l'aide d'une grammaire avec des règles BNF. Un ensemble d'opérateurs est défini pour composer les interactions. Nous avons défini un ensemble d'opérateurs de base suffisant pour exprimer le caractère séquentiel et parallèle des interactions. Tout autre opérateur peut être défini à partir de ces opérateurs de base. Le choix de ces opérateurs est guidé par soucis de simplicité. Les opérateurs sont simples et suffisants pour exprimer tout autre opérateur plus complexe.

Nous notons par $\boldsymbol{A}$ l'ensemble des actions interactives de base. Un élément de cet ensemble est noté a. La syntaxe du modèle ***Mg*** est donnée par la grammaire suivante :

$$\begin{array}{lll} T & ::= & T[]T \mid T >> T \mid T \,|||\, T \mid T \,||\, T \mid Enonce \\ Enonce & ::= & a; Enonce \mid a \,|||\, Enonce \mid a \,||\, Enonce \mid \delta \text{ avec } a \in A \end{array}$$

La première règle définit les tâches d'interaction possibles en composant les différents énoncés en tâches plus complexes. La seconde règle définit les énoncés en fonction des actions interactives de base. La signification informelle des différents opérateurs est donnée comme suit : Soient P, P', Q et Q' des termes du modèle ***Mg*** et $''a''$ un élément de l'ensemble $\boldsymbol{A}$ alors :

- δ est un terme qui ne fait rien (mot vide) ;
- **[]** est l'opérateur de choix non déterministe. L'exécution de $P[]Q$ est soit celle de P, soit celle de Q ;
- $>>$ est l'opérateur de séquence. L'exécution de $P >> Q$ est celle de P suivie de celle de Q ;
- **;** désigne le préfixage. L'exécution de $a; P$ est l'exécution de l'action "a" suivie de l'exécution de P ;
- $|||$ désigne l'opérateur de composition parallèle entrelacé. Si P peut exécuter a et se comporte ensuite comme P', alors $P \,|||\, Q$ (resp. $Q \,|||\, P$) peut exécuter "a" pour se comporter ensuite comme $P' \,|||\, Q$ (resp. $Q \,|||\, P'$).
- $||$ est l'opérateur de composition parallèle. Si P peut exécuter "a" et se comporte ensuite comme P', alors $P \,||\, Q$ (resp. $Q \,||\, P$) peut exécuter "a" pour se comporter ensuite comme $P' \,||\, Q$ (resp. $Q \,||\, P'$). De plus $P \,||\, Q$ peut exécuter (a_1, a_2) et se comporter ensuite comme $P' \,||\, Q'$, avec a_1 une action exécutée par P et a_2 une action exécutée par Q.

Exemple. Le terme $(a_1 >> (a_2[]a_3) >> \delta)[](a_4 \,|||\, a_5)$ est généré par la grammaire Mg. a_1, a_2, a_3, a_4 et a_5 sont des actions interactives de base.

3.2.2.2 Sémantique statique

La sémantique statique est définie par un ensemble d'attributs ou propriétés associés aux éléments syntaxiques de la grammaire de Mg. Ces attributs permettent de vérifier certaines propriétés non liées à l'exécution du système et de typer les éléments syntaxiques. L'implémentation de ces attributs peut se faire dans n'importe quelle technique formelle et permet de vérifier certaines propriétés statiques de l'IHM multimodale.

Nous définissons, comme exemples, deux attributs et nous indiquons leurs utilisations.

1. Nous définissons pour chaque action son type de modalité. Ainsi, nous notons par ***Modalité (a)*** la modalité de l'action *a*. Si pour des raisons données, le concepteur de l'IHM veut interdire la mise en parallèle de deux modalités *mi* et *mj*. Ceci peut être réalisé au niveau syntaxique en introduisant une condition sur la compatibilité des modalités pour l'opérateur du parallélisme. Si *a1*, *a2* sont deux actions interactives de base alors cette propriété est exprimée comme suit :

$$a1 \,||\, a2 \; ssi \begin{cases} modalite(a1) \in \{m1, m2\} \Rightarrow modalite(a2) \notin \{m1, m2\} \\ modalite(a2) \in \{m1, m2\} \Rightarrow modalite(a1) \notin \{m1, m2\} \end{cases}$$

2. Nous définissons par ***Durée (a)*** la durée de l'action "a". Pour vérifier l'équivalence non fonctionnelle de deux modalités on peut utiliser l'attribut Durée. Deux modalités sont dites équivalentes si elles permettent de réaliser la même tâche avec la même durée d'exécution. Une évaluation du temps d'exécution d'une tâche est calculée en fonction des durées des différentes actions interactives de base.

3.2.2.3 Sémantique dynamique

La sémantique dynamique ou opérationnelle décrit le comportement dynamique du système. La sémantique opérationnelle associe à un terme de la grammaire Mg, un modèle sous la forme d'un système de transitions étiquetées.

La description des modèles se fait en définissant tout d'abord l'ensemble des états constitué des termes de la grammaire MG, une relation de transition $\rightarrow$ entre les termes de Mg. Cette relation décrit ce qu'on peut observer du comportement d'un terme. Les actions sont les éléments "a" de l'ensemble $\boldsymbol{A}$.

Soient P, P', Q et Q' des termes de la grammaire, *a*, *a1* et *a2* des actions de l'ensemble $\boldsymbol{A}$.

$P \xrightarrow{a} Q$ se lit : P a une transition par "a" menant vers Q, et signifie que dans le comportement de P, on peut observer initialement "a" et que le comportement ultérieur correspond à celui de Q. On dit que Q est un successeur (par "a") de P. Le comportement d'un terme consiste, essentiellement, à exécuter des actions et le comportement d'un terme composé $\mathbf{Op}(P1, \ldots, Pn)$ doit se déduire des comportements de $P1, \ldots, Pn$.

Le domaine de définition de la relation de transitions est

$$\rightarrow \subseteq Mg \times A \times Mg \text{ et } P \xrightarrow{a} Q \text{ où } (P, a, Q) \in Mg \times A \times Mg$$

Elle est décrite par induction structurelle selon le style de G.Plotkin [Plotkin, 1981] au moyen de règles de la forme

$$\frac{Premisses}{Conclusion}$$

- La conclusion est de la forme $\mathbf{Op}(P1, \cdots, Pn) \xrightarrow{a} Q$, où n est l'arité de l'opérateur **Op** ;
- L'ensemble des prémisses (qui peut être vide) est de la forme

$$\{Pi \xrightarrow{a1} P'i / i \in I \subseteq [1, n]\} \cup \{Pjnot \xrightarrow{aj} / j \in J \subseteq [1, n]\}$$

Une telle règle se lit :

Si pour tout i dans I, Pi a une transition par ai vers $P'i$, et pour tout j dans J, Pj n'a pas de transition par aj, alors $\mathbf{Op}(P1, \cdots, Pn)$ a une transition par a vers Q.

Les règles de la relation de transitions sont données comme suit :

L'arrêt δ est un terme qui correspond à l'état d'arrêt. Il correspond au processus STOP de l'algèbre CCS. C'est un état à partir duquel aucune action n'est possible. La figure 3.2 montre la représentation graphique de ce terme.

$$\delta \rightarrow$$

Fig. 3.2 – Système de transitions de l'arrêt

L'opérateur de préfixage ; est un opérateur emprunté de CCS. Un état correspondant au terme $a;P$ peut effectuer une transition en exécutant l'action a et passer à l'état correspondant au terme P. La représentation graphique de cette règle est donnée par la figure 3.3.

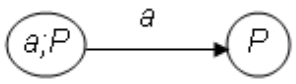

Fig. 3.3 – Système de transitions du préfixage

$$a; P \xrightarrow{a} P$$

L'opérateur de choix [] est un opérateur de CCS et de Lotos. Il y a deux règles de transition pour cet opérateur. Une règle pour définir la composition à gauche et une autre pour définir la composition à droite. Si une transition est possible à partir d'un état correspondant au terme P (resp. Q) pour aller à un autre état correspondant au terme P' (resp. Q'), alors dans le système composé correspondant au terme $P[]Q$ la même transition reste possible et fait passer le système à l'état correspondant au terme P'(resp. Q'). La figure 3.4 montre la représentation graphique de ces règles.

Composition à droite $\dfrac{P \xrightarrow{a} P'}{P[]Q \xrightarrow{a} P'}$

Composition à gauche $\dfrac{Q \xrightarrow{a} Q'}{P[]Q \xrightarrow{a} Q'}$

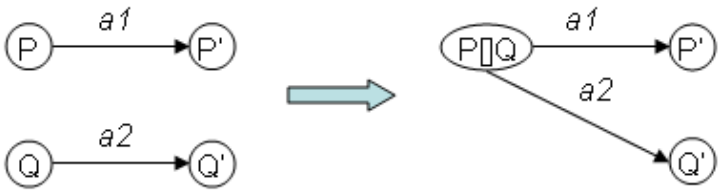

Fig. 3.4 – Système de transitions du choix

L'opérateur de séquence » est un opérateur emprunté de Lotos. Les règles de cet opérateur expriment le comportement de la mise en séquence de deux termes P suivi de Q. Les deux règles expriment le fait que $P >> Q$ se comporte comme P jusqu'à ce qu'il atteigne la fin de son exécution, ensuite il se comportera comme Q. La figure 3.5 montre la représentation graphique de ces règles.

$$\frac{P \xrightarrow{a} P' et P' \neq \delta}{P >> Q \xrightarrow{a} P' >> Q} \qquad \frac{P \xrightarrow{a} P' et P' = \delta}{P >> Q \xrightarrow{a} Q}$$

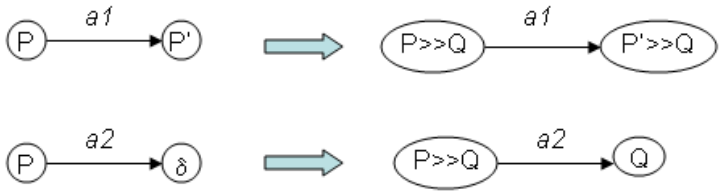

Fig. 3.5 – Système de transitions de la séquence

L'opérateur d'entrelacement ||| est un opérateur emprunté de Lotos. Les règles de cet opérateur expriment le comportement parallèle entrelacé de deux termes. Si de l'état P le système transite par *a1* vers *P'*, alors l'état composé $P \,|||\, Q$ transite par *a1* vers l'état composé $P' \,|||\, Q$. De la même manière, si de l'état Q le système transite par *a2* vers l'état *Q'*, alors l'état composé $P \,|||\, Q$ transite par *a2* vers l'état $P \,||\, Q'$. La figure 3.6 montre la représentation graphique de ces règles.

Composition à droite $\dfrac{P \xrightarrow{a} P'}{P|||Q \xrightarrow{a} P'|||Q}$

Composition à gauche $\dfrac{Q \xrightarrow{a} Q'}{P|||Q \xrightarrow{a} P|||Q'}$

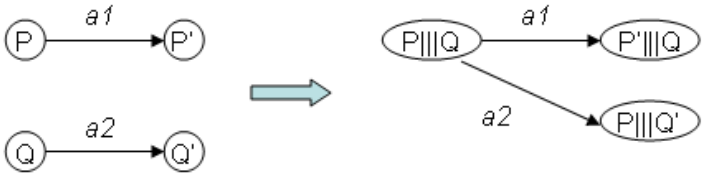

Fig. 3.6 – Système de transitions de l'entrelacement

L'opérateur de parallèle || est un opérateur emprunté de Lotos. Contrairement à l'opérateur précèdent qui ne permet qu'une seule action à s'exécuter à la fois. L'opérateur parallèle || autorise l'exécution d'une action du premier système ou bien une action du deuxième système ou bien les deux actions en même temps issues des deux systèmes. Nous avons imposé une condition sur les modalités des actions mises en parallèle. Nous considérons qu'une modalité ne peut produire deux actions en même temps. C'est pour montrer l'utilisation de la sémantique statique introduite dans la section précédente. La figure 3.7 montre la représentation graphique de ces règles.

Composition à droite $\dfrac{P \xrightarrow{a} P'}{P||Q \xrightarrow{a} P'||Q}$

$$\textit{Composition à gauche}\ \frac{Q \xrightarrow{a} Q'}{P||Q \xrightarrow{a} P||Q'}$$

$$\frac{P \xrightarrow{a1} P' \textit{ et } Q \xrightarrow{a2} Q' \textit{ avec } a1 \in A_{mi},\ a2 \in A_{mj} \textit{ et } modalite(a1) \neq modalite(a2)}{P||Q \xrightarrow{(a1,a2)} P'||Q'}$$

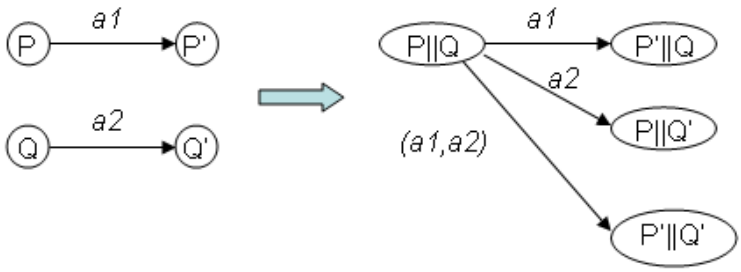

Fig. 3.7 – système de transitions du parallèle

Exemple : En appliquant les règles sémantiques des opérateurs définis dans la section 3.2.2.3, nous construisons le système de transitions du terme

$$\boxed{(a_1 >> (a_2 [] a_3) >> \delta) [] (a_4 \,|||\, a_5)}$$

présenté dans la section 3.2.2.1. Pour cela, nous procédons par étape, en construisant les systèmes de transitions des sous termes, puis nous les composons afin d'obtenir le système global. Pour des raisons de représentation nous nommons les sous termes comme suit :

$$\boxed{\begin{array}{l} \text{T0} = (a_1 >> (a_2 [] a_3) >> \delta) [] (a_4 \,|||\, a_5) \\ \text{T1} = a_1 >> (a_2 [] a_3) \\ \text{T2} = (a_2 [] a_3) \\ \text{T3} = (a_4 \,|||\, a_5) \end{array}}$$

Les figures 3.8 et 3.9 montrent respectivement les systèmes de transitions des termes T1 et T3. La figure 3.10 montre le système de transitions global correspondant au terme T0 (T1[]T3).

3.2.3 Modèle paramétré de fusion d'interactions multimodales

Le modèle présenté dans la section précédente définit l'interaction multimodale de façon générique sans prendre en considération le type de multimodalité de l'interface. En effet, l'ensemble ***A*** contient l'ensemble des actions produites par toutes

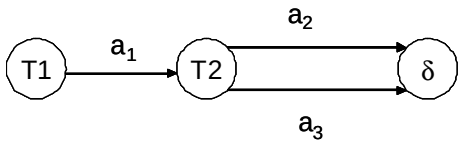

Fig. 3.8 – Système de transitions du terme T1

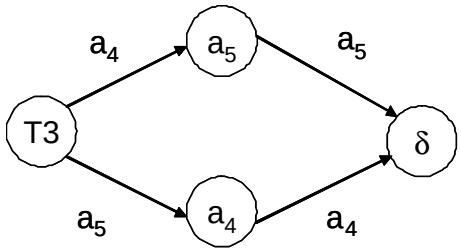

Fig. 3.9 – Système de transitions du terme T3

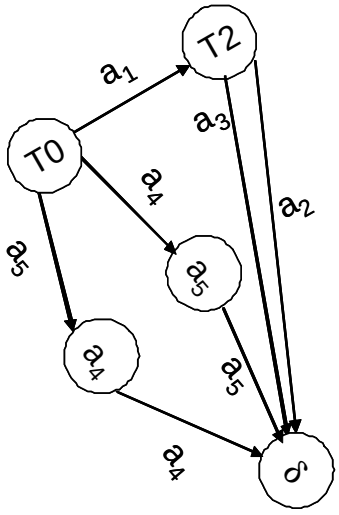

Fig. 3.10 – Système de transitions du terme T0

les modalités du système. Dans cette section nous proposons de paramétrer le modèle afin de permettre la conception d'IHM selon le type de multimodalité choisie. Pour cela, nous définissons de nouveaux éléments syntaxiques nécessaires pour définir des modèles par type de multimodalité. Ainsi, le concepteur choisira le modèle correspondant pour modéliser l'IHM en fonction du type de multimodalité souhaité.

L'idée est d'identifier les actions interactives de base par rapport aux modalités qui les produisent ainsi que les opérateurs utilisés à chacun des niveaux d'abstraction. Nous distinguons deux niveaux d'abstraction principaux : un niveau pour les énoncés composés des actions interactives de base et un niveau pour les tâches composées des énoncés. Les deux niveaux ont été choisis par rapport à l'espace de conception présenté dans la section 1.3.2 qui distingue le parallélisme entre les actions interactives de base au niveau énoncé et le parallélisme entre les tâches plus complexes.

3.2.3.1 Eléments syntaxiques

Nous conservons les mêmes opérateurs définis dans le modèle générique Mg présenté précédemment ainsi que leur sémantique. Les tâches et les énoncés sont paramétrés par les modalités utilisées pour leur exécution. L'ensemble des opérateurs est restreint au niveau des tâches et au niveau des énoncés selon l'espace de conception choisi. On note par :

- A_{mi}, l'ensemble des actions ai produites par la modalité mi. Ainsi, si nous disposons de n modalités, alors
- $A = \bigcup_{i=1}^{n} A_{mi}$ et on note "a" les éléments de A.
- AM désigne l'ensemble des actions générées par un sous-ensemble de modalités du système. Si le système dispose de n modalités, alors
  $AM = \bigcup_{k \in 1..n} A_{mk}$ avec $A_{mk} \subseteq A$.

A partir des définitions précédentes et du modèle sémantique précédent, nous pouvons décliner différents types d'interactions multimodales. Chaque type correspond à un sous-ensemble du modèle général Mg présenté dans la section précédente. La sémantique reste toujours exprimée par le même type de système de transitions. Nous donnons ci-dessous les différents modèles correspondant aux classes d'IHM définies dans le chapitre 1.

3.2.3.2 Type exclusif

Dans ce type d'interfaces, les énoncés sont produits de manière séquentielle. Ceci implique que l'opérateur de parallélisme ne doit pas être utilisé pour composer les énoncés. Seuls les opérateurs du choix ([]) et la séquence (>>) sont présents au niveau de la première règle qui définit l'IHM en fonction des énoncés et des tâches.

Un énoncé est composé des actions issues d'une seule modalité. Si les actions de base qui participent dans la formation d'un énoncé sont issues de la modalité m_i alors il est noté $Enonce_{Am_i}$. Dans la règle qui le génère, seules les actions de l'ensemble A_{mi} sont acceptées. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MEx suivante :

$$\begin{array}{lll} IHM & ::= & IHM[]IHM \mid IHM >> IHM \mid Enonce_{Am_i} \\ Enonce_{Am_i} & ::= & ai; Enonce_{Am_i} \mid \delta \; avec \; ai \in Am_i \end{array}$$

3.2.3.3 Type alterné

Comme pour le type exclusif, les énoncés dans une interface de type alterné, sont composés de manière séquentielle. Par contre, un énoncé peut être produit par plusieurs modalités. Ainsi, les énoncés dans la première règle sont indexés par l'ensemble AM qui est un ensemble d'actions produites par un sous ensemble de modalités du système. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MAlt suivante :

$$\begin{array}{lll} IHM & ::= & IHM[]IHM \mid IHM >> IHM \mid Enonce_{AM} \\ Enonce_{AM} & ::= & a; Enonce_{AM} \mid a \;|||\; EnonceAM \mid \delta \; avec \; a \in AM \end{array}$$

3.2.3.4 Type synergique

Dans les IHM de type synergique, les énoncés sont composés de manière séquentielle. De ce fait, dans la première règle nous ne gardons que les opérateurs de séquence et de choix. Les énoncés peuvent être produits par plusieurs modalités, ce qui explique que les actions qui participent à leur réalisation sont prises de l'ensemble AM. Les actions interactives de base peuvent être produites en parallèle ou entrelacées, ce qui explique l'utilisation des opérateurs du parallèlisme et le parallèle entrelacé dans la règle qui génère les énoncés. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MSyn suivante :

$$\begin{array}{lll} IHM & ::= & IHM[]IHM \mid IHM >> IHM \mid Enonce_{AM} \\ Enonce_{AM} & ::= & a; Enonce_{AM} \mid a \;|||\; Enonce_{AM} \mid a \;||\; Enonce_{AM} \mid \\ & & \delta \; avec \; a \in AM \end{array}$$

3.2.3.5 Type parallèle exclusive

Contrairement aux trois types d'IHM précédents, dans ce type d'interface les énoncés peuvent être produits de manière parallèle ou séquentielle. C'est ce qui

explique l'utilisation de l'opérateur parallèle entrelacé dans la première règle de la grammaire. Chaque énoncé est composé d'actions issues d'une seule modalité et c'est ce qui fait que chaque énoncé dans la grammaire est indexé par l'ensemble A_{mi} constitué des actions interactives produites par la modalité mi. Dans ce type d'IHM, il n'est pas toléré l'utilisation de deux actions en même temps et c'est ce qui explique l'absence de l'opérateur parallèle dans la première règle de la grammaire. Ainsi, la syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MPex suivante :

IHM	$::=$	$IHM[]IHM \mid IHM >> IHM \mid IHM \mid\mid\mid IHM \mid Enonce_{Am_i}$
$Enonce_{Am_i}$	$::=$	$a_i; Enonce_{Am_i} \mid \delta \ avec \ a_i \in Am_i$

3.2.3.6 Type parallèle simultané

Ce type d'IHM a les mêmes contraintes sur les énoncés que le type précédent, mais plusieurs modalités peuvent être utilisées en même temps. Ainsi, l'opérateur parallèle est utilisé dans la première règle. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MPsim suivante :

IHM	$::=$	$IHM[]IHM \mid IHM >> IHM \mid IHM \mid\mid IHM \mid$ $IHM \mid\mid\mid IHM \mid Enonce_{Am_i}$
$Enonce_{Am_i}$	$::=$	$a_i; Enonce_{Am_i} \mid \delta \ avec \ a_i \in Am_i$

3.2.3.7 Type parallèle alterné

Dans ce type d'IHM, un énoncé peut être produit par plusieurs modalités en parallèle entrelacé. Ceci explique la présence de l'opérateur parallèle entrelacé dans la deuxième règle qui génère les énoncés. Les énoncés sont indexés par l'ensemble des actions interactives AM qui les produisent et qui sont produites par un sous ensemble de modalités. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MPalt suivante :

IHM	$::=$	$IHM[]IHM \mid IHM >> IHM \mid IHM \mid\mid\mid IHM \mid Enonce_{AM}$
$Enonce_{AM}$	$::=$	$a; Enonce_{AM} \mid a \mid\mid\mid Enonce_{AM} \mid \delta \ avec \ a \in AM$

3.2.3.8 Type parallèle synergique

Ce type d'IHM est celui qui regroupe toutes les combinaisons possibles et rejoint ainsi le modèle générique Mg présenté dans la section précédente.

Les énoncés sont composés de manière parallèle ou séquentielle. C'est ce qui explique la présence de tous les opérateurs dans la première règle qui compose les énoncés. Chacun des énoncés peut être produit par plusieurs modalités et c'est ce qui explique l'indexation des énoncés par l'ensemble AM. Plusieurs actions peuvent être déclenchées en même temps et c'est ce qui explique la présence de l'opérateur parallèle dans les deux règles. La syntaxe du modèle correspondant à ce type d'interface est donnée par la grammaire MPsyn suivante :

IHM	::=	$IHM[]IHM \mid IHM >> IHM \mid IHM$ ||IHM | IHM|||IHM | $Enonce_{AM}$
$Enonce_{AM}$	::=	$a; Enonce_{AM} \mid a \mid\mid\mid Enonce_{AM} \mid a \mid\mid Enonce_{AM} \mid \delta$ *avec* $a \in AM$

La sémantique des différents opérateurs est identique à celle définie pour le modèle générique Mg. Ces différents modèles peuvent être implémentés dans n'importe quelle technique formelle. Afin d'illustrer notre démarche de conception nous proposons la modélisation d'une partie d'une application multimodale comme étude de cas.

3.3 Application à l'étude de cas

Pour appliquer notre modèle, nous avons choisi l'application MATIS [Nigay94] comme étude de cas. Nous modélisons une IHM qui permet à l'utilisateur de formuler une seule requête. Nous considérons deux modalités : *parole* et *manipulation directe* avec la souris. Avant de présenter le modèle correspondant aux types d'interfaces choisis, nous définissons l'ensemble des actions interactives de base générées par chacune des modalités du système. A_{parole} et $A_{manipulationDirecte}$ sont les ensembles des actions du système générées, repectivement par les modalités *parole* et *manipulationDirecte* en utilisant la souris.

A_{parole} = {'show me flights', 'from', 'this', 'City','To', 'Boston', 'Oslo'}
$A_{manipulationDirecte}$ = {ClicBoston, ClicOslo, ClicFrom, ClicTo, ClicNrequête}

Avec les actions de base suivantes réalisées par la souris :
- ClicBoston, ClicOslo : clic de la souris, respectivement, sur la ville Boston et sur la ville Oslo dans une liste de villes.
- ClicFrom, ClicTo : clic de la souris sur le champ texte respectivement, From et To du formulaire de saisie d'une requête ;
- ClicNrequête : clic de la souris sur le bouton attribué à la création d'une nouvelle requête. Le clic sur ce bouton engendre la création d'une nouvelle fenêtre de formulaire de saisie d'une nouvelle requête.

Nous présentons le modèle de l'IHM selon le type de multimodalité choisi. Nous donnons la syntaxe et le système de transitions correspondant.

3.3.1 IHM de type exclusif

Une expression d'une IHM de type exclusive peut être décomposée en fonction des tâches comme suit :

$$(CrerReq) >> (RemplirFrom) >> (RemplirTo)$$

Telle que *CreerReq* est la tâche qui permet de créer une nouvelle fenêtre pour formuler une requête, *RemplirFrom* est la tâche qui permet de remplir le champ *From* correspondant à la ville de départ dans le formulaire de la requête et *RemplirTo* est la tâche correspondant à la tâche qui permet de remplir le champ *To* correspondant à la ville de destination.

L'expression de cette IHM en fonction des énoncés est donnée par :

$$\begin{array}{l}(Crer1req[]Crer2req) >> \\ (Remplir1From[]Remplir2From) >> \\ (Remplir1To[]Remplir2To))\end{array}$$

Les énoncés *Creer1Req*, *Remplir1From* et *Remplir1To* permettent de réaliser respectivement les tâches *CreerReq*, *RemplirFrom* et *RemplirTo* en utilisant la parole. Les énoncés *Creer2Req*, *Remplir2From* et *Remplir2To* permettent de réaliser respectivement les tâches *CreerReq*, *RemplirFrom* et *RemplirTo* en utilisant la manipulation directe.

Chacun des énoncés est défini en fonction des actions interactives de base comme suit :

$$\begin{array}{lcl}Creer1Req & = & 'showmeflights'; \delta \\ Creer2Req & = & clicNrequete; \delta \\ Remplir1From & = & 'From'; 'Boston'; \delta \\ Remplir2From & = & clicFrom; clicBoston; \delta \\ RemplirTo & = & 'To'; 'Oslo'; \delta \\ Remplir2To & = & clicTo; clicOslo; \delta\end{array}$$

Enfin, l'expression de l'IHM en fonction des actions de base est obtenue en remplaçant chacun des énoncés par son expression en fonction des actions de base :

$$
\begin{array}{|l}
(('showmeflights';\delta)[](ClicNrequete;\delta)) >> \\
(('From';' Boston';\delta)[](ClicFrom;ClicBoston;\delta)) >> \\
(('To';' Oslo';\delta)[](ClicTo;ClicOslo;\delta))
\end{array}
$$

L'arbre correspondant à cette modélisation est donné par la figure 3.11 suivante :

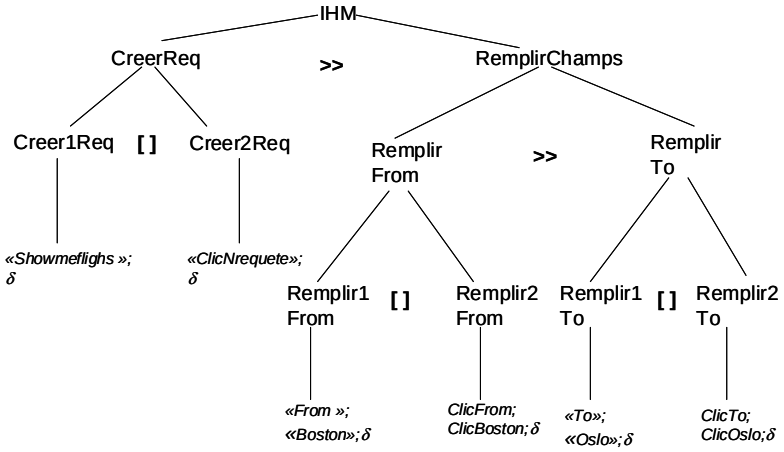

Fig. 3.11 – Arbre de décomposition de l'interaction

Le système de transitions est obtenu par composition des différents systèmes de transitions des différents énoncés et tâches.

Créer1req et *Créer2req* sont deux énoncés qui permettent chacun de créer une fenêtre dans laquelle l'utilisateur saisit les différents paramètres de la requête.

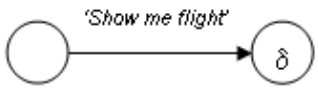

Fig. 3.12 – Système de transitions de Creer1req

Les expressions syntaxiques de $Creer1req$ et $Creer2req$ sont respectivement $('showmeflights';\delta)$ et $(clicNrequete;\delta)$. Leurs systèmes de transitions respectifs sont données par les figures 3.12 et 3.13.

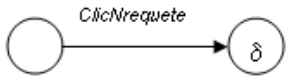

Fig. 3.13 – Système de transitions de Creer2req

*Remplir*1*From* et *Remplir*2*From* sont deux énoncés qui permettent chacun de remplir le champ From dans la fenêtre du formulaire par le mot 'Boston'. Leurs expressions syntaxiques sont respectivement $('From';' Boston'; \delta)$ et $(ClicFrom; ClicBoston; \delta)$.

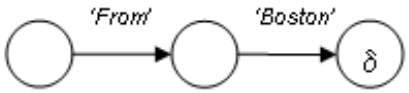

Fig. 3.14 – Système de transitions de Remplir1From

Leurs systèmes de transitions sont donnés respectivement par les figures 3.14 et 3.15.

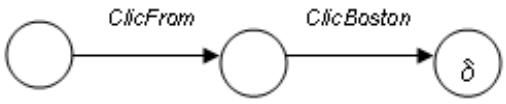

Fig. 3.15 – Système de transitions de Remplir2From

*Remplir*1*To* et *Remplir*2*To* sont deux énoncés qui permettent chacun de remplir le champ To dans la fenêtre du formulaire par le mot $'Oslo'$. $('To';' Oslo'; \delta)$ et $(ClicTo; ClicOslo; \delta)$ sont les expressions syntaxiques correspondantes respectivement.

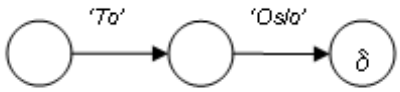

Fig. 3.16 – Système de transitions de Remplir1To

Les figures 3.16 et 3.17 décrivent les systèmes de transitions correspondants.

Le système de transitions de l'IHM est obtenu par la composition des systèmes de transitions des différents énoncés selon les règles sémantiques présentées plus haut.

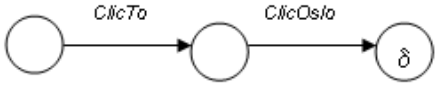

Fig. 3.17 – Système de transitions de Remplir2To

Le système de transitions de la tâche $Creer$ dont l'expression syntaxique est $(Crer1req[]Crer2req)$, est obtenu par composition des deux systèmes de $Creer1req$ et $Creer2req$. Il est représenté par la figure 3.18.

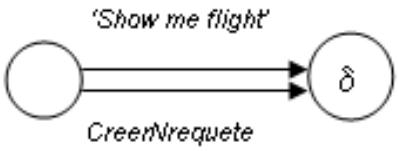

Fig. 3.18 – Système de transitions de Creer

Le système de transitions de la tâche $RemplirFrom$ dont l'expression syntaxique est $(Remplir1From[]Remplir2From)$ est obtenu par composition des deux systèmes de *Remplir1From* et *Remplir2From*. Il est représenté par la figure 3.19.

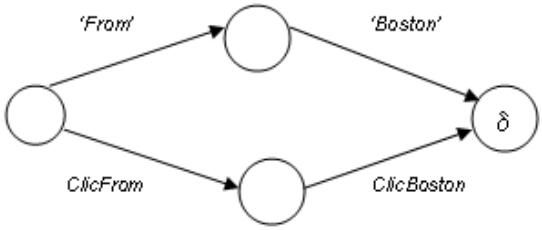

Fig. 3.19 – Système de transitions de RemplirFrom

Le système de transitions de la tâche $RemplirTo$ dont l'expression syntaxique est $(Remplir1To[]Remplir2To)$ est obtenu par composition des deux systèmes de $Remplir1To$ et de $Remplir2To$. Il est représenté par la figure 3.20.

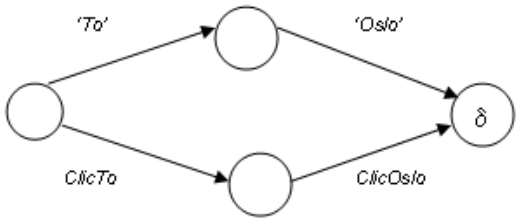

Fig. 3.20 – Système de transitions de RemplirTo

Enfin le système de transitions de l'IHM dont l'expression syntaxique en fonction des énoncés est $(Crer1req[]Crer2req) >> (Remplir1From[]Remplir2From) >> (Remplir1To[]Remplir2To)$ est donné par la figure 3.21.

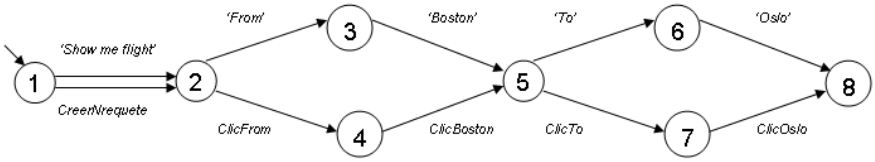

Fig. 3.21 – Système de transitions de l'IHM exclusive

3.3.2 Type alterné

Nous reprenons le même exemple traité dans la section précédente, mais nous reformulons l'expression des énoncés de façon à respecter la grammaire du modèle de type alterné. Nous définissons les énoncés en fonction des actions interactives de base comme suit :

$$
\begin{array}{|lcl|}
\hline
Creer1Req & = & 'showmeflights'; \delta \\
Creer2Req & = & clicNrequte; \delta \\
Remplir1From & = & ClicFrom; 'Boston'; \delta \\
Remplir2From & = & 'From'; clicBoston; \delta \\
RemplirTo & = & clicTo; 'Oslo'; \delta \\
Remplir2To & = & 'To'; clicOslo; \delta \\
\hline
\end{array}
$$

Enfin, l'expression de l'IHM en fonction des actions de base est obtenue en remplaçant chacun des énoncés par son expression en fonction des actions de base :

$$
\begin{array}{|l|}
\hline
(('showmeflights'; \delta)[](ClicNrequte; \delta)) \\
\gg ((\text{ClicFrom}\,;\ \text{'Boston '};\delta)[]('From'; ClicBoston; \delta)) \\
\gg ((\text{ClicTo}\,;\ \text{'Oslo '};\ \delta)[]('To'; ClicOslo; \delta)) \\
\hline
\end{array}
$$

Le système de transitions correspondant est donné par la figure 3.22.

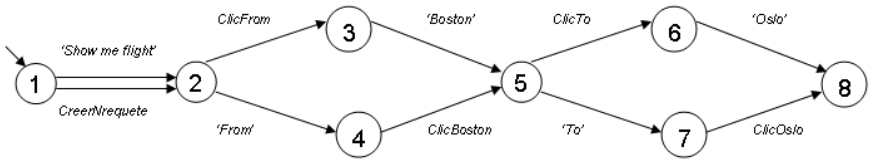

Fig. 3.22 – Système de transitions de l'IHM alternée

3.3.3 Type parallèle exclusif

Pour ce type d'IHM, nous reprenons les énoncés du modèle exclusif et nous composons les tâches avec l'opérateur du parallèle entrelacé (interleaving). Chacun des énoncés est défini en fonction des actions interactives de base comme pour le cas de l'IHM de type exclusif. L'expression de l'IHM en fonction des actions de base est obtenue en combinant ces énoncés avec les opérateurs de séquence, choix et interleaving. Elle est donné comme suit :

$$
\begin{array}{|l}
((\prime showmeflights\prime;\delta)[](ClicNrequte;\delta)) \\
\gg (((\text{'From'}\,;\,\text{'Boston '};\delta)|||(ClicTo;ClicOslo;\delta))[] \\
((\text{ClicFrom};\text{ClicBoston};\delta)|||(\prime To\prime;\prime Oslo\prime;\delta)))
\end{array}
$$

Le système de transitions correspondant est donné par la figure 3.23.

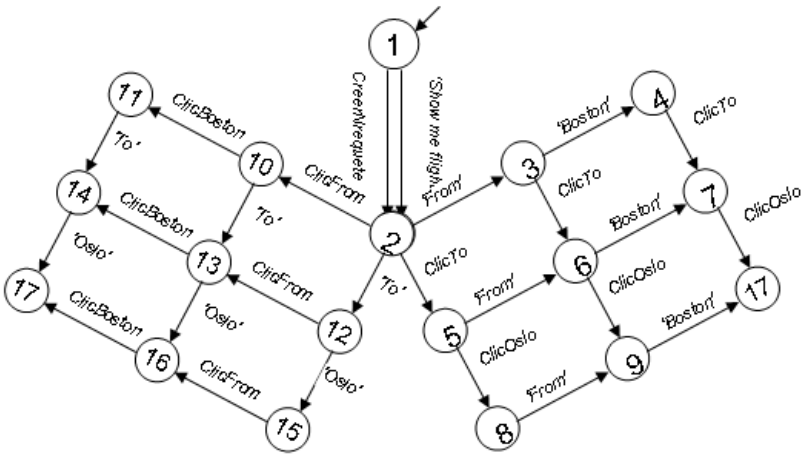

Fig. 3.23 – Système de transitions de l'IHM parallèle exclusive

3.4 Modèle formel pour les propriétés CARE

La vérification des propriétés d'utilisabilité CARE nécessite leur expression d'une manière formelle. La définition formelle donnée dans [Coutaz *et al.*, 1995a] se limite au niveau syntaxique, ce qui laisse beaucoup d'ambiguïtés quant à la sémantique du modèle sur lequel sera interprétée cette syntaxe. Dans cette section, nous proposons de définir un modèle sémantique pour ces propriétés. En se referant aux deux approches de model-checking présentées dans la section 2.3.1, nous définissons deux modélisations formelles de propriétés CARE. La première est basée sur les algèbres de processus comme pour le modèle de l'interaction présenté dans les sections précédentes. Il est défini par une syntaxe et une sémantique basée sur les systèmes de transitions décrivant le comportement et l'enchaînement dans le temps des différentes utilisations des différentes modalités. La seconde modélisation est basée sur les logiques temporelles. Dans les deux modèles formels que nous proposons, les propriétés CARE sont définies par rapport aux actions interactives de base.

3.4.1 Modèle opérationnel

Le modèle opérationnel permet de décrire la propriété avec un système de transitions étiquetées. Nous donnons un modèle sémantique pour chacune des propriété-

tés CARE en utilisant un ensemble d'opérateurs de composition similaires à ceux définis pour le modèle de l'interaction multimodale. Les opérateurs définissent les contraintes temporelles et l'enchaînement des utilisations des modalités dans le temps.

Chacun des modèles est présenté par sa syntaxe et la sémantique opérationnelle de ses différents opérateurs. Les modèles que nous définissons expriment les propriétés pour une tâche d'interaction donnée. Nous précisons que dans les modèles, nous ne nous intéressons pas aux actions elles même mais plutôt à leurs modalités. Pour cela une abstraction des actions interactives est effectuée en remplaçant chaque action par sa modalité.

3.4.1.1 Syntaxe

Nous donnons la syntaxe des différents modèles sous forme de grammaires avec un ensemble de règles. La règle correspondant à S décrit la tâche principale en fonction des sous tâches en les composant avec des opérateurs de composition déjà présentés dans le modèle formel d'interaction de l'IHM3. Pour chaque sous-tâche nous définissons une règle qui décrit sa construction. Nous considérons que le système dispose de n modalités notées m_i (avec $1 \leq i \leq n$).

Nous notons l'ensemble des n modalités du système par M. Nous décrivons les modèles des propriétés CARE pour deux modalités m_i et m_j de l'ensemble M (avec $1 \leq i, j \leq n$).

Equivalence. L'équivalence exprime le choix entre deux modalités pour réaliser la tâche. Ceci est exprimé par le choix entre deux tâches réalisées chacune par une modalité différente. La tâche T_{mi} est une tâche réalisée par la modalité m_i et la tâche T_{mj} est réalisée par la modalité m_j.

$$\begin{array}{lcl} S & ::= & Tm_i[]Tm_j \\ Tm_i & ::= & m_i; Tm_i \mid \delta \\ Tm_j & ::= & m_j; Tm_j \mid \delta \end{array}$$

Complémentarité. Dans le modèle exprimant la complémentarité, il faut assurer que dans chaque chemin d'exécution, les modalités m_i et m_j apparaissent chacune au moins une fois.

La tâche T_{mji} est une tâche réalisée avec des actions de modalités m_i ou m_j, mais dont au moins une est de modalité m_j.

T_m* est une tâche réalisée avec des actions de modalités m_i et m_j, ou bien uniquement avec m_i ou m_j.

Si la première action est réalisée avec la modalité m_i alors la suite (T_{mji}) est réalisée avec des actions de modalités m_i ou m_j mais au moins une action de

modalité m_j doit être exécutée $(m_j; T_{m*})$ avant de passer ensuite au libre choix entre des actions de modalités m_i ou m_j et enfin l'arrêt(δ).

$$\begin{array}{lll} S_0 & ::= & S_1 \mid S_2 \mid S_0 [] S_0 \\ S_1 & ::= & m_i; Tm_{j_i} \\ S_2 & ::= & m_j; Tm_{i_j} \\ Tm_{j_i} & ::= & m_i; Tm_{j_i} \mid m_j; Tm_{j_i} \mid m_j; Tm* \\ Tm_{i_j} & ::= & m_j; Tm_{i_j} \mid m_i; Tm_{i_j} \mid m_i; Tm* \\ Tm* & ::= & m_j; Tm* \mid mi; Tm* \mid \delta \end{array}$$

Assignation. Dans ce cas, seule la modalité assignée (m_i) est utilisée pour réaliser la tâche. Plusieurs manières sont possibles pour effectuer cette tâche (T_{mj} ou bien T_{mk}) mais elles sont toutes réalisées avec la même modalité m_i.

$$\begin{array}{lll} S & ::= & Tmi \mid Tmj [] Tmk \\ Tmi & ::= & mi; Tmj \mid \delta \\ Tmk & ::= & mi; Tmk \mid \delta \\ Tmj & ::= & mi; Tmj \mid \delta \end{array}$$

Redondance. La redondance est exprimée par la mise en parallèle de deux tâches équivalentes réalisées chacune par une modalité. L'équivalence est exprimée par le choix de réaliser cette même tâche par une modalité ou par l'autre.

$$\begin{array}{lll} S & ::= & Tmi [] S_1 \\ S_1 & ::= & Tmj [] S_2 \\ S_2 & ::= & Tmi \mid\mid Tmj \\ Tmi & ::= & mi; Tmi \mid \delta \\ Tmj & ::= & mj; Tmj \mid \delta \end{array}$$

3.4.1.2 Sémantique

La sémantique opérationnelle des différents opérateurs est la même que celle donnée dans le modèle formel de l'interaction. A chaque terme correspond un modèle sous forme de système de transitions exprimant le comportement dynamique de chacune des propriétés. L'état initial du système de transitions correspond à l'état initial de la tâche et l'état final correspond à l'état final de la tâche.

Le système de transitions utilisé pour modéliser ces propriétés suppose que les transitions sont instantanées et donc pas de quantification de temps. Seul l'ordonnancement permet d'exprimer la chronologie du temps. Dans ce cas, la notion d'intervalle de temps *tw* utilisée dans le modèle donné dans [Coutaz *et al.*, 1995b] est remplacée par un intervalle d'états. Ce dernier est défini par l'état initial et l'état

final d'un ensemble de chemins dans le système de transitions. Plusieurs chemins peuvent exister dans un intervalle d'états.

3.4.1.3 Exemples

Nous supposons deux modalités : *parole* et *manipulation directe* avec la souris. Nous donnons pour chaque modèle correspondant à une propriété CARE, l'expression syntaxique et le système de transitions correspondant.

L'expression $Direct; Direct; Parole; Direct; Parole; \delta$ correspond à une tâche effectuée par deux actions successives réalisées avec la manipulation directe suivie d'une interaction réalisée avec la parole, ensuite une interaction réalisée avec la manipulation directe et enfin une interaction avec la parole.
C'est une tâche ou les modalités parole et manipulation directe sont utilisées de manière complementaires.
Une expression concrétisant cette propriété peut être la suivante :

$$\boxed{CreerNRequete; ClicForm;'Boston', ClicTo;'Oslo'; \delta}$$

L'expression $(Direct; Direct; Direct; \delta)[](Direct; \delta)$ correspond à une tâche effectuée exclusivement avec la *manipulation directe* avec la souris. La tâche est réalisée soit avec trois interactions successives, soit avec une seule interaction avec la *manipulation directe.*

C'est une tâche où la modalité manipulation directe est assignée.

L'expression $(parole; parole; parole; \delta) \;||\; (direct; direct; direct; \delta)$ correspond à une tâche effectuée avec les deux modalités parole et manipulation directe d'une manière parallèle.

3.4.2 Modèle Logique

Dans cette section nous définissons une logique modale que nous notons LCARE pour l'expression des propriétés CARE.

3.4.2.1 Syntaxe

Soit M l'ensemble des modalités et $mi \in M$.

$$\boxed{\phi ::= Eq(M) \mid Assig(mi) \mid Red(M) \mid Comp(M) \mid \phi \; et \; \phi \mid \phi \; ou \; \phi \mid non \; \phi}$$

3.4.2.2 Sémantique

Les formules de LCARE sont interprétées sur un système de transitions défini par $(Q, A, \rightarrow, q_0)$, où

Les chemins d'exécution sont définis par une suite d'états ainsi que par les actions permettant de transiter entre ces états. Ils sont de la forme $q_0 \stackrel{a_0}{\rightarrow} q_1 \stackrel{a_1}{\rightarrow} .... \stackrel{a_{i-1}}{\rightarrow} q_i ... q_n$ où q_i représente un état et a_i l'action qui fait transiter le système de l'état q_i vers l'état q_{i+1}.

Nous définissons l'ensemble des modalités offertes par un ensemble M. La modalité i appartenant à cet ensemble est notée m_i.

Pour chaque modalité m_i, un ensemble d'événements A_{mi} est défini. Les éléments de cet ensemble sont notés a_i. Nous définissons une fonction Mod qui attribue à chaque événement la modalité avec laquelle il est produit. Si l'événement a est produit par la modalité m_i alors Mod(a)=m_i. q et q' sont deux états du système de transitions.
La relation de satisfiabilité est donnée par :

1. $(q, q') \mid= Equi(M)$ *ssi*
   $\forall mi \in M, \exists q(= q_0) \stackrel{a0}{\rightarrow} q_1 \stackrel{a1}{\rightarrow} ... \stackrel{an}{\rightarrow} ...q'$ *telque* $\forall a_j 1 \leq j \leq n, Mod(a_j) = m_i$.

   Informellement, elle signifie que les modalités de l'ensemble M sont équivalentes pour atteindre l'état q' à partir de l'état q si et seulement si, pour chaque modalité mi de l'ensemble M, il existe un chemin d'exécution allant de l'état q jusqu'à l'état q', tel que toutes les actions effectuées sur ce chemin sont produites par la modalité mi. Pour noter les états du chemin d'exécution sous la forme $q_0,\ q_1,\ q_2, \cdots$, nous renommons l'état q par q_0.

2. $(q, q') \mid= Assig(m_i)$ *ssi*
   $\forall q(= q_0) \stackrel{a0}{\rightarrow} q1 \stackrel{a1}{\rightarrow} ... \stackrel{an}{\rightarrow} q', \forall j, 0 \leq j \leq n, Mod(a_j) = m_i$

   Elle signifie que la modalité mi est assignée pour atteindre l'état q' à partir de l'état q, si et seulement si tous les chemins d'exécution allant de q vers q' sont effectués par des actions produites par la modalité mi.

3. $(q, q') \mid= Comp(M)$ *ssi*
   $(\forall q(= q_0) \stackrel{a0}{\rightarrow} q1 \stackrel{a1}{\rightarrow} ... \stackrel{an}{\rightarrow} q', \forall j, 0 \leq j \leq n, Mod(aj) \in M$ *et* $\forall mi \in M, \exists j, 0 \leq j \leq n, Mod(aj) = mi$ *et* $non(\exists i, mi \in M$ *et* $\forall j, 0 \leq j \leq n, Mod(aj) = mi))$

   Informellement, elle signifie que les modalités de l'ensemble M sont complémentaires pour atteindre l'état q' à partir de l'état q, si et seulement si sur chacun des chemins d'exécution allant de q vers q'

(a) toutes les actions effectuées sur ce chemin sont produites par des modalités de l'ensemble M ;

(b) chaque modalité de l'ensemble M est utilisée dans ce chemin d'exécution ;

(c) il n'existe pas une modalité de l'ensemble M qui permet à elle seule d'atteindre l'état q' à partir de l'état q.

4. $(q, q') \mid= Red(M)$ *ssi*
$\forall mi \in M, \exists q(= q0) \overset{a0}{\rightarrow} q1 \overset{a1}{\rightarrow} ... \overset{an}{\rightarrow} q'/\forall j, 0 \leq j \leq n, Mod(aj) = mi$ *et* $\exists \sigma = \sigma_1 \otimes \sigma_2 \otimes ... \otimes \sigma_n$ *avec* $\sigma_i = q(= q0) \overset{a0}{\rightarrow} q1 \overset{a1}{\rightarrow} ... \overset{an}{\rightarrow} q'/\forall j, 0 \leq j \leq n, Mod(a_j) = m_i$
$\otimes$ est l'opérateur du produit cartésien entre les chemins d'exécution. Il est obtenu en appliquant l'opérateur de produit cartésien présenté dans la section 2.4.3.1, tel que chaque système de transitions Ai est remplacé par un chemin d'exécution σ_i.

Elle signifie informellement que les modalités de l'ensemble M sont utilisées de manière redondante pour atteindre l'état q' à partir de l'état q, si et seulement si

(a) chaque modalité mi de l'ensemble M permet d'atteindre l'état q' à partir de l'état q. Ceci exprime l'équivalence des modalités de l'ensemble M.

(b) il existe un ensemble de chemins d'exécution résultant du produit de tous les chemins d'exécution réalisés chacun par une modalité de l'ensemble M.

Nous exprimons ces formules dans les logiques temporelles CTL et LTL dans les chapitres suivants.

Exemple : Soient
- $M = \{parole, manipulationdirecte\}$, l'ensemble des modalités du système ;
- $A = \{a1, a2, a3, a4, a5, a6\}$, l'ensemble des actions interactives du système, avec les actions $a1$, $a2$ et $a4$ générées par la modalité *manipulation directe* et les actions $a3$, $a5$ et $a6$ générées par la modalité *parole*.

Le système de transitions correspondant est donné sur la figure 3.24.

Le couple d'états $(q = 2, q'5)$ satisfait la formule logique $Equi(M)$. En effet pour chacune des modalites *parole* et *manipulation directe*, il existe un chemin d'exécution allant de l'état 2 à l'état 5 réalisé par la modalité concernée.

Pour la modalité *manipulation directe*, il existe le chemin d'exécution $2 \overset{a2}{\rightarrow} 3 \overset{a4}{\rightarrow} 5$ tel que $Mod(a2) =$ *manipulation directe* et $Mod(a4) =$ *manipulation directe*.

Pour la modalité *parole*, il existe le chemin d'exécution $2 \overset{a3}{\rightarrow} 3 \overset{a5}{\rightarrow} 5$ tel que $Mod(a3) =$ *parole* et $Mod(a5) =$ *parole*.

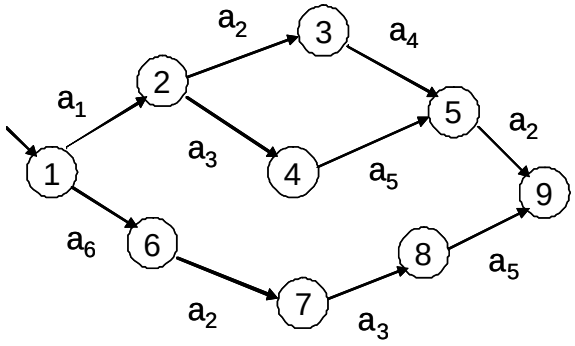

Fig. 3.24 – Système de transitions

Le couple $(q = 7, q' = 9)$ satisfait la formule $Assig(parole)$. En effet, le seul chemin qui existe entre l'état 7 et l'état 9 ($7 \xrightarrow{a3} 8 \xrightarrow{a5} 9$) est réalisé par des actions ($a3$ et $a5$) produites par la modalité *parole*.

3.5 Conclusion

Nous avons présenté dans ce chapitre :
- un modèle formel pour l'interaction selon le type d'IHM multimodale
- deux modèles formels pour l'expression des propriétés d'utilisabilite CARE.

Les modèles sont génériques et indépendants de toute technique formelle. Au niveau modélisation de l'interaction de l'IHM multimodale, nous avons proposé un modèle générique et paramétré permettant de modéliser l'interaction multimodale en entrée selon le type d'IHM souhaité. Ces types sont issus des espaces de conception définis par la communauté des chercheurs dans le domaine des IHM multimodales. Au niveau modélisation des propriétés d'utilisabilité des IHM multimodales, nous avons proposé deux modèles formels pour exprimer les propriétés CARE. Le premier modèle est opérationnel et décrit la propriété par un système de transitions. Le deuxième modèle est logique et exprime une propriété par une formule logique. Plusieurs approches de conception et vérifications peuvent être suivies lors du développement d'un système multimodal. Les modèles formels que nous avons proposés seront utilisés selon l'approche de développement choisie. Une traduction de ces modèles vers les modèles des techniques formelles est nécessaire afin de les mettre en oeuvre.

Les prochains chapitres montrent l'implantation des modèles proposés dans les deux techniques formelles : model-checking et preuve. Le chapitre 5 illustre l'utilisation de l'approche de conception par composition et la vérification des IHM multimodales par la technique du model-checking. Dans le chapitre 6, nous montrons l'utilisation de la technique de raffinement pour le développement et la preuve pour la vérification des propriétés.

Chapitre 4

Mise en oeuvre du modèle générique dans la technique de model-checking

4.1 Introduction

L'objectif de ce chapitre est la mise en oeuvre de notre modèle formel générique pour la modélisation des IHM multimodales ainsi que la vérification de leurs propriétés d'utilisabilité CARE dans les techniques de vérification sur modèle. Nous nous efforçons de montrer comment deux techniques de model-checking, l'une fondée sur une logique temporelle linéaire avec PROMELA/Spin et l'autre fondée sur une logique temporelle arborescente avec SMV, peuvent modéliser une IHM Multimodale à partir du modèle générique proposé. Ces deux techniques permettent d'implémenter les systèmes de transitions ainsi que la vérification de propriétés exprimées dans une logique temporelle. SMV permet d'exprimer les propriétés dans la logique temporelle arborescente CTL et Promela/Spin permet de les exprimer dans la logique temporelle linéaire LTL. Dans les deux outils, la conception est ascendante et donc utilise la composition.

Plusieurs outils, implémentant la technique de vérification sur modèle (model-checking), existent. Ils permettent d'analyser les modèles des systèmes à états finis et de vérifier des propriétés temporelles d'une manière automatique. Certains construisent explicitement l'espace d'états, et d'autres travaillent sur une représentation symbolique de cet espace d'états afin d'optimiser l'espace mémoire et le temps de calcul. Nous avons choisi deux outils mettant en oeuvre cette technique : SMV et SPIN.

- ***SMV :*** est un outil générique qui permet la vérification de propriétés temporelles. Il utilise la logique temporelle CTL pour spécifier les propriétés du système décrit par un système de transitions, les diagrammes de décision binaires (BDD) pour manipuler l'espace d'états et un algorithme symbolique pour l'exploration de cet espace d'états. Ceci permet la vérification de systèmes plus

complexes en taille que ceux utilisés par les outils non-symboliques.

– ***PROMELA/SPIN :*** est un outil qui utilise le langage PROMELA pour décrire les systèmes de transitions et la logique temporelle LTL pour exprimer les propriétés du système. De plus, avec Spin, nous pouvons animer l'exécution du système, ce qui offre la possibilité de débuguer la spécification et de localiser les erreurs.

Nous présentons pour chacun des outils, son principe de fonctionnement et de modélisation. Ensuite, nous présentons comment nous représentons les différents éléments de notre modèle générique afin de le valider. Enfin, nous montrons comment notre étude de cas est traitée.

4.2 Mise en oeuvre avec SMV

SMV est un outil de model-checking symbolique. Il comporte un langage de description des automates dits aussi machines à états finis, Il permet de traiter des systèmes ne manipulant que des structures de données finies (booléen, type énuméré, intervalle fixé, tableau de dimension fixe) et de valider des formules CTL sur les exécutions. Le langage d'entrée de SMV permet de décrire le modèle du système et sa spécification. Le modèle utilisé par SMV est une structure de Kripke dont les états sont définis par un ensemble de variables d'états qui peuvent être de type booléen ou scalaire. La spécification est définie dans la classe SPEC par un ensemble de formules de logique temporelle CTL qui représentent les propriétés attendues du système. Enfin, un générateur de code est associé à SMV.

4.2.1 Principe de modélisation en SMV

Une spécification SMV est constituée d'un ensemble de modules. Un module en SMV a un état (un ensemble d'attributs dans la clause VAR), une relation de transition qui définit la transition entre les différents états (clauses Init et Next), et des axiomes décrivant l'évolution des états.

Les attributs sont déclarés dans la clause VAR, l'état initial du module est défini dans la clause $INIT$ et la relation de transition qui définit comment les états évoluent est définie dans la clause $ASSIGN$. Cette relation de transitions peut être définie par des axiomes dans la clause $TRANS$. Le tableau 4.1 montre la structure générale d'un module en SMV. Un module SMV est identifé par un *nom* défini dans la première ligne après le mot réservé **Module**. Dans la clause VAR, toutes les variables sont définies chacune avec son type. Dans la clause $ASSIGN$, les variables sont affectées par les valeurs qu'elles prennent dans les différents états du système. Dans la clause $SPEC$ toutes les formules logiques exprimées en CTL sont décrites afin d'être vérifiées par le système de transitions décrit par le module.

Module <nom>
VAR
$x1 : Type; \ldots ; xn : Type;$
ASSIGN
$Init(x1) := \ldots ;$
$\vdots$
$Init(xn) := \ldots ;$
$Next(x1) :=$ **case**
$\vdots$
esac ;
$Next(xn) :=$ **case**
$\vdots$
esac ;
SPEC
propriétés exprimées en CTL

Tab. 4.1 – Structure générale d'un module en SMV

Nous présentons dans ce qui suit comment sont codés les systèmes de transitions et les propriétés dans SMV.

4.2.1.1 Système de transitions

Le système de transitons est spécifié au moyen du langage d'entrée de SMV qui permet de décrire un système de transitions par la donnée de sa relation de transition.

- *Les variables* dans SMV peuvent être déclarées comme des booléens ou bien de type énuméré. Elles peuvent être déclarées également comme tableaux.
- *Les états* du système de transitions sont composés d'une valuation de toutes les variables du système spécifié.
- *L'état initial* du système est défini par les valeurs initiales de chaque variable dans le système. Il peut être défini de deux manières :
  - en utilisant la notation $init()$.
    $init(x)$, où x est une variable, définit l'état initial de la variable x. Par exemple, l'instruction $init(x) := 0$ affecte la valeur initiale 0 à la variable x,
  - en utilisant l'instruction d'affectation après le mot réservé $INIT$.
    Les instructions $x := 3$ et $y := 4$ introduites après le mot $INIT$ définissent l'état initial du système avec la variable x qui vaut 3 et la variable y qui vaut 4.
- *La relation de transition* définit l'état suivant du système. Elle est définie pour chaque variable et peut être définie de deux manières :

– en utilisant la notation $next()$ qui détermine l'état suivant d'une variable. $next(x) := 1$ affecte la valeur 1 dans l'état suivant de x.
– une expression booléenne introduite par le mot Clé $TRANS$ qui définit une relation de transitions entre les états. $next(x) = 0 \Rightarrow next(y) = 1$ signifie que quand la variable x passe à la valeur 0, y passe à la valeur 1.

Toutes ces déclarations, ainsi que les définitions introduites par **ASSIGN** peuvent être combinées. Elles s'ajoutent l'une après l'autre et restreignent l'ensemble des états initiaux et/ou des transitions possibles.

Chaque variable dans le système est définie par son automate ou système de transitions. Ce dernier est décrit à l'aide des deux notations $init()$ et $next()$.

Le système de transitions global est obtenu par le produit synchronisé des différents systèmes de transitions associés à chaque variable. Notons que l'intérêt des model-checker est leur capacité à ne pas construire explicitement cet automate. La relation de transition de la structure Kripke du système, et son état initial, sont déterminés par un ensemble d'affectations introduites par le mot réservé ASSIGN.

Le tableau 4.2 décrit le code correspondant au système de transitions de la figure 4.1

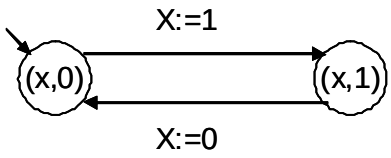

Fig. 4.1 – Un exemple de système de transitions

Module exemple
VAR
$x : \{0, 1\};$
ASSIGN
$Init(x) := 0;$
$Next(x) :=$ **case**
$0 : 1;$
$1 : 0;$
esac ;

Tab. 4.2 – Un module en SMV

C'est un système à une seule variable dont les valeurs possibles sont 0 ou 1 déclarée dans la clause *VAR*. Initialement la variable prend la valeur 0 qui constitue l'état initial du système. L'état suivant est défini pas la notation $next()$. Dans le cas où la valeur de x vaut 0, alors la valeur suivante prendra la valeur 1, et dans le

cas où elle vaut 1, alors elle prendra la valeur 0. Ceci est exprimé avec l'instruction *case . . . esac*.

4.2.1.2 Expression de propriétés en SMV

Plusieurs propriétés peuvent être spécifiées pour un même système. Les propriétés sont exprimées sous forme de formules de logique temporelle CTL et sont introduites dans la clause *SPEC*. SMV effectue la vérification et si l'une de ces propriétés est fausse, il s'arrête et donne une trace en guise de contre-exemple, qui peut être utilisée pour trouver l'erreur qui a causé l'échec.

4.2.2 Principes de représentation du modèle générique dans SMV

Pour pouvoir coder notre modèle dans SMV [Kamel, 2004], il faut d'abord donner les règles de passage de notre modèle vers le langage d'entrée de SMV. Cette section a pour objectif de montrer comment les systèmes de transitions de notre modèle formel, ainsi que les propriétés d'utilisabilité des IHM multimodales sont codés dans le langage de spécification de SMV.

4.2.2.1 Systèmes de transitions

Chaque variable d'état du modèle formel est représentée par une variable dans le modèle SMV. Pour simplifier la mise en oeuvre, nous regroupons l'ensemble des variables d'état dans une seule variable qui identifie l'état du système.

La structure utilisée par SMV pour la représentation du système est une structure de Kripke où une exécution est identifiée par une suite d'états identifiés par les valeurs des variables d'état. Les actions qui provoquent le changement d'états n'apparaissent pas dans les chemins d'exécution sur lesquels les formules de logique temporelle sont interprétées.

Puisque les propriétés que nous voulons exprimer et vérifier en SMV sont liées aux types d'actions qui provoquent les changements d'états, il est nécessaire de représenter ces actions dans les chemins d'exécution. Une solution consiste à coder ces actions dans les états du système. Pour cela, une variable *action* est ajoutée au système, dont le domaine de définition est l'ensemble des actions possibles du système. Nous avons choisi de coder les actions dans l'état source de la transition réalisée par l'action considérée. La figure 4.2 montre le codage des actions dans les états du système de transitions.

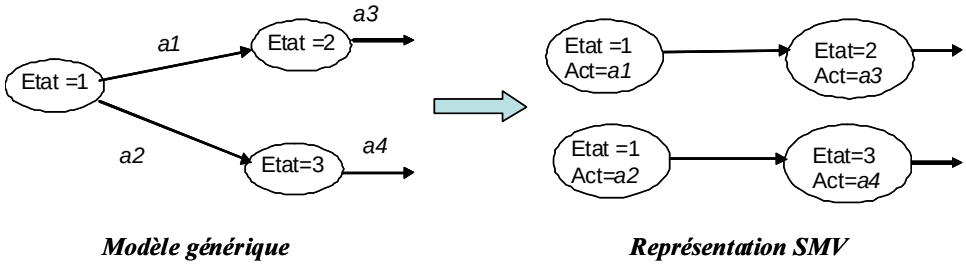

Fig. 4.2 – Transformation du système de transitions

4.2.2.2 Propriétés

Lors du raisonnement sur les propriétés liées uniquement aux modalités, une abstraction correcte du système est réalisée en remplaçant chaque action par son type, lui même représenté par la modalité qui a permis la réalisation de cette action. Ainsi, une variable correspondant aux différentes modalités vient remplacer la variable correspondant aux actions du système. Le domaine de cette variable est l'ensemble des modalités possibles dans le système. Cette abstraction permet de simplifier la vérification car l'ensemble des modalités est fini. Cette abstraction s'exprime de la façon suivante.

Le système de transitions de la figure 4.3 représente un exemple de système avec quatres états représentés par la variable *Etat*.

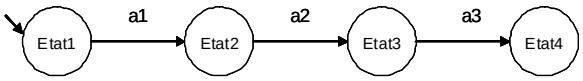

Fig. 4.3 – Système de transisions avec *Etat*

Le système de la figure 4.4 représente sa transformation en incluant les actions dans les états du système.

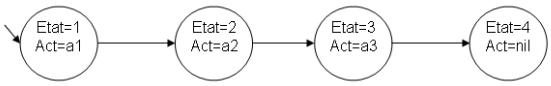

Fig. 4.4 – Système de transisions avec *Etat* et *Act*

Si les trois actions a_1, a_2 et a_3 sont produites respectivement par les modalités m_1,m_2 et m_3, alors l'abstraction du système de transitions de la figure 4.4 donne le système présenté sur la figure 4.5

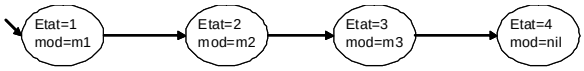

Fig. 4.5 – Système de transitions après abstraction

Ainsi, lors de la vérification du type d'actions qui a permis au système de transiter de l'état 1 à l'état 4, au lieu de comparer la variable *act* avec les actions $a1$, $a2$ et $a3$, la comparaison portera sur les modalités qui sont décrites de façon plus simple. Si de plus les actions avaient la même modalité alors cette abstraction fournirait un automate encore plus simple.

Notons que l'automate de la figure 4.5 peut être considéré comme l'abstraction de l'automate de la figure 4.6 si les actions $a4$ et $a5$ sont réalisées par les modalités m_2 et m_3.

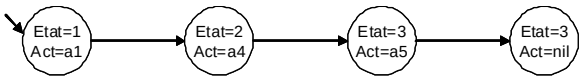

Fig. 4.6 – Système de transitons réalisé avec d'autres actions

Cela montre qu'un même automate abstrait permet de modéliser plusieurs systèmes concrets. Il faut noter que cette abstraction permet de modéliser les propriétés CARE.

4.2.2.3 Modélisation des propriétés CARE

Les propriétés que nous exprimons en CTL sont celles définissant les propriétés d'utilisabilité d'IHM multimodales et qui sont les propriétés CARE. Pour chacune des propriétés nous donnons le schéma général pour une tâche T représentée par un système de transitions caractérisé par l'état initial $EtatI$ et l'état final $EtatF$. $EtatI$ et $EtatF$ résument chacun un ensemble de valeurs de variables d'état caractérisant le système. La variable *etat* regroupe l'ensemble des variables du système et la variable *mod* représente les modalités utilisées dans le système.

Complémentarité. Deux modalités m_1 et m_2 sont dites complémentaires pour la réalisation de la tâche T si la formule donnée sur le tableau 4.3 est vérifiée par le système de transitions de l'IHM3.

Cette formule exprime en trois parties, que :

1. il existe au moins un chemin d'exécution dans le système de transitions tel que si un état est identifié comme l'état initial ($etat = etatI$) de la tâche d'interaction multimodale T, alors il existe (E) une exécution tel que tous les états suivants sont accessibles par les modalités m_1 ou m_2 jusqu'à ce que l'état final est atteint ($etat = etatF$) ;

$$
\left.\begin{array}{l} EG((etat = etatI) \\ \quad \Rightarrow E(((mod = m_1) or (mod = m_2)) \cup (etat = etatF))) \end{array}\right\} (1)
$$

and

$$
\left.\begin{array}{l} notEG((etat = etatI) \\ \quad \Rightarrow E((mod = m_1) \cup (etat = etatF))) \end{array}\right\} (2)
$$

and

$$
\left.\begin{array}{l} notEG((etat = etatI) \\ \quad \Rightarrow E((mod = m_2) \cup (etat = etatF))) \end{array}\right\} (3)
$$

Tab. 4.3 – Formule générique CTL de la propriété de complémentarité

2. il n'existe aucun chemin qui démarre de l'état $etatI$ et se termine à l'état $etatF$ tel que toutes les transitions sont effectuées seulement par la modalité m_1 ;
3. il n'existe aucun chemin qui démarre de l'état $etatI$ et se termine à l'état $etatF$ tel que toutes les transitions sont effectuées seulement la modalité m_2.

Assignation. Une modalité m_i est assignée à la tâche d'interaction multimodale T si la formule donnée sur le tableau 4.4 est vérifiée par le système de transitions de l'IHM.

$$
\begin{array}{l} AG((etat = etatI) \\ \quad \Rightarrow A((mod = m_i) \cup (etat = etatF))) \end{array}
$$

Tab. 4.4 – Formule générique CTL de la propriété d'assignation

Cette formule exprime que tous les chemins partant de l'état $etatI$ jusqu'à l'état $etatF$ sont tout le temps réalisés (A) avec des actions de modalité m_i.

Equivalence. Deux modalités m_1 et m_2 sont équivalentes pour la réalisation de la tâche d'interaction multimodale T si la formule donnée sur le tableau 4.5 est vérifiée par le système de transitions de l'IHM.

Cette formule exprime en deux parties, que sur tous les chemins d'exécution,

1. si un état est identifié comme l'état initial ($etat = etatI$) de la tâche d'interaction multimodale T, alors il existe (E) au moins un chemin tel que ses états sont atteints toujours (AG) par la modalité m_1 jusqu'a ce que l'état final ($etat = etatF$) soit atteint ;
2. elle exprime la même chose que dans 1) mais avec la modalité m_2.

$$
\begin{array}{|l|}
\hline
\left.\begin{array}{l}(AG((etat = etatI) \Rightarrow \\ \quad E((mod = m_1) \cup (etat = etatF))\\ )\end{array}\right\} (1) \\
\text{And} \\
\left.\begin{array}{l}(AG((etat = etatI) \Rightarrow \\ \quad E((mod = m_2) \cup (etat = etatF))\\ )\end{array}\right\} (2) \\
\hline
\end{array}
$$

Tab. 4.5 – Formule générique CTL de la propriété d'équivalence

Redondance. Pour exprimer la propriété de redondance dans la logique temporelle CTL, nous devons énumérer toutes les combinaisons possibles entre les modalités au niveau de chaque état. Nous n'avons pas pu représenter cette propriété dans une expression générique de la logique CTL. Elle nécessite une logique plus riche, permettant d'exprimer l'opérateur de parallèle.

Il est important de noter que nous avons donné un schéma générique des propriétés CARE exprimées en CTL. Leur utilisation sur des systèmes IHM3 est immédiate comme nous le verrons ci-dessous avec l'étude de cas.

4.2.3 Application à l'étude de cas

Dans cette section, nous présentons l'implémentation des deux IHM modélisées dans le chapitre précédent. Pour chacune, nous donnons le code SMV correspondant au système de transitions ainsi que les propriétés CARE vérifiées et celles qui ne sont pas vérifiées pour la tâche $RemplirFrom$.

Cas de l'IHM exclusive. La figure 4.7 rappelle le système de transitions correspondant à cette IHM. La tâche $RemplirFrom$ est identifiée par son état initial 2 et son état final 5. C'est cette tâche que nous avons choisie pour vérifier ses propriétés CARE.

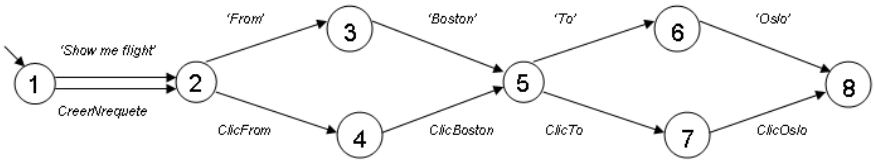

Fig. 4.7 – Système de transitions de l'IHM exclusive

Avant de donner le code SMV correspondant à ce système de transitions, nous montrons, sur la figure 4.8, le nouveau système de transitions obtenu après avoir

codé les modalités des actions dans les états du système. Nous désignons par P la modalité parole et par M la modalité manipulation directe. Chaque état du système de transitions contient deux valeurs : une valeur pour la variable état du système et une valeur pour la modalité des actions effectuées pour transiter dans le système de transitions.

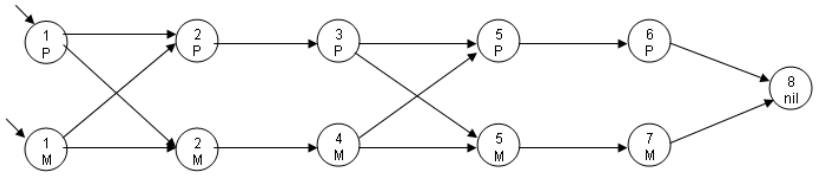

Fig. 4.8 – Système de transitions SMV : abstraction de l'IHM exclusive

Les tableaux 4.6 et 4.7 montrent le code correspondant au système de transitions de la figure 4.8.

MODULE main
% *déclaration des variables*
VAR
etat : {1,2,3,4,5,6,7,8} ;
action :{nil, parole, manipulation} ;
ASSIGN
% *Initialisation des variables*
init(etat) := 1 ;
init(action) :={manipulation,parole} ;
% *Définition de l'état suivant en fonction de l'état courant du système*
next(etat) :=
case
etat = 1 : 2 ; etat = 2 : {3, 4} ;
etat = 3 : 5 ; etat = 4 : 5 ;
etat = 5 : {6,7} ; etat = 6 : 8 ;
etat = 7 : 8 ;
1 : etat ;
esac ;

Tab. 4.6 – Code SMV de l'IHM3 de type exclusif

Deux propriétés ont été spécifiées pour la tâche $RemplirFrom$: l'équivalence et la complémentarité. La propriété d'équivalence est vérifiée par le système tandis que la propriété de complémentarité ne l'est pas. Ces propriétés sont exprimées en logique temporelle CTL dans la clause SPEC. La première formule exprime la

```
% Définition de la modalité suivante en fonction de l'état et de la modalité courante
next(action) :=
case
etat = 1 : {parole, manipulation} ;
(etat = 2) et (action=parole) : {parole} ;
(etat = 2) et (action=manipulation) :{manipulation} ;
etat = 3 : {parole,manipulation} ;
etat = 4 : {manipulation, parole} ;
(etat = 5) et (action=parole) : {parole} ;
(etat = 5) et (action=manipulation) :{manipulation} ;
etat = 6 : nil ;
etat = 7 : nil ;
1 : action ;
esac ;
SPEC
% Définition des formule CTL à vérifier sur le système ···
```

Tab. 4.7 – Code SMV de l'IHM3 de type exclusif

propriété de complémentarité des deux modalités *parole* et *manipulation directe* pour la tâche $RemplirFrom$ (voir tableau 4.8). Elle exprime en tois parties, que :

1. il existe au moins un chemin d'exécution dans le système de transitions tel que si un état est identifié comme l'état initial de la tâche d'interaction $RemplirFrom$ $(etat = 2)$, alors il existe (E) une exécution tel que tous les états suivants sont accessibles par les modalités *parole* ou *manipulation directe* jusqu'à ce que l'état final est atteint $(etat = 5)$;
2. il n'existe aucun chemin qui démarre de l'état 2 et se termine à l'état 5 tel que toutes les transitions sont effectuées seulement par la modalité *parole* ;
3. il n'existe aucun chemin qui démarre de l'état 2 et se termine à l'état 5 tel que toutes les transitions sont effectuées seulement la modalité *manipualtion directe*.

$$\left.\begin{array}{l}(\mathbf{EG}((etat = 2)\\ \Rightarrow \mathbf{E}(((action = parole) or (action = manipulation)) \bigcup (etat = 5)))\end{array}\right\} (1)$$

and

$$not\mathbf{EG}((etat = 2) \Rightarrow \mathbf{E}((action = parole) \bigcup (etat = 5))) \quad (2)$$

and

$$not\mathbf{EG}((etat = 2) \Rightarrow \mathbf{E}((action = manipulation) \bigcup (etat = 5))) \quad (3)$$

Tab. 4.8 – Propriété de complémentarité : non satisfaite

Cette propriété n'est pas satisfaite par le système de l'IHM. SMV donne un contre exemple pour montrer une exécution où la formule n'est pas satisfaite.

La figure 4.9 montre, en guise de contre exemple, une trace ne vérifiant pas la propriété de complémentarité. En effet, chaque ligne désigne l'évolution d'une variable dans le temps. Chaque colonne désigne l'état du système à un temps donné. Le temps est séquentiel et discret allant de 1 à 6. On voit clairement sur la trace de ce contre exemple que la valeur de la variable action est toujours égale à la valeur parole de l'état 2 jusqu'à arriver à l'état 5. Ceci ne vérifie pas la condition numéro 2 dans la formule de logique temporelle CTL citée plus haut.

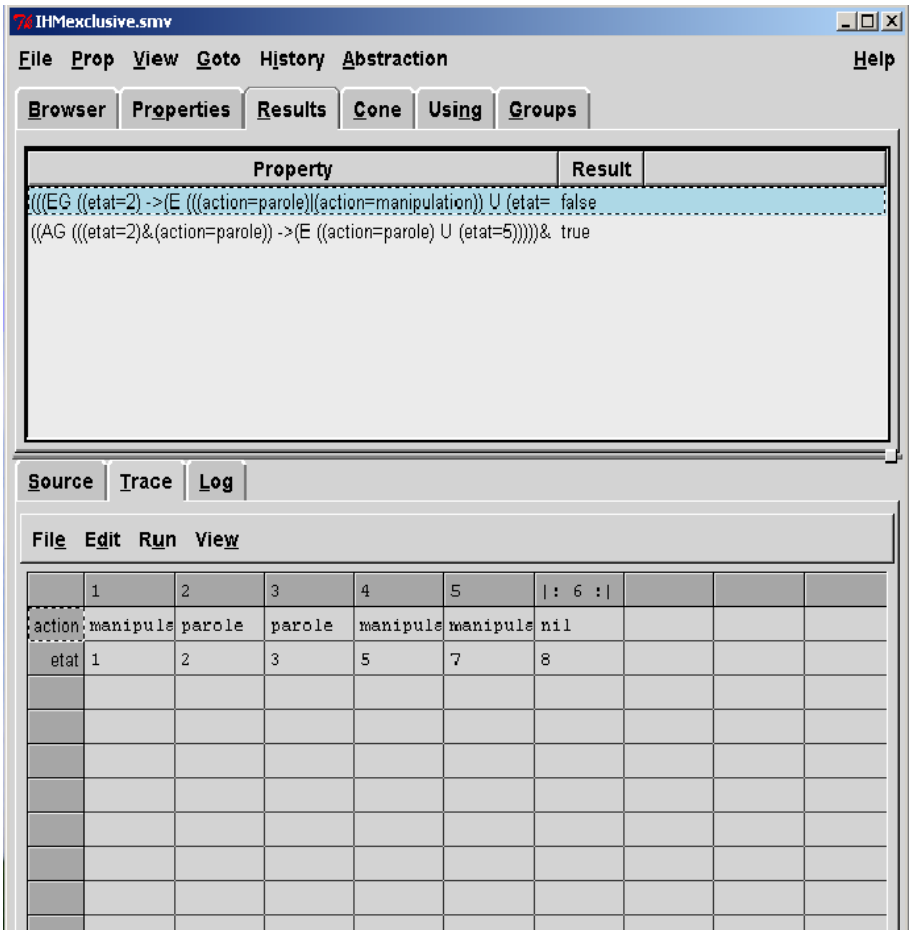

Fig. 4.9 – Résultats de la vérification de l'IHM de type exclusif

La deuxième formule (tableau 4.9) exprime la propriété d'équivalence pour la tâche $RemplirFom$ identifiée par l'état initial 2 et l'état final 5. Elle est exprime en deux parties, que sur tous les chemins d'exécution :

1. si un état est identifié comme l'état initial de la tâche d'interaction multimodale $RemplirFrom$ $(etat = 2)$, alors il existe (E) au moins un chemin tel que ses états sont atteints toujours (AG) par la modalité *parole* jusqu'a ce que l'état final soit atteint $(etat = 5)$;

2. elle exprime la même chose que dans (1) mais avec la modalité *manipulation directe*.

$$\left.\begin{array}{l}(\mathbf{AG}(((etat = 2) and (action = parole)) \\ \Rightarrow \mathbf{E}((action = parole) \bigcup (Etat = 5)))))\end{array}\right\}(1)$$
$$\left.\begin{array}{l}and(\mathbf{AG}(((etat = 2) and (action = manipulation)) \\ \Rightarrow \mathbf{E}((action = manipulation) \bigcup ((etat = 5))))))\end{array}\right\}(2)$$

Tab. 4.9 – Propriété d'équivalence : satisfaite

Cette propriété est bien satisfaite par le système correspondant à l'IHM.

Code de l'IHM alternée. La figure 4.10 rappelle le système de transitions correspondant à l'IHM de type alterné. Comme pour le système de l'IHM de type exclusif, la tâche *RemplirFrom* est identifiée par son état initial 2 et son état final 5. Le système de transitions de la figure 4.11 est obtenu en intégrant les modalités des actions dans les états du système. De la même manière que pour le système de l'IHM de type exclusive, nous désignons par P la modalité parole et par M la modalité manipulation directe. C'est un système obtenu en abstrayant les actions par les modalités qui les génèrent.

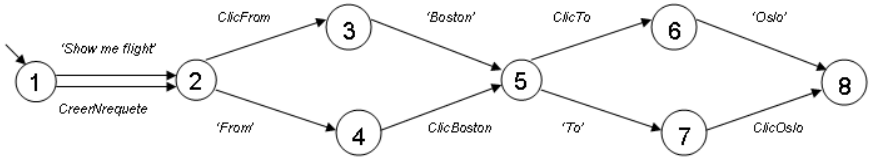

Fig. 4.10 – Système de transitions de l'IHM alternée

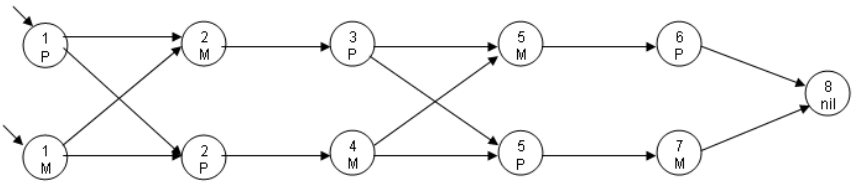

Fig. 4.11 – Système de transitions SMV : abstraction de l'IHM de type alterné

Le tableau 4.10 montre le code correspondant au système de transitions de la figure 4.11 issu de l'abstraction.

Les même propriétés vérifiées sur le système de l'IHM de type alterné ont été vérifiées sur le système de l'IHM de type exclusive, à savoir la propriété de complémentarité 4.8 et la propriété d'équivalence 4.9. Contrairement à l'IHM de type alterné, la

MODULE main
% *Déclaration des variables*
VAR
etat : {1,2,3,4,5,6,7,8} ; action :{nil, parole, manipulation} ;
ASSIGN
% *Initialisation des variables*
init(etat) := 1 ; init(action) :={manipulation,parole} ;
% *Définiton de l'état suivant en fonction de l'état courant du système*
next(etat) :=
case
etat = 1 : 2 ; etat = 2 : {3, 4} ;
etat = 3 : 5 ; etat = 4 : 5 ; etat = 5 : {6,7} ;
etat = 6 : 8 ; etat = 7 : 8 ;
1 : etat ;
esac ;
% *Définition de la modalité suivante en fonction de l'état et de la modalité courante*
next(action) :=
case
etat = 1 : {parole, manipulation} ; (etat = 2) et (action=manipulation) : {parole} ;
(etat=2) et (action=parole) : {manipulation} ;
etat = 3 : {parole,manipulation} ; etat = 4 : {manipulation, parole} ;
(etat = 5) et (action=manipulation) : {parole} ;
(etat=5) et (action=parole) : {manipulation} ;
etat = 6 : nil ; etat = 7 : nil ;
1 : action ;
esac ;
SPEC
% Définition des formules CTL à vérifier sur le système ···

Tab. 4.10 – Code SMV de l'IHM de type alterné

propriété de complémentarité des deux modalités (*parole* et *manipulation directe*) est satisfaite par le système de l'IHM de type exclusive, tandis que la propriété d'équivalence ne l'est pas.

Nous présentons en annexe une mise en oeuvre utilisant la technique de model-checking avec SMV, pour un système de CAO (Conception Assisté par Ordinateur) que nous avons tréaité dans [Kamel & Ait-Ameur, 2005]. Cette mise en oeuvre montre l'intérêt de la modélisation formelle de l'interaction dans un système de CAO.

4.3 Mise en oeuvre avec Promela/Spin

Dans ce qui suit, nous présentons la mise en oeuvre de notre modèle en utilisant le model-checker Promela/Spin [Holzmann, 1991]. Cet outil permet de décrire les systèmes à l'aide de systèmes de transitions et de vérifier des propriétés exprimées à l'aide de la logique temporelle LTL. C'est un outil largement utilisé pour la spécification et vérification des protocoles de communication [Ruys & Langerak, 1997]. L'outil 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 permet d'empêcher l'entrelacement d'autres actions avec cette suite d'actions.
- et la vérification des propriétés temporelles.

Les capacités du langage promela permettent d'exprimer une grande variété de comportements.

4.3.1 Principe de modélisation en Promela/Spin

Avant d'exposer notre modèle générique en Promela/SPIN, nous présentons dans cette section les principes de modélisation de vérification d'un système en Promela/SPIN. Nous montrons comment un système de transitions est codé en Promela ainsi que les propriétés à vérifier sur le système.

4.3.1.1 Système de transitions en Promela

Les systèmes de transitions sont décrits dans le langage Promela. Ce dernier est un langage de spécification dont la syntaxe est proche de celle du langage C avec quelques primitives de communication. Il permet de décrire le comportement de chacun des processus d'un système, et les interactions entre ces processus. La communication entre les processus se fait par transmission de messages via des canaux ou par variables partagées. La communication via les canaux peut être synchrone ou asynchrone. La synchronisation est réalisée à l'aide de rendez-vous sur les ports de communication.

Le tableau 4.11 montre la structure générale d'un modèle décrit en Promela. Les types et les variables globales sont décrits en premier. Dans cet exemple on trouve la déclaration d'un type enuméré *mtype*, une variable i de type *byte* , une variable x de type *mtype*, et une variable f de type *boolean*.

Chaque processus est décrit dans un module local précédé par le mot reservé **proctype**. L'initialisation du système se fait dans le module **init**. Le lancement d'exécution des différents processus se fait au niveau du module init à l'aide de l'instruction **run**.

```
mtype = {val1, val2, ···}
byte i;
mtype x;
bool f;
proctype processus1() {
···
}
proctype processus2() {
···
}
init {
run processus1; run processus2 ···
}
```

Tab. 4.11 – Structure générale d'un système en Promela

Le tableau 4.12 présente un exemple de processus décrit en Promela. Ce processus reçoit la valeur de i, la décrémente dans le cas où elle vaut 1 et l'incrémente dans le cas où elle vaut 0.

```
proctype processus1 (byte i) {
do
: : (i == 0) -> i++
: : (i == 1) -> i- - -
od
}
```

Tab. 4.12 – Exemple d'un processus en Promela

Etant donné un système spécifié en Promela, Spin peut effectuer une simulation aléatoire ou dirigée de l'exécution du système et génère un programme C. Le programme généré peut être utilisé pour effectuer la vérification des propriétés du système ou pour animer le système.

4.3.1.2 Propriétés

Les propriétés du système sont exprimées par des formules de la logique temporelle linéaire LTL. L'outil Spin permet d'animer l'exécution et de consulter le contenu des variables pendant l'exécution. Ceci permet de bien débuguer la spécification.

4.3.2 Principes de représention du modèle générique dans Promela/Spin

Nous présentons dans cette section les règles que nous définissons pour pouvoir coder notre modèle générique en PROMELA/SPIN.

4.3.2.1 Système de transitions et modalités

De la même manière que pour la mise en ouvre dans SMV, nous considérons deux variables dans le système. Une variable qui désigne l'état du système et une autre variable pour designer la modalité utilisée pour les actions d'interactions. Contrairement au langage d'entrée de SMV, Promela est proche des langages évolués ce qui facilite la transcription des opérateurs de composition de notre model formel de conception. Notons que nous utilisons le mécanisme de variables partagées dans notre modélisation.

4.3.2.2 Les opérateurs de composition

Nous avons codé les différents opérateurs de composition de notre modèle dans le langage Promela comme suit :

L'opérateur de $préfixage$; est réalisé par l'instruction de séquence en Promela (;) dont la syntaxe est $inst1; inst2$ telles que $inst1$ et $inst2$ sont deux instructions en PROMELA.

L'opérateur de $séquence >> P >> Q$ est réalisé par l'instruction *goto*. A la fin d'un processus P on insère une instruction *goto* Deb_Q où Deb_Q est l'étiquette de l'instruction qui va vers le début du processus Q.

Deb_P : $instP1$;
⋮ ;
$instPm$;
goto Deb_Q ;
Deb_Q : $instQ1$;
⋮;
$instQn$;

L'opérateur de $choix$ **[]** est réalisé en utilisant l'instruction conditionnelle IF. Le choix entre deux processus P et Q est donné sous la forme suivante :

If
: : $etat_{PQ}$ =1 -> goto $etat1\_P$
: : $etat_{PQ}$ =1 -> goto $etat1\_Q$
Fi

où $etat_{PQ}$ est l'état initial du processus composé $P[]Q$, $etat1\_P$ est l'état initial du processus P et $etat1\_Q$ est l'état initial du processus Q.

L'opérateur de *l'entrelacement* $|||$ est réalisé par l'instruction $Run(P)$ qui permet de lancer l'exécution en entrelacement (interleaving) d'un processus identifié par P. L'instruction *Run* ne peut être lancée qu'à l'initialisation du système. Ceci ne permet pas d'exprimer le parallélisme à plusieurs niveaux d'abstraction des tâches, ce qui limite notre expérimentation avec cet opérateur.

4.3.2.3 Expression des propriétés dans LTL

De la même manière que pour la mise en oeuvre avec SMV, nous procédons à l'abstraction des actions d'interaction en remplaçant chacune des actions par la modalité qui la génère. Ainsi une variable *mod* est introduite pour remplacer la variable *action*.
La variable *mod* est mise à jour par la valeur de la modalité correspondante au type d'action effectuée dans une transition. Cette mise à jour est exécutée de façon *atomique* en même temps que l'instruction de changement des valeurs de la variable d'état comme suit :

$$\boxed{Atomic(mod := mi; instruction)}$$

où *instruction* désigne l'instruction qui permet de changer les valeurs des variables d'état du système.

Nous donnons les schémas de chacune des propriétés CARE par une formule de logique temporelle LTL. Le schéma correspond à une propriété vérifiée par une tâche T caractérisée par son état initial $etatI$ et son état final $etatF$.

Complémentarité Deux modalités m_1 et m_2 sont dites complémentaires pour la réalisation de la tâche T si la formule donnée sur le tableau 4.13 est vérifiée par le système.

$$\boxed{\begin{array}{l} G((etat = etatI) \Rightarrow ((mod = m_1 \ or \ mod = m_2) \bigcup (etat = etatF))) \ \} (1) \\ and \\ G(not((etat = etatI) \Rightarrow (mod = m_1 \cup (etat = etatF)) \ \Big\} (2) \\ and \\ G(not((etat = etatI) \Rightarrow ((mod = m_2) \cup (etat = etatF)))) \ \Big\} (3) \end{array}}$$

Tab. 4.13 – Formule générique LTL de la propriété de complémentarité

Cette formule signifie que dans le système de transitions de l'IHM (toutes les traces d'exécution),

1. si un état est identifié comme étant l'état initial de la tâche T ($etat = etatI$) alors tous les états qui le suivent sont atteints soit par la modalité $m1$ ou $m2$ ($mod = m_1$ *or* $mod = m_2$) jusqu'à ($\cup$) la fin de la réalisation de cette tâche T ($etat = etatF$) ; et
2. il n'existe pas d'exécution telle que si un état est identifié comme étant l'état initial alors tous les états qui le suivent sont atteints avec la modalité m1 jusqu'à ($\cup$) l'état final de la tâche T ; et
3. il n'existe pas d'exécution telle que si un état est identifié comme étant l'état initial alors tous les états qui le suivent sont atteints avec la modalité $m2$ jusqu'à ($\cup$) l'état final de la tâche T ;

Assignation Une modalité m_i est assignée à la tâche T si la formule donnée sur le tableau 4.14 est vérifiée par le système de l'IHM.

$$\boxed{G((etat = etatI) \Rightarrow ((mod = mi) \bigcup (etat = etatF)))}$$

Tab. 4.14 – Formule générique LTL de la propriété d'assignation

Cette formule signifie que dans le système de transitions de l'IHM (toutes les traces d'exécution), si un état $etatI$ est identifié comme étant l'état initial de la tâche T ($etat = etatI$), alors tous les états qui le suivent sont atteints par la modalité mi jusqu'à la fin de la réalisation de la tâche T ($etat = etatF$).

Redondance Pour les mêmes raisons que pour la logique CTL, nous n'avons pas pu exprimer la propriété de redondance dans la logique temporelle LTL.

Equivalence Deux modalités m_1 et m_2 sont dites équivalentes pour la réalisation de la tâche T si la formule donnée sur le tableau 4.15 est vérifiée par le système :

$$\boxed{\begin{array}{c} G((etat = etatI) \Rightarrow (((mod = m_1) \cup (etat = etatF)) \\ \text{or}((\text{mod}{=}\text{m2}) \bigcup (etat = etatF)))) \end{array}}$$

Tab. 4.15 – Formule générique LTL de la propriété d'équivalence

Cette formule signifie que dans le système de transitions de l'IHM (toutes les traces d'exécution), si un état $etatI$ est identifié comme étant l'état initial de la tâche T, alors tous les états qui le suivent sont atteints pas la modalité $m1$ jusqu'à atteindre l'état $etatF$ identifiant la fin de la tâche T, ou bien ils sont atteints par la modalité $m2$ jusqu'à atteindre l'état $etatF$ identifiant la fin de la tâche T.

4.3.3 Application à l'étude de cas

Nous implémentons les mêmes cas mis en oeuvre en SMV. Les tableaux 4.16 et 4.17 montrent le code correspondant à l'IHM de type alterné, et les tableaux 4.18 et 4.19 montrent le code correspondant à l'IHM de type exclusif.

```
mtype = {parole, manipulation, nil}; % les modalités du système
byte etat,random;
mtype action; % variable désignant la modalité utilisée

proctype alterne()
{    random =1;
% initialisation du système à l'état 1. La modalité est initialisée soit
% avec parole soit avec manipualtion directe
    if
       : :random =1 -> atomic{action =parole;etat =1};
       : :random=1 -> atomic{action=manipulation;etat =1};
    fi;
    goto E1;
% première transition vers l'état 2. La modalité passe soit à parole soit
% à manipulation directe d'une manière aléatoire
E1 : random=1;
    if
       : : random==1 -> atomic{action=parole;etat=2};
       : : random==1 -> atomic{action=manipulation;etat=2};
    fi;
    goto E2;
% transition du système vers l'état 3
E2 : if
       : : action == manipulation -> atomic{action=parole; etat=3};
    goto E34;
% transition du système vers l'état 4
       : : action==parole-> atomic{action=manipulation; etat=4};
    goto E34;
    fi;
```

Tab. 4.16 – Code Promela de l'IHM de type alterné

```
% transition du système vers l'état 5
E34 : random=1 ;
    if
       : : random==1 -> atomic{action=parole ;etat=5} ;
       : : random==1 -> atomic{action =manipulation ;etat=5} ;
    fi ;
    goto E5 ;
% transition du système vers l'état 6 ou 7 selon la modalité courante
E5 : if
       : : action=manipulation -> atomic{etat=6 ;action=parole} ;
       : : action=parole-> atomic{etat=7 ;action=manipulation} ;
    fi ;
   goto E67 ;
% transition du système vers l'état final 8
E67 : atomic{etat=8 ; action=nil} ;
}
init {
    run alterne() ;
}
```

Tab. 4.17 – Code Promela de l'IHM de type alterné : suite

La variable *random* a été introduite pour réaliser le non déterminisme. L'instruction $IF..FI$, choisira de manière aléatoire les deux instructions gardées par la même garde $random = 1$. Les transitions sont réalisées d'une manière ininterrompue avec l'instruction *atomic*.

```
mtype = {parole, manipulation, nil} ; % les modalités du système
byte etat,random ;
mtype action ; % variable désignant la modalité utilisée
proctype exclusif()
{    random =1 ;
% initialisation du système à l'état 1. La modalité est initialisée soit
% avec parole soit avec manipualtion directe
    if
       : :random =1 -> atomic{action =parole ;etat =1} ;
       : :random=1 -> atomic{action=manipulation ;etat =1} ;
    fi ;
    goto E1 ;
% premiere transition vers l'état 2. La modalité passe soit à parole soit
% à manipulation directe d'une manière aléatoire
E1 : random=1 ;
    if
       : : random==1 -> atomic{action=parole ;etat=2} ;
       : : random==1 -> atomic{action=manipulation ;etat=2} ;
    fi ;
    goto E2 ;
% transition du système vers l'état 4
E2 : if
       : : action == manipulation -> atomic{action=manipulation ; etat=4} ;
    goto E34 ;
% transition du système vers l'état 3
       : : action==parole-> atomic{action=parole ; etat=3} ;
    goto E34 ;
    fi ;
```

Tab. 4.18 – Code Promela de l'IHM de type exclusif

Les propriétés de complémentarité et d'équivalence des deux modalités parole et manipulation directe ont été vérifiées à l'aide du vérificateur SPIN. Chaque propriété est exprimée par une formule de logique temporelle LTL. SPIN génère le code Promela correspondant afin de générer le système de transitions correspondant nécessaire pour l'algorithme de vérification.

La formule correspondant à la propriété de complémentarité est définie sur le tableau 4.20. Elle est définie en trois parties sur tous les chemins d'exécutions :

1. si un état est identifié comme étant l'état initial de la tâche $RemplirFrom$ ($etat = 2$) alors tous les états qui le suivent sont atteints soit par la modalité *parole* ou *manipulation directe* ($mod = parole\ or\ mod = manipulation\ directe$)

```
%transition du système vers l'état 5
E34 : random=1 ;
    if
       : : random==1 -> atomic{action=parole ;etat=5} ;
       : : random==1 -> atomic{action =manipulation ;etat=5} ;
    fi ;
    goto E5 ;
%transition du système vers l'état 6 ou 7 selon la modalité courante
E5 : if
       : : action=manipulation -> atomic{etat=7 ;action=manipulation} ;
       : : action=parole-> atomic{etat=6 ;action=parole} ;
    fi ;
   goto E67 ;
%transition du système vers l'état final 8
E67 : atomic{etat=8 ; action=nil} ;
}
init {
    run exclusif() ;
}
```

Tab. 4.19 – Code Promela de l'IHM de type exclusif : suite

jusqu'à ($\cup$) la fin de la réalisation de cette tâche $RemplirFrom$ $(etat = 5)$; et

2. il n'existe pas d'exécution telle que si un état est identifié comme étant l'état initial $(etat = 2)$ alors tous les états qui le suivent sont atteints avec la modalité *parole* jusqu'à ($\cup$) l'état final de la tâche $RemplirFrome$ $(etat = 5)$; et
3. il n'existe pas d'exécution telle que si un état est identifié comme étant l'état initial alors tous les états qui le suivent sont atteints avec la modalité *manipulation directe* jusqu'à ($\cup$) l'état final de la tâche $RemplirFrom$;

$$
\begin{array}{l}
\left.\begin{array}{l} G((etat = 2) \Rightarrow ((action = parole \ or \ action = manipulation) \\ \qquad\qquad \bigcup (etat = 5))) \end{array}\right\} (1) \\
and \\
\left.\begin{array}{l} G(not((etat = 2) \Rightarrow (action = parole \cup (etat = 5)) \\ and \end{array}\right\} (2) \\
\left.\begin{array}{l} G(not((etat = 2) \Rightarrow ((action = manipulation) \cup (etat = 5)))) \end{array}\right\} (3)
\end{array}
$$

Tab. 4.20 – Formule LTL de la propriété de complémentarité : étude de cas

La formule correspondant à la propriété d'équivalence est définie sur le tableau 4.21. Elle signifie que dans le système de transitions de l'IHM (toutes les traces d'exécution), si un état est identifié comme étant l'état initial de la tâche $RemplirFrom$

($etat = 2$), alors tous les états qui le suivent sont atteints pas la modalité *parole* jusqu'à atteindre l'état 5 identifiant la fin de la tâche $RemplirFrom$, ou bien ils sont atteints par la modalité *manipulation directe* jusqu'à atteindre l'état 5 identifiant la fin de la tâche $RemplirFrom$.

$$G((etat = 2) \Rightarrow (((action = parole) \cup (etat = 5)) \text{ or}((\text{action=manipulation}) \bigcup (etat = 5))))$$

Tab. 4.21 – Formule LTL de la proprieté d'équivalence : étude de cas

Les deux formules sont obtenues en remplaçant dans la formule générique correspondant à ces propriétés, les variables correspondant à l'état et la modalité (état et action) et les valeurs correspondantes à l'état initial et final (2 et 5) de la tâche ainsi que les valeurs des modalités (*parole*, *manipulation directe*).

La figure 4.12 montre l'exécution de la vérification de la propriété d'équivalence des deux modalités *parole* et *manipulation directe* pour la tâche $RemplirFrom$ sur le système de l'IHM de type alterné. La formule à vérifier est saisie dans le champs texte *formula*. Dans l'espace *Never Claim* l'automate correspondant à la négation de la formule à vérifiée est genéré. Dans l'espace *Verification Result* on trouve le résultat de l'exécution qui est *not valid* dans ce cas. Dans ce cas (formule non vérifiée) SPIN propose une exécution pas à pas afin de localiser l'erreur.

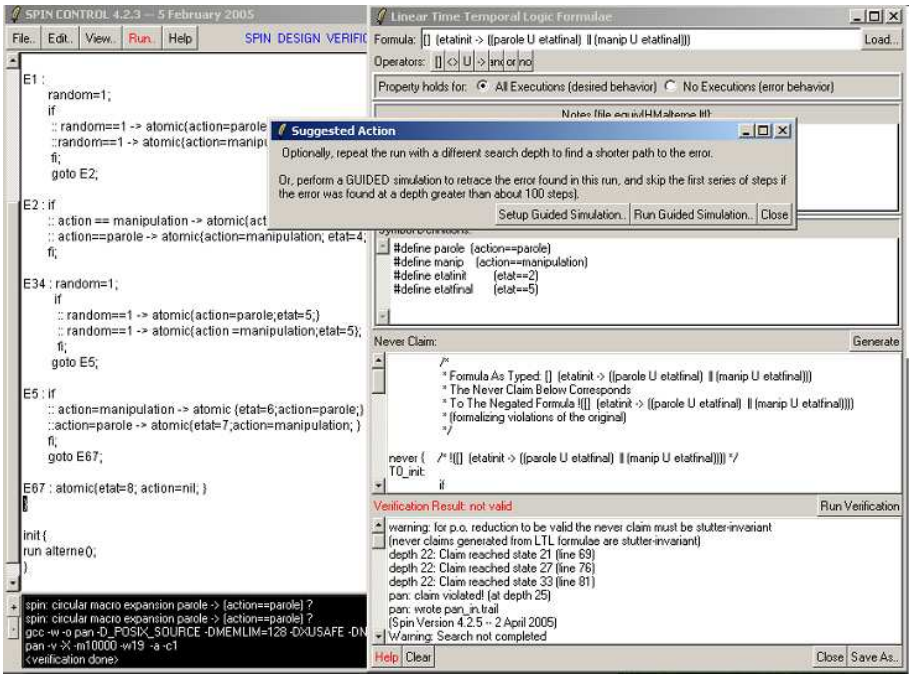

Fig. 4.12 – Vérification de l'équivalence pour l'IHM de type alterné

4.4 Utilisation de vérificateurs sur modèle pour les IHM3

Nous avons présenté dans ce chapitre les deux mises en oeuvre que nous avons effectuées dans la technique de model-checking : SMV et SPIN. Bien que les deux techniques partagent les même principes, elles diffèrent sur un ensemble de points que nous relevons dans ce qui suit

Visualisation. Spin permet de voir l'exécution pas à pas avec visualisation du contenu des variables, ce qui permet de mieux localiser les erreurs contrairement à SMV qui donne la trace représentant le contre exemple sans faire référence aux instructions qui génèrent ces erreurs.

Taille du système. Au niveau des algorithmes de vérification utilisés par chacun des outils, celui de SMV est plus efficace car il est symbolique et est basé sur les structures de BDD ce qui permet de coder des systèmes plus complexes en taille que celui de Spin.

Expression du système. Promela, langage d'entrée pour SPIN, est proche des langages de haut niveau, ce qui a permis la description du système de transitions plus facilement que le langage d'entrée de SMV. Les opérateurs de composition de notre modèle formel peuvent être traduits d'une manière plus automatique. Une des difficultés que nous avons rencontrée avec SMV est la modélisation du système. Le langage d'entrée de SMV impose la description du système de transitions de chacune des variables d'état. La description du système de transitions du système global, qui découle des descriptions de ces variables d'états, impose la prise en considération les relations qui existent entre ces différentes variables.

Expression des propriétés. Les deux logiques temporelles CTL et LTL ont le même pouvoir d'expression pour les propriétés CARE. Grâce aux deux logiques temporelles CTL et LTL, les propriétés de Complémentarité, d'Equivalence et d'Assignation ont été vérifiées, mais il nous a paru impossible d'exprimer la propriété de Redondance à cause du manque du pouvoir d'expression de l'opérateur du parallèle dans les deux logiques temporelles.

La technique de model-checking a pour avantage le fait qu'elle est automatique et permet le debugage de la spécification pour corriger des éventuelles erreurs. Elle permet également de vérifier des propriétés dynamiques avant l'implémentation du système. Les limites de cette technique sont définies par l'explosion combinatoire des états du système, ce qui rend la vérification complexe. Les efforts effectués par la communauté des chercheurs ont permis de réduire ces limites en développant de nouveaux algorithmes tel que celui utilisé par SMV, de nouvelles techniques de décomposition, etc.

Dans le chapitre suivant, nous présentons une mise en oeuvre en utilisant la technique de preuve avec le raffinement en utilisant l'atelier B.

4.5 Conclusion

Nous avons montré dans ce chapitre la mise en oeuvre de notre modèle générique dans deux techniques de model-checking. Nous avons montré comment les systèmes de transitions de l'IHM issus du modèle générique sont codés dans les deux langages d'entrée de SMV et Promela. Dans les deux techniques, la transcription des systèmes de transitions a été effectuée sans difficulté particulière bien que nous notons plus de souplesse avec Promela qu'avec SMV. En effet, SMV utilise les états et les variables d'états pour décrire un système ce qui nous a obligé à coder les actions dans les états, alors que Promela exprime explicitement les actions qui font transiter d'un état à un autre. Dans les deux cas, les modélisations ont été réalisées en totalité et tous les éléments issus de nos modèles génériques ont pu être représentés.

Nous avons exprimé également toutes les propriétés CARE, sauf celle de la redondance qui nécessite une logique pouvant exprimer le parallélisme. Nous avons exprimé ces propriétés dans des schémas génériques ce qui permet de les appliquer facilement pour tout type de système.

Nous n'avons pas mesuré la complexité des vérifications de propriétés ni l'impact des modélisations sur cette complexité. Par contre, l'abstraction que nous avons effectuée en n'utilisant que les variables exprimant les changements de modalités a réduit la taille des systèmes de transitions sur lesquels portait la vérification, et a donc diminué le coût de cette vérification.

Chapitre 5

Mise en oeuvre du modèle générique dans une technique de preuve avec B

5.1 Introduction

Ce chapitre présente la mise en oeuvre de notre modèle générique à l'aide des systèmes d'événements en utilisant la méthode B dans sa version événementielle. Avant de présenter cette mise en oeuvre, nous commençons tout d'abord par une brève présentation de la méthode B et du langage B événementiel que nous utiliserons pour coder notre modèle. Dans cette technique de preuve, le système de transitions est codé par le système d'événements du langage B événementiel et les propriétés sont exprimées dans la logique du premier ordre. La conception dans ce cas procède par décomposition. Elle est descendante et utilise le raffinement comme opération de décomposition.

5.2 Méthode de spécification B

La méthode B couvre les différentes étapes du développement d'un logiciel, depuis la spécification abstraite jusqu'au dernier raffinement qui conduit à la traduction directe en une implémentation dans un langage de programmation. Elle se base sur les raffinements successifs pour modéliser les systèmes en passant d'une modélisation abstraite jusqu'à une modélisation concrète.

Seuls les éléments nécessaires à la compréhension de notre utilisation de B seront présentés dans cette section. Plus d'informations sur la méthode B événementiel se trouvent dans [Abrial, 1996b].

5.2.1 Machine abstraite

La *machine abstraite* constitue le mécanisme de structuration de base dans le processus de développement en B. Elle représente la modélisation d'un système informatique composé d'un ensemble de variables permettant de décrire l'état du système et des opérations permettant de modifier les valeurs de ces variables.

Dans une machine abstraite B, une spécification est décrite à l'aide d'un ensemble de clauses, de structures de données et de structures de contrôle. Le tableau 5.1 montre l'organisation des différentes clauses pouvant être présentes dans la spécification d'une machine abstraite.

Les variables de la machine, déclarées dans la clause VARIABLES, sont contraintes par des propriétés déclarées dans la clause INVARIANT. Par exemple, c'est dans la clause INVARIANT que les variables sont typées. Les opérations sont exprimées par les substitutions généralisées et font évoluer les variables dans différents états qui doivent tous satisfaire l'invariant. Les formules du langage sont exprimées dans la logique des prédicats du premier ordre.

Machine nomdelamachine
SETS
définition des ensembles
CONSTANTS
définition des constantes
PROPERTIES
définition de propriétés logiques sur les constantes
VARIABLES
définition des variables de la machine
INVARIANT
définition de l'invariant
ASSERTIONS
définition des assertions
INITIALISATION
définition des valeurs initiales des variables
OPERATIONS
définition des opérations de la machine
END

Tab. 5.1 – Modèle général d'une machine abstraite

5.2.2 Substitutions généralisées et obligations de preuve

La cohérence d'une machine abstraite est assurée par la validation d'obligations de preuve (OP). Une OP est un théorème à démontrer, indiqué par la théorie de B

et généré à partir de la description du système en question.

La modélisation des modifications des données des machines abstraites s'effectue au sein des opérations à l'aide d'un pseudo-code nommé *substitutions*. Les substitutions sont des notations mathématiques qui jouent le rôle de transformateurs de prédicats. Ces substitutions se basent sur le calcul de la plus faible précondition [Dijkstra, 1976].

Les substitutions permettent également d'établir systématiquement des obligations de preuve à partir des composants B (machines abstraites, raffinements ou implantations). Quand une substitution est utilisée, le système de génération d'obligations de preuve associé au système de preuve automatique s'assure que cette substitution est valide compte tenu des invariants de la machine et des préconditions de l'opération concernée.

MACHINE M
VARIABLES
x
INVARIANT
$I(x)$
ASSERTIONS
$A(x)$
INITIALISATION
$Init(x)$
OPERATIONS
$u \longleftarrow O(w) =$
S
END

Tab. 5.2 – Machine abstraite B avec une opération.

Le tableau 5.2 montre une machine abstraite générique simplifiée en B (sans constantes ni propriétés logiques sur ces constantes)

Les obligations de preuve de cette machine à vérifier sont décrites sur le tableau 5.3.

	Obligation de preuve
INV_1	$[Init(x)]I(x)$
INV_2	$I(x) \Rightarrow [S]I(x)$
INV_3	$I(x) \Rightarrow A(x)$

Tab. 5.3 – Obligations de preuve d'une machine abstraite B.

La première INV_1 concerne l'initialisation (une opération particulière) qui établit l'invariant après l'appel de l'opération d'initialisation (il n'y a pas de valeur avant l'initialisation). La deuxième obligation de preuve INV_2 concerne l'opération O qui préserve l'invariant c'est-à-dire que, sous couvert de l'invariant, l'opération établit l'invariant. Enfin, la troisième INV_3 concerne la clause **ASSERTIONS**. $A(x)$ est un prédicat vérifié dans tout état. Ce n'est pas un invariant, c'est une propriété qu'il est possible de déduire de l'invariant $I(x)$.

Une opération a différentes formes possibles selon le type de substitution utilisé. Par conséquent, selon la substitution employée l'obligation de preuve INV_2 n'est plus la même. Le tableau 5.4 présente les principales substitutions utilisées dans nos modèles.

Nom de la substitution	Syntaxe de l'opération
bloc	**BEGIN** $T(x)$ **END**
précondition	**PRE** $P(x)$ **THEN** $T(x)$ **END**
sélection	**SELECT** $G(x)$ **THEN** $T(x)$ **END**
choix non borné	**ANY** l **WHERE** $G(x,l)$ **THEN** $T(x)$ **END**

Tab. 5.4 – Substitutions utilisées pour définir la forme d'une opération.

Dans le cas où l'opération utilise une substitution *précondition*, l'obligation de preuve INV_2 serait :

$$(INV_2) \quad I(x) \wedge P(x) \Rightarrow [T(x)]I(x)$$

Enfin, le tableau 5.5 définit les différentes formes de substitutions employées pour $T(x)$.

Nom de la substitution	Transformation de prédicat
devient égal	$[x := E]P(x) \equiv [x := E]P(x)$
appel d'opération	$[R \longleftarrow op(E)]P(x) \equiv [X := E; T(x); R := Y]P(x)$
identité	$[skip]P(x) \equiv P(x)$
simultanée	$T(x) \parallel U(y)$ (x et y distinctes)
séquencement	$[T(x); U(x)]P(x) \equiv [T(x)][U(x)]P(x)$

Tab. 5.5 – Substitutions utilisées pour définir le corps d'une opération.

La substitution *appel d'opération* permet d'appliquer la substitution d'une opération, en remplaçant les paramètres formels par des paramètres effectifs. L'appel d'opération se définit sous quatre formes différentes, selon la présence de paramètres d'entrée et de sortie. Si op est définie par $Y \longleftarrow op(X) = S$, la signification d'un appel de $R \longleftarrow op(E)$ (où X est une liste d'expressions représentant les paramètres

d'entrée de *op* et E une liste d'expressions représentant les paramètres d'entrée effectifs de *op*) est obtenu en remplaçant les paramètres formels par ll'expression des paramètres réels (idem pour le résultat).

Quant à la substitution *simultanée*, elle permet la composition en parallèle de deux substitutions.

La méthode B propose des techniques de structuration qui permettent la décomposition d'une machine B. En effet, il n'est pas envisageable de spécifier complètement un système au moyen d'une unique et seule machine abstraite car le nombre d'obligations de preuve serait trop important. Le passage par des raffinements successifs est une solution.

5.2.3 Modèle du B événementiel

En B événementiel, on parle de *modèle* plutôt que de *machine*. Les *opérations* définies en B classique sont remplacées par les *événements* et la clause OPERATIONS est remplacée par la clause EVENTS. La partie dynamique du système est définie par un ensemble d'événements gardés. Chaque événement est gardé par une expression logique et il ne s'exécute que quand cette garde est vérifiée. Les événements peuvent être en concurrence et seules les gardes permettent de définir leur séquencement. Le tableau 5.6 présente la structure d'un modèle B événementiel générique.

Nous présentons ici la notion d'événement, la préservation de l'invariant et le raffinement. Des informations complémentaires peuvent être trouvées dans [Cansell, 2003].

5.2.4 Les événements

Un événement correspond à un changement d'état et modélise donc une transition discrète du système à modéliser. Un événement possède un nom, une garde (condition nécessaire au déclenchement de l'événement) et une action ou encore appelée corps de l'événement. L'action est définie par une substitution qui décrit la manière dont l'événement modifie les variables.

Le langage B est asynchrone. En effet, si deux événements d'un modèle ont leurs gardes vraies au même instant, ils ne sont pas déclenchés en même temps : il y a entrelacement des événements dans un ordre non déterminé. Toutefois, la durée d'exécution d'un événement est considérée comme nulle et cet instant correspond au changement d'état.

Le B événementiel n'admet que trois formes possibles d'événements. Nous les présentons ci-dessous. Dans ce qui suit, nous choisissons la variable x comme variable d'état, définie dans la clause **VARIABLES**.

MODEL Nom
SETS
Noms de types et noms d'ensembles
CONSTANTS
Déclaration du nom des constantes
PROPERTIES
Définition des propriétés logiques des constantes
VARIABLES
Déclaration du nom des variables
INVARIANT
Définition des propriétés statiques par des formules logiques
ASSERTIONS
Définition de propriétés sur les variables et les constantes
INITIALISATION
Description de l'état initial
EVENTS
Énumération des événements associés au modèle
END

Tab. 5.6 – Modèle B événementiel générique.

Événement simple. L'événement suivant est le plus simple. Sa garde est toujours vraie et $S(x)$ est une substitution qui modifie l'état de la variable x :

$nomEvt$ =
BEGIN
$S(x)$
END

Événement gardé. Un événement gardé est une substitution $S(x)$ gardée par l'expression logique $G(x)$. L'événement se déclenche lorsque la garde $G(x)$ est vraie.

$nomEvt$ =
SELECT
$G(x)$
THEN
$S(x)$
END

Événement indéterministe. L'événement suivant est un événement indéterministe gardé par $\exists l.G(l,x)$. Cet événement ne peut se déclencher que s'il existe des valeurs pour les variables locales l qui satisfont la condition $G(x,l)$.

nomEvt =
ANY l **WHERE**
$G(x, l)$
THEN
$S(x, l)$
END

5.2.5 Préservation de l'invariant

Une fois que le système est construit, il faut montrer qu'il est cohérent. Suivant la même démarche que pour le B classique, il faut montrer que l'événement d'initialisation préserve l'invariant et que chaque événement du système préserve également l'invariant. Plus précisément, pour qu'un événement soit faisable, il faut que sous couvert de la garde de l'événement et l'invariant une transition de cet événement soit toujours possible.

Initialisation. L'événement d'initialisation est une substitution de la forme $Init(x)$. Nous avons donc une obligation de preuve identique à celle du B classique :

$$(INV_1) \quad [Init(x)]I(x)$$

Événement simple. Pour une action de la forme $S(x)$ l'obligation de preuve est la suivante :

$$(INV_2) \quad I(x) \Rightarrow [S(x)]I(x)$$

Cet événement est faisable sous couvert de l'invariant $I(x)$ et si le prédicat obtenu après transformation de l'invariant par la substitution $S(x)$ établit cet invariant.

Événement gardé. Pour l'événement gardé par $G(x)$ et de substitution $S(x)$ comme action, l'obligation de preuve est la suivante :

$$(INV_3) \quad I(x) \Rightarrow (G(x) \Rightarrow [S(x)]I(x)) \text{ ou alors } I(x) \wedge G(x) \Rightarrow [S(x)]I(x)$$

Événement indéterministe. Pour l'événement gardé par $\exists l.G(l, x)$ et de substitution $S(x)$ comme action, l'obligation de preuve de l'événement indéterministe est la suivante :

$$(INV_4) \quad I(x) \Rightarrow (\forall l.(G(l, x) \Rightarrow [S(x, l)]I(x)))$$

Récapitulatif des obligations de preuve pour l'invariant. Le tableau 5.7 récapitule les obligations de preuve de préservation de l'invariant.

	Nature de l'événement	Obligation de preuve
INV_1	Initialisation	$[Init(x)]I(x)$
INV_2	Simple	$I(x) \Rightarrow [S(x)]I(x)$
INV_3	Gardé	$I(x) \Rightarrow (G(x) \Rightarrow [S(x)]I(x))$
INV_4	Indéterministe	$I(x) \Rightarrow (\forall l.(G(l,x) \Rightarrow [S(x,l)]I(x)))$

Tab. 5.7 – Obligations de preuve d'un modèle B événementiel.

5.2.6 Raffinements

Un modèle B peut être raffiné. Les éléments du modèle abstrait sont remplacés progressivement par d'autres éléments plus concrets afin d'arriver à un système concret.

Le processus de raffinement consiste à

- remplacer les structures de données abstraites par des structures de données concrètes,
- et remplacer les substitutions abstraites par des substitutions concrètes.

Un raffinement est un modèle dont les comportements sont des comportements du modèle abstrait. Le tableau 5.8 représente la forme générique d'un modèle raffiné.

Il satisfait donc l'invariant du modèle abstrait. Un nouveau invariant est rajouté au modèle du raffinement. Il consiste à lier les variables abstraites (du modèle abstrait) au variables concrètes à celle du modèle du raffinement. Cet invariant est appelé invariant de *collage*. On retrouve les événements du modèle abstrait avec le même nom mais peuvent changer de gardes et de corps. De nouveaux événements sont rajoutés aux modèles du raffinement. Un raffinement est correct si :

- l'initialisation préserve l'invariant
- chaque événement du système préserve l'invariant

Les modèles du B événementiel sont représentés en B classique car, il n'existe pas encore d'outil qui implémente le B événementiel. Lors de la traduction, il faut ajouter dans la clause ASSERTIONS, la propriété qui permet d'assurer que le système est réactif. Cette propriété est exprimée en indiquant que la disjonction des gardes abstraites implique la disjonctions des gardes concrètes. C'est-à-dire que le système concret ne se bloque pas plus que le système abstrait. Enfin, un variant (entier naturel) décrivant une suite décroissante d'entiers naturels doit être introduit pour indiquer que les nouveaux événements (concrets) ne sont pas déclenchés indéfiniment.

REFINEMENT Nom
REFINES
Noms du modèle abstrait
SETS
Noms de types et noms d'ensembles
CONSTANTS
Déclaration du nom des constantes
PROPERTIES
Définition des propriétés logiques des constantes
VARIABLES
Déclaration du nom des variables
INVARIANT
Définition des propriétés statiques par des formules logiques
VARIANT
Définition du variant décrémenté par les événements
ASSERTIONS
Définition de propriétés sur les variables et les constantes
INITIALISATION
Description de l'état initial
EVENTS
Énumération des événements associés au modèle
END

Tab. 5.8 – Refinement B événementiel générique.

5.3 Principes de représentation du modèle générique formel en B événementiel

Dans cette section nous montrons comment notre modèle, dont la sémantique est définie par les systèmes de transitions, peut être codé en B événementiel qui possède également une sémantique à base des systèmes de transitions. En première étape, nous présentons la description des modèles (systèmes de transitions) ensuite, nous présentons comment les propriétés CARE peuvent être exprimées et prouvées également dans le modèle.

5.3.1 Système de transitions

Nous présentons comment nous codons les événements et les actions à chacun des niveaux de raffinement, ainsi que les variables d'états caractérisant les états du système.

5.3.1.1 Actions et événements

Pour tous les niveaux d'abstraction, nous utilisons un ensemble d'étiquettes pour identifier les événements (noms des événements). Les actions interactives de base sont introduites au dernier niveau de raffinement, c'est-à-dire le niveau qui raffine les énoncés. Ces actions interactives sont classées par rapport aux modalités qui les produisent. Ainsi, dans le dernier niveau de raffinement, nous définissons un ensemble d'actions pour chacune des modalités du système. Pour chaque modalité m_i nous définissons l'ensemble des actions qu'elle produit et que nous notons Ens_Modi. Pour chaque ensemble, nous définissons une variable act_modi qui prend ses valeurs dans cet ensemble et qui représente les actions interactives de modalités m_i.

Nous rappelons que le système que nous codons est une abstraction du système réel. Nous remplaçons chaque action par la modalité qui la génère. Cette abstraction est motivée par le fait que les propriétés que nous voulons vérifier sont des propriétés liées à l'utilisation des modalités.

5.3.1.2 Variables d'états

En plus de la variable act_modi, un ensemble de variables est défini. Ces variables sont dites variables d'état car elles définissent l'état de l'IHM lors de la conception. Elles sont modifiées par les différents événements du modèle. A ces variables, est ajouté un ensemble de variants qui représentent des variables permettant de définir les contraintes de précédence entre les événements. Ce sont des variables qui décroissent vers la valeur 0. Les variants sont utilisés dans les raffinements et dans les nouveaux événements qui doivent le faire décroître pour assurer que les événements de l'abstraction sont déclenchés.

5.3.2 Opérateurs de décomposition

Les opérateurs de décomposition définissent le comportement du système raffiné. Ils concernent la partie dynamique du système. Ce sont les mêmes opérateurs de composition utilisés dans notre modèle formel, mais ici nous les définissons comme opérateurs de décomposition car la construction du modèle est descendante par décompositions et raffinements successifs. La représentation des différents opérateurs de composition d'une algèbre de processus en B évènementiel est donnée dans [Ait-Ameur *et al.*, 2005].

Dans le modèle par composition nous construisons un système C à partir de deux sous systèmes A et B tels que $C = Op(A, B)$ avec Op un opérateur de composition. Dans le modèle par décomposition, nous cherchons A et B tels que $C = Op(A, B)$. On représente C par un modèle abstrait raffiné en A et B.

Dans le cas de B événementiel, il s'agit de raffiner les événements. Pour cela, nous définissons pour chaque opérateur de notre modèle générique un cadre générique de

décomposition qui correspond à son raffinement.

Nous supposons $EvtA$, l'événement du modèle abstrait que nous voulons décomposer en $EvtAr_1$ Op $EvtAr_2$ avec Op un opérateur de composition de notre modèle formel. $EvtAr_1$ et $EvtAr_2$ sont deux événements du modèle du raffinement qui raffinent ou décomposent l'événement $EvtA$.

Nous distinguons deux types d'événements : les événements abstraits qui correspondent aux tâches et énoncés du modèle, et les événements concrets qui correspondent aux actions interactives de base. Cette distinction nous amène à définir deux types de décomposition.

Décomposition abstraite. Elle correspond au raffinement d'événements abstraits en d'autres événements abstraits. C'est une décomposition qui produit un système plus détaillé mais reste toujours abstrait par rapport au système réel que nous voulons développer. Les événements manipulés sont indépendants de toute modalité. Ainsi, seules les variables d'états abstraites sont représentées.

Décomposition concrète. Elle correspond au raffinement d'événements abstraits en événements concrets des différents ensembles Ens_Modi. Ce sont les événements du système réel. C'est à ce niveau de raffinement qu'apparaissent les différentes actions interactives de base liées aux différentes modalités.

La figure 5.1 montre à quels niveaux interviennent les décompositions abstraites et concretes. Les Ti dénotent des tâches abstraites, les Eij dénotent des énoncés, et enfin, les $evij$ dénotent les événements concrets.

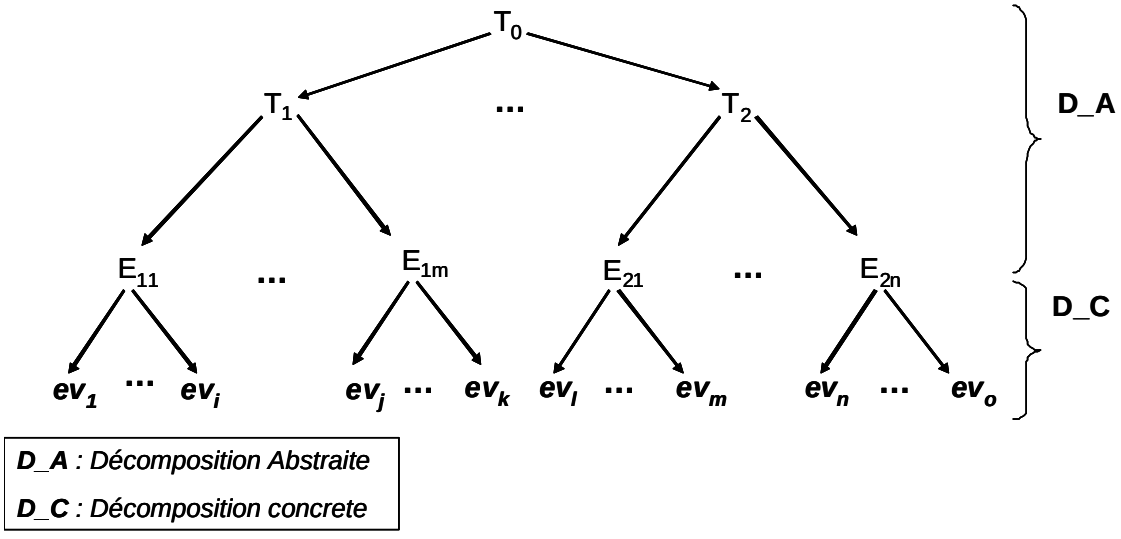

Fig. 5.1 – Décomposition abstraite et concrete

5.3.2.1 Décomposition abstraite

Cette décomposition correspond, dans la grammaire du modèle générique, à la règle qui décompose la tâche en sous tâche jusqu'aux énoncés.

$$T ::= T[]T \mid T >> T \mid T \mid\mid\mid T \mid T \mid\mid T \mid Enonce$$

Le tableau 5.9 présente le schéma du modèle abstrait. L'événement $EvtA$ dont la garde est G_0 et dont le corps est la substitution S_0 sera décomposé en $EvtAr_1$ **Op** $EvtAr_2$ où $EvtAr_1$ et $EvtAr_2$ sont les deux événements qui raffinent l'événement $EvtA$, et Op est l'opérateur de décomposition. Nous représentons chacun des modèles correspondant au raffinement de $M1$ pour chacun des opérateurs de notre modèle.

MODEL $M1$
INVARIANT
$I(var_i)$
EVENTS
$EvtA =$
SELECT
G_0
THEN
S_0
END ;

Tab. 5.9 – Modèle abstrait M1

L'invariant $I(var_i)$ permet la description de propriétés sur les variables d'états (var_i) du modèle.

La séquence >>. Cet opérateur décompose l'événement $EvtA$ en la séquence de deux événements $EvtAr_1$ suivi de $EvtAr_2$ (tableau 5.10). L'exécution de l'événement $EvtAr_1$ est conditionnée par la possibilité de l'exécution de l'événement $EvtA$ dans le modèle abstrait.

Le variant $VarSeq$ est initialisé à la valeur 2 pour permettre à $EvtAr_1$ de s'exécuter en premier. L'exécution de l'événement $EvtAr_1$ modifie la valeur du variant par la valeur 1 afin de permettre à l'événement $EvtAr_2$ de s'exécuter. C'est l'événement $EvtAr_2$ qui donne le contrôle à l'événement $EvtA$ de l'abstraction pour s'exécuter en modifiant la valeur du variant $VarSeq$ par 0.

L'événement qui se déclenchera le premier est défini dans la clause INITIALISATION en affectant la valeur 2 au variant $VarSeq$. L'événement $EvtAr_1$ s'exécute le premier et modifie la valeur du variant pour donner la main à l'événement $EvtAr_2$ qui le modifie à son tour pour donner la main à l'événement $EvtA$ de l'abstraction.

REFINEMENT M_2
REFINES M_1
VARIABLES
$VarSeq$, $vari$, $varj$
INVARIANT
$J(vari, varj) \wedge VarSeq \in \{0, 1, 2\} \wedge J'(varj)$
ASSERTIONS
$G_0 \Rightarrow ((G_1 \wedge VarSeq = 2) \vee$
$(\mathrm{G}_2 \wedge VarSeq = 1) \vee (G'_0 \wedge VarSeq = 0))$
INITIALISATION
VarSeq := 2
EVENTS

$EvtAr_1 =$	$EvtAr_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge VarSeq = 2$	$G_2 \wedge VarSeq = 1$	$G'_0 \wedge VarSeq = 0$
THEN	**THEN**	**THEN**
$S_1 \parallel$	$S_2 \parallel$	S'_0
$VarSeq := 1$	$VarSeq := 0$	**END** ;
END ;	**END** ;	

Tab. 5.10 – Modèle générique du raffinement en séquence

Le Choix [] : L'événement $EvtA$ est décomposé en $EvtAr_1[]EvtAr_2$ (tableau 5.11). Dans ce cas, un des événements $EvtAr_1$ ou $EvtAr_2$ est exécuté et d'une manière exclusive. La possibilité d'exécution de l'événement $EvtA$, exprimée par sa garde G_0, doit exprimer la possibilité de l'exécution de l'un ou l'autre des deux événements qui le décomposent.

Le variant $VarChoix$ est introduit pour contrôler le séquencement entre les événements. Dans ce cas, la variable $VarChoix$ est initialisée aléatoirement à la valeur 1 ou à la valeur 2. Dans le cas où elle est initialisée à 1, la garde de l'événement $EvtAr_1$ peut être vérifiée et permet ainsi l'exécution de son corps. La substitution $VarChoix := 0$ permet de donner le contrôle à l'événement $EvtA$ pour s'exécuter. Le même raisonnement est appliqué dans le cas où la variable $VarChoix$ est initialisée à la valeur 2. Dans ce cas, c'est l'événement $EvtAr_2$ qui s'exécute pour donner la main ensuite à $EvtA$.

L'invariant $J(vari, varj)$ décrit la correspondance entre les variables du modèle du raffinement ($M2$) et celles du modèle abstrait ($M1$). L'invariant $VarChoix \in \{0, 1, 2\}$ permet de typer le variant $VarChoix$. Finalement, l'invariant $J'(varj)$ permet d'exprimer des propriétés supplémentaires sur les variables de la conception apparues dans le raffinement.

REFINEMENT $M2$
REFINES $M1$
VARIABLES
$VarChoix$, $vari$, $varj$
INVARIANT
$J(vari, varj) \wedge VarChoix \in \{0, 1, 2\} \wedge J'(varj)$
ASSERTIONS
$G_0 \Rightarrow ((G_1 \wedge VarChoix = 1) \vee$
$(\mathrm{G}_2 \wedge VarChoix = 2) \vee (G'_0 \wedge VarChoix = 0))$
INITIALISATION
$Var := \{1, 2\}$
EVENTS

$EvtAr_1 =$	$EvtAr_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge VarChoix = 1$	$G_2 \wedge VarChoix = 2$	$G'_0 \wedge VarChoix = 0$
THEN	**THEN**	**THEN**
$S_1 \mid\mid$	$S_2 \mid\mid$	$S'_0 \mid\mid$
$VarChoix := 0$	$VarChoix := 0$	**END** ;
END ;	**END** ;	

Tab. 5.11 – Modèle générique du raffinement en choix

L'entrelacment : L'événement $AvtA$ est décomposé dans ce cas en $EvtAr_1 \mid\mid\mid EvtAr_2$ (tableau 5.12). Les événements $EvtAr_1$ et $EvtAr_2$ peuvent s'exécuter dans un ordre quelconque. Ils sont indépendants. Le contrôle n'est redonné à l'événement $EvtA$ de l'abstraction que si les deux événements sont exécutés.

Nous introduisons deux variants $VarInt_1$ et $VarInt_2$ pour contrôler l'exécution des deux événements $EvtAr_1$ et $EvtAr_2$ respectivement. Ils sont initialisés à la valeur 1 et l'exécution des événements les remet à 0 pour permettre l'exécution de l'événement $EvtA$ dont on a rajouté à sa garde l'expression $(VarInt_1 = 0 \wedge VarInt_2 = 0)$.

L'opérateur du parallélisme (synchrone) ne peut être codé en B car la sémantique du B événementiel ne le permet pas. En effet le B événementiel possède une sémantique asynchrone avec entrelacement. Ainsi nous le codons de la même manière que l'opérateur d'entrelacement.

REFINEMENT $M2$
REFINES $M1$
VARIABLES
$VarInt_1$, $VarInt_2$, $vari$, $varj$
INVARIANT
$J(vari, varj) \wedge VarInt_1 \in \{0,1\} \wedge VarInt_2 \in \{0,1\} \wedge J'(varj)$
ASSERTIONS
$G_0 \Rightarrow ((G_1 \wedge VarInt_1 = 1) \vee$
$(G_2 \wedge VarInt_2 = 1) \vee (G'_0 \wedge VarInt_1 = 0 \wedge VarInt_2 = 0))$
INITIALISATION
$VarInt1 := 1 \;||$
$VarInt2 := 1 \;||$
EVENTS

$EvtAr_1 =$	$EvtAr_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge$	$G_2 \wedge$	$G'_0 \wedge VarInt_1 = 0 \wedge$
$VarInt_1 = 1$	$VarInt_2 = 1$	$VarInt_2 = 0$
THEN	**THEN**	**THEN**
$S_1 \;\|\|$	$S_2 \;\|\|$	S'_0
$VarInt_1 := 0$	$VarInt_2 := 0$	**END** ;
END ;	**END** ;	

Tab. 5.12 – Modèle générique du raffinement en entrelacement

5.3.2.2 Décomposition concrète

Cette décomposition correspond, dans la grammaire du modèle générique, à la règle qui décompose les énoncés en événements de base.

$$Enonce ::= a; Enonce \mid a \;|||\; Enonce \mid a \;||\; Enonce \mid \delta \text{ avec } a \in A$$

A ce niveau les événements sont raffinés en les décomposant en utilisant les actions de base. Ce sont les raffinements du niveau énoncé. Chaque énoncé est raffiné en actions de base composées soit en séquentiel soit en parallèle selon les opérateurs utilisés.

Prefixage : Soit l'événement abstrait $EvtA$ décomposé en $EvtC1; EvtC2$ avec $EvtC1$ et $EvtC2$ deux événements concrets non décomposables. Le modèle générique du raffinement de l'événement $EvtA$ en $EvtC1; EvtC2$ est présenté sur le tableau 5.13. L'événement $EvtC1$ est de modalité m_i alors que l'événement $EvtC2$ est de modalité m_j.

Le variant $VarPre$ contrôle l'exécution des deux événements $EvtC_1$ et $EvtC_2$ de la même manière que le variant $VarSeq$ de l'opérateur de séquence. Les ensembles Ens_Modi et Ens_Modj définissent les actions produites, respectivement, par les modalités m_i et m_j. Les variables act_modi et act_modi sont deux variables qui définissent les actions de base de modalité respective m_i et m_j. Initialement, la

REFINEMENT $M2$
REFINE $M1$
SETS
$Ens\_Modi = \{act1\_mi, act2\_mi, ..., actn\_mi, rien\}$
$Ens\_Modj = \{act1\_mj, act2\_mj, ..., actm\_mj, rien\}$
VARIABLES
$Varpre$, act_modi, act_modj
INVARIANT
$act\_modi : Ens\_Modi$ and
$act\_modj : Ens\_Modj$
ASSERTIONS
⋮
INITIALISATION
$VarPre := 2$ ||
$act\_modi :\in Ens\_Modi$ ||
$act\_modj := rien$
EVENTS

$EvtC_1 =$	$EvtC_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge VarPre = 2$	$G_2 \wedge VarPre = 1$	G'_0
THEN	**THEN**	$\wedge VarPre = 0$
S_1 ||	S_2 ||	**THEN**
$act\_modi := rien$ ||	$act\_modi :\in Ens\_Modi$ ||	S'_0
$act\_modj :\in Ens\_Modj$ ||	$act\_modj := rien$ ||	**END** ;
$VarPre := 1$	$VarPre := 0$	
END ;	**END** ;	

Tab. 5.13 – Modèle générique du raffinement en préfixage

variable représentant les actions produites par la modalité m_i est initialisée à une action de l'ensemble Ens_Modi qui est la première action qui sera exécutée. La variable représentant les actions produites par la modalité m_j est initialisée à $rien$, car aucune action de modalité m_j n'est autorisée à s'exécuter initialement.

Quant l'événement Evc_1 est déclenché, la variable correspondant aux actions de modalité m_i est affectée par la valeur $rien$. La variable correspondant aux actions

produites par la modalité m_j est affectée par une valeur de l'ensemble des actions de modalité m_j ($act\_modj :\in Ens\_Modj$). Le même principe est adopté quand l'événement $EvtC_2$ est déclenché.

L'entrelacement : Soit l'événement abstrait $EvtA$ décomposé en $EvtC1$ ||| $EvtC2$ avec $EvtC_1$ et $EvtC_2$ deux événements concrets non décomposables. Le tableau 5.14 présente le modèle générique correspondant à ce raffinement. Les ensembles Ens_Modi et Ens_Modj ainsi que les variables act_modi et act_modj sont définis de la même manière que pour l'opérateur de préfixage présenté dans le paragraphe précédeent.

REFINEMENT $M2$
REFINES $M1$
SETS
$Ens\_Modi = \{act1\_mi, act2\_mi, ..., actn\_mi, rien\}$
$Ens\_Modj = \{act1\_mj, act2\_mj, ..., actm\_mj, rien\}$
VARIABLES
$VarInt_1$, $VarInt_2$, act_modi, act_modj
INVARIANT
$act\_modi : Ens\_Modi$ and
$act\_modj : Ens\_Modj$
ASSERTIONS
⋮
INITIALISATION
$VarInt_1 := 1 \;||\; VarInt_2 := 1 \;||$
$act\_modi := rien \;||\; act\_modj := rien$
EVENTS

$EvtC_1 =$	$EvtC_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge$	$G_2 \wedge$	$G'_0 \wedge$
$VarInt_1 = 1$	$VarInt_2 = 1$	$VarInt_1 = 0 \wedge$
THEN	**THEN**	$VarInt_2 = 0$
$S_1 \;\|\|$	$S_2 \;\|\|$	**THEN**
$act\_modi :\in Ens\_Modi \;\|\|$	$act\_modj :\in Ens\_Modj \;\|\|$	S'_0
$VarInt_1 := 0$	$VarInt_2 := 0$	**END** ;
END ;	**END** ;	

Tab. 5.14 – Modèle générique du raffinement en entrelacment

5.3.3 Expression des propriétés

Pour l'expression des propriétés, nous utilisons le modèle formel opérationnel que nous avons proposé dans la section 3.4.1. La tâche est conçue selon le modèle de la propriété à vérifier. La vérification dans ce cas, consiste à prouver que le modèle de la tâche est bien validé par le modèle du système. Ceci revient à vérifier que les événements du dernier raffinement du modèle de la tâche exprimée par le modèle de la propriété, sont des événements du système (l'IHM3 dans notre cas).

Afin de vérifier les propriétés liées aux modalités telles que les propriétés CARE, une abstraction du système de transitions est intéressante dans le sens où seules les modalités seront manipulées au lieu des actions. L'ensemble EnsMod énumère l'ensemble des modalités utilisées dans le système. La variable *mod* prend ses valeurs de cet ensemble et désigne la modalité utilisée pour chaque action.

Les ensembles des événements ($Ens\_Modi, Ens\_Modj, etc...$) seront remplacés par un seul ensemble de modalités ($EnsMod$). Les différentes variables des actions ($act\_modi, act\_modj, etc...$) seront remplacées par une seule variable représentant les modalités (Mod). Ainsi, le raffinement correspondant à l'opérateur du préfixage est remplacé par le modèle présenté sur le tableau 5.15.

Aucun des événements concrets ne peut être réalisé que si la variable *mod* contienne la modalité qui permet de le réaliser. Dans le cas de l'événement $EvtC_1$, on a renforcé sa garde par $mod = m_i$, car il est réalisé par la modalité m_i. Quand l'événement est réalisé, la modalité prend une valeur de l'ensemble des modalités du système. Ceci est réalisé par la substitution $mod :\in EnsMod$.

Avec le même principe de décomposition, les différents raffinements sont définis en suivant l'expression de la tâche à vérifier. C'est au dernier raffinement que l'ensemble $EnsMod$ est introduit.

REFINEMENT M2
REFINES M1
SETS
$EnsMod = \{m_1, m_2, ..., m_n, rien\}$
VARIABLES
Var, Mod
INVARIANT
$Mod : EnsMod$ and
⋮
ASSERTIONS
⋮
INITIALISATION
$Var := 2$
$Mod :\in EnsMod$
EVENTS

$EvtC_1 =$	$EvtC_2 =$	$EvtA =$
SELECT	**SELECT**	**SELECT**
$G_1 \wedge var = 2 \wedge$	$G_2 \wedge var = 1 \wedge$	$G'_0 \wedge var = 0$
$mod = m_i$	$mod = m_j$	**THEN**
THEN	**THEN**	S'_0
$S_1 \mid\mid$	$S_2 \mid\mid$	**END** ;
$mod :\in EnsMod \mid\mid$	$mod :\in EnsMod$	
$Var := 1$	$Var := 0$	
END ;	**END** ;	

Tab. 5.15 – Modèle générique du raffinement en préfixage après abstraction

Si les événements obtenus à la fin du dernier raffinement appartiennent aux événements de l'IHM3, nous pouvons conclure que la tâche conçue est prise en compte par le système. Ceci signifie qu'elle vérifie la propriété souhaitée, du moment qu'elle est conçue selon le modèle formel de la proprité, et elle est valide puisque elle est prise en compte par le système de l'IHM3.

Nous donnons, dans ce qui suit et pour chacune des propriétés CARE, le modèle B événementiel que nous proposons pour sa vérification.

5.3.3.1 Equivalence

Pour vérifier cette propriété nous raffinons l'événement Evt, correspondant à la tâche sur laquelle nous voulons vérifier la propriété, par $Evtmi[]Evtmj$. Les événements $Evtmi$ et $Evtmj$ sont réalisés, respectivement, avec les modalités m_i et

REFINEMENT Equivalence
REFINES Evtmimj
SETS
$EnsMod = \{m_1, m_2, ..., m_n\}$
VARIABLES
Mod
INVARIANTS
$Mod : EnsMod$ and
...
ASSERTIONS
...
INITIALISATION
$Mod :\in EnsMod$
EVENTS

$Evtmi_1 =$	$Evtmj_1 =$	$Evtmi_2 =$	$Evtmj_2 =$
SELECT	**SELECT**	**SELECT**	**SELECT**
$Gi1 \wedge$	$Gj1 \wedge$	$Gi2 \wedge$	$Gj2 \wedge$
$mod = m_i$	$mod = m_j$	$mod = m_i$	$mod = m_j$
THEN	**THEN**	**THEN**	**THEN**
$S_1 \parallel$	$S_1 \parallel$	$S_2 \parallel$	$S_2 \parallel$
$mod :\in EnsMod$	$mod :\in EnsMod$	$mod :\in EnsMod$	$mod :\in EnsMod$
END ;	**END** ;	**END** ;	**END** ;

.....

Tab. 5.16 – Modèle générique de la propriété d'équivalence

m_j. Le raffinement de $Evtmi$ aboutit au dernier niveau à des événements réalisés avec la modalité m_i, et le raffinement de $Evtmj$ aboutira au dernier niveau à des événements réalisés par la modalité m_j (tableau 5.16).

Les événements $Evtmi_k$ raffinent l'événement $Evtmi$ et les événements $Evtmj_l$ raffinent l'événement $Evtmj$. La variable mod est initialisée à une modalité quelconque du système ($mod :\in EnsMod$). Pour qu'un des événements $Evtmi_k$ soit déclenché, il faut que sa garde Gik soit vérifiée et que la variable mod contienne la valeur m_i qui permet de le réaliser. Le même principe est appliqué pour les événements $Evtmj_k$.

5.3.3.2 Complémentarité

De la même manière que les propriétés précédentes, la propriété de complémentarité est vérifiée par la tâche correspondant à un évènement abstrait Evt, si

nous trouvons une décomposition, selon le modèle formel de la propriété, où les évènements du dernier raffinement correspondent à des évènements du système. Nous utilisons toujours le même principe pour la représentation des modalités. On ajoute $mod = m_i$ à la garde des chacun des évènements de modalité m_i. On

REFINEMENT complementarite
REFINES Evt
SETS
$EnsMod = \{m_1, m_2, ..., m_n, rien\}$
VARIABLES
Mod, fin_tache, $Util\_m_i$, $Util\_m_j$
INVARIANT
mod : EnsMod and
fin_tache : BOOL and
$Util\_m_i$: BOOL and
$Util\_m_j$: BOOL and
$(fin\_tache = TRUE) => (Util\_m_i = TRUE \wedge Util\_m_j = TRUE)$
ASSERTIONS
⋮
INITIALISATION
$mod :\in EnsMod$ ||
$Util\_m_i := FALSE$ ||
$Util\_m_j := FALSE$
EVENTS

$Evtk\_mi =$	$Evtl\_mj =$
SELECT	**SELECT**
$Gki \wedge mod = m_i$	$Gkj \wedge mod = m_j$
THEN	**THEN**
Si_1 || $mod :\in EnsMod$ ||	Sj_1 || $mod :\in EnsMod$ ||
$Util\_m_i := TRUE$	$Util\_m_j := TRUE$
END ;	**END** ;

.....

Tab. 5.17 – Modèle générique de la propriété de complémentarité

peut envisager de vérifier cette propriété sans passer par le modèle formel opérationnel de la propriété (tableau 5.17). Ceci est réalisé en associant une variable indiquant l'utilisation de chaque modalité ($Util\_m_i$). Cette variable est initialisée à $FALSE$ au départ. Ensuite, elle est mise à $TRUE$ à chaque utilisation de la modalité correspondante (m_i). La vérification de la propriété de complémentarité est vérifiée si chacune de ces variables est égale à $TRUE$ à la fin de la tâche

$((fin\_tache = TRUE) => (Util\_m_i = TRUE \wedge Util\_m_j = TRUE))$. Ceci garantit l'utilisation au moins une fois de chacune des modalités et c'est ce qui définit la propriété de complémentarité.

Les événements $Evtk\_mi$ sont des événements réalisés par la modalité m_i. Pour cela, nous renforçons les gardes de ces événements par $mod = m_i$. Quand un de ces événements est réalisé, la variable *mod* prend une valeur de l'ensemble des modalités du système ($mod :\in EnsMod$) et la variable correspondant à l'utilisation de la modalité m_i est mise à *true* ($Util\_m_i := TRUE$). Le même principe est adopté pour les événements $Evtl\_mj$ réalisés avec la modalité m_j.

5.3.3.3 Assignation

Le tableau 5.18 présente le modèle générique de cette propriété. Pour exprimer l'assignation de la modalité m_i à une tâche donnée, il faut ajouté $mod = m_i$ aux gardes de tous les événements ($Evtx$) de son dernier raffinement.

REFINEMENT Assignation
REFINES tâche
SETS
$EnsMod = \{m_1, m_2, \cdots, m_n\}$
VARIABLES
mod
INVARIANT
$mod : EnsMod$
ASSERTIONS
...
INITIALISATION
$mod :\in EnsMod$
EVENTS
$Evtx =$
SELECT
$G_x \wedge mod = m_i$
THEN
$S_x \ ||$
$mod :\in EnsMod$
END ;
.....

Tab. 5.18 – Modèle générique de la propriété d'assignation

5.3.3.4 Redondance

Pour vérifier que deux modalités sont redondantes pour la réalisation d'une tâche (événement abstrait), il faut trouver un raffinement à cet événement selon le modèle opérationnel de la propriété de redondance présenté dans la section 3.4.1. Si l'événement représentant la tâche à vérifier est noté Evt alors il faut trouver un raffinement qui décompose Evt en $(Evtm_i[]Evtm_j[](Evtm_i|||Evntm_j))$ tel que la modalité m_i est assignée à l'événement $Evtm_i$ et la modalité m_j est assignée à l'événement $Evtm_j$. Un premier raffinement est réalisé selon les modèles génériques correspondant aux opérateurs de choix et d'entrelacement. Ensuite, chacun des événements $Evtm_i$ et $Evtm_j$ est raffiné selon le modèle de la propriété d'assignation présenté précédement. Nous avons choisi de donner les différents raffinements de cette propriété sur l'exemple que nous présentons dans la section 5.3.5.

5.3.4 Preuve de propriétés

La preuve des propriétés est effectuée d'une manière opérationnelle en encodant la tâche selon le modèle de la propriété en utilisant les modèles B événementiel génériques de la propriété présentés dans la section précédente. Le système de l'IHM3 est conçu par raffinements successifs en utilisant les modèles B événementiel génériques de chaque opérateur de décomposition. La validation de la tâche par le système de l'IHM3 est effectuée en vérifiant que les événements de base obtenus lors du dernier raffinement du modèle de la tâche appartiennent aux événements du dernier raffinement du système de l'IHM3.

5.3.5 Application à l'étude de cas

Afin, d'illustrer notre mise en oeuvre, nous avons appliqué notre modélisation sur une étude de cas dans le but de montrer l'applicabilité de notre démarche. Il s'agit de la tâche qui permet de remplir le champ texte par la ville *Boston* dans le système *Matis*, utilisée comme étude de cas dans notre mise en oeuvre dans le chapitre précédent. Lors de la réalisation de cette tâche, les modalités *parole* et *manipulation directe* sont utilisées de manière redondante. L'expression de la tâche est donnée comme suit :

$$\boxed{\begin{array}{l}(('From';' Boston';)[](ClicFrom;ClicBoston) \\ [](('From';' Boston';)||(ClicFrom;ClicBoston)))\end{array}}$$

avec $'From'$, $'Boston'$, $ClicFrom$ et $ClicBoston$ des actions interactives de base définies dans la section 4.3.3.

L'arbre de décomposition de ce système est donné sur la figure 5.2.

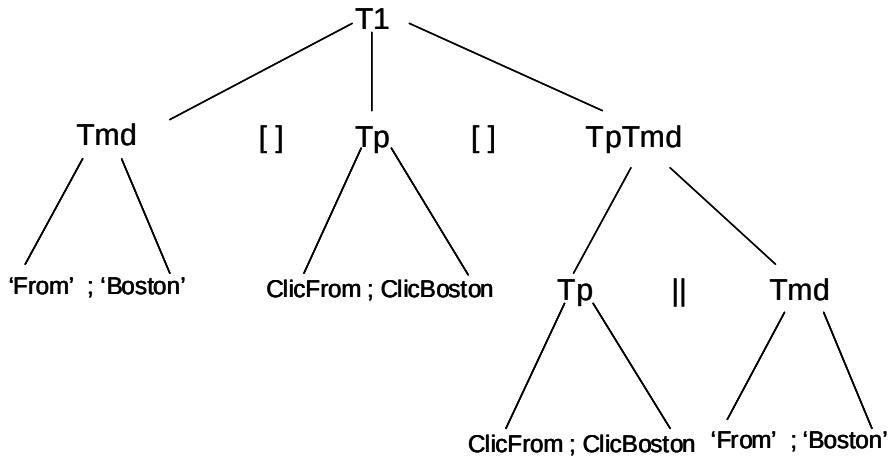

Fig. 5.2 – Arbre de décomposition du système

Les tâches $T1$, Tmd, Tp et $TpTmd$ sont des tâches abstraites.
La décompostion hiérarchique de la tâche $T1$ peut être décrite comme suit :

$$\begin{array}{lcl} T1 & = & Tmd \,[]\, Tp \,[]\, TpTmd \\ TpTmd & = & Tp \,||\, Tmd \\ Tp & = & 'From' \,;\, 'Boston' \\ Tmd & = & ClicFrom \,;\, ClicBoston \end{array}$$

Trois niveaux de raffinements abstraits correspondant aux niveaux hiérarchiques dans l'arbre de décomposition de cette tâche sont nécessaires. Le premier modèle décrit la tâche $T1$. Le premier raffinement décrit la décomposition $Tmd[]Tp[]TpTmd$. Le deuxième raffinement décrit la décomposition de $TpTmd$ par $Tp||Tmd$. Finalement le dernier raffinement consiste à décomposer les tâches abstraites par les événements concrets en raffinant l'événement Tp en $('From' \,;\, 'Boston')$ et l'événement Tmd en $(ClicFrom \,;\, ClicBoston)$.

MODEL IHM
VARIABLES
$T1$
INVARIANT
$T1 : BOOL$
INITIALISATION
$T1 := FALSE$
EVENTS
$EvtT1 =$
SELECT
G_0
BEGIN
$T1 := TRUE$
END

Tab. 5.19 – Modèle B de la tâche T1

Premier modèle en B : Il s'agit de la modélisation de la tâche du plus haut niveau $T1$. Elle est donnée sur le tableau 5.19. Il y'a un seul événement $EvtT1$ et la substitution $T1 := TRUE$ signifie que la tâche $T1$ est réalisée.

Premier raffinement : Le premier niveau de raffinement concerne la décomposition de la tâche $T1$ en $Tmd[]Tp[]TpTmd$. Le raffinement correspondant est obtenu en appliquant la règle de décomposition de l'opérateur du choix. Sa modélisation B événementiel est donnée sur le tableau 5.20. Le modèle est enrichi par de nouveaux événements : $EvtTp$, $EvtTmd$ et $EvtTpTmd$ correspondant respectivement aux tâches abstraites Tp, Tmd et $TpTmd$. Les substitutions $Tp := TRUE$, $Tmd := TRUE$ et $TpTmd := TRUE$ désignent respectivement que les tâches abstraites TP, Tmd et $TpTmd$ sont réalisées.

Le variant $VarChoix$ est initialisé aléatoirement à une des valeurs 1, 2 ou 3. Chacun des événements $EvtTp$, $EvtTmd$ et $EvtTpTmd$ le rend à la valeur 0 pour permettre à l'événement $EvtT1$ correspondant à la tâche $T1$ de se réaliser.

REFINEMENT $raffinement1$
REFINES IHM
VARIABLES
$VarChoix$, Tp, Tmd, $TpTmd$, $\cdots$
INVARIANT
$VarChoix \in \{0,1,2,3\} \wedge Tp : BOOL \wedge Tmd : BOOL \wedge TpTmd : BOOL \wedge \cdots$
ASSERTIONS
$\ldots$
INITIALISATION
$VarChoix := \{1,2,3\} \;||\; \cdots$
EVENTS

$EvtTp =$
SELECT
$G1 \wedge VarChoix = 1$
THEN
$Tp := TRUE \;||$
$VarChoix := 0$
END;

$EvtTmd =$
SELECT
$G2 \wedge VarChoix = 2$
THEN
$Tmd := TRUE \;||$
$VarChoix := 0$
END;

$EvtTpTmd =$
SELECT
$G3 \wedge VarChoix = 3$
THEN
$TpTmd := TRUE \;||$
$VarChoix := 0$
END;

$EvtT1 =$
SELECT
$G'_0 \wedge VarChoix = 0$
THEN
$T1 := TRUE$
END;

Tab. 5.20 – Modèle du premier raffinement

Deuxième raffinement : Ce raffinement correspond à la décompostition de la tâche abstraite $TpTmd$ en $Tp||Tmd$. Il est obtenu en appliquant la règle correspondant au raffinement de l'opérateur du parallèle. A ce niveau de raffinement $T1 = Tp[]Tmd[](Tp||Tmd)$. Le modèle B événementiel correspondant est donné sur les tableaux 5.21 et 5.22.

EvtTpp et *EvtTmdd* correspondent respectivement aux tâches abstraites Tp et Tmd qui raffinent la tâche abstraite *TpTmd*. Les variants $VarInt1$ et $VarInt2$ sont initialisés chacun à 1. La réalisation de l'événement *EvtTpp* rend le variant $VarInt1$ à 0 et la réalisation de l'événement *EvtTmdd* rend le variant $VarInt2$ à 0. Après l'exécution des deux événements *EvtTpp* et *EvtTmdd* la réalisation de l'événement abstrait *EvtTpTmd* est possible. Les événements $EvtT1$, *EvtTp* et *EvtTmd* restent inchangés.

REFINEMENT $raffinement2$
REFINES $raffinement1$
VARIABLES
$VarInt_1, VarInt_2, \cdots$
INVARIANT
$VarInt_1 \in \{0,1\} \wedge VarInt_2 \in \{0,1\} \wedge \cdots$
ASSERTIONS
...
INITIALISATION
$VarInt1 := 1 \parallel VarInt2 := 1 \parallel$
...

Tab. 5.21 – Modèle du deuxième raffinement : 1

EVENTS

$EvtTpp =$	$EvtTmdd =$	$EvtTpTmd =$
SELECT	**SELECT**	**SELECT**
$G4 \wedge$	$G5 \wedge$	$G'3 \wedge VarInt_1 = 0 \wedge$
$VarInt_1 = 1$	$VarInt_2 = 1$	$VarInt_2 = 0$
THEN	**THEN**	**THEN**
$Tp := TRUE \parallel$	$Tmd := TRUE \parallel$	$TpTmd := TRUE$
$VarInt_1 := 0$	$VarInt_2 := 0$	**END** ;
END ;	**END** ;	
$EvtT1 =$	$EvtTp =$	$EvtTmd =$
...	...	...

Tab. 5.22 – Modèle du deuxième raffinement : 2

Troisième raffinement Ce troisième raffinement consiste à décomposer les événements abstraits *EvtTp* et *EvtTpp* correspondant à la tâche abstraite TP et les événe-

ments $EvtTmd$ et $EvtTmdd$ correspondant à la tâche Tmd en événements concrets tels que $('From'\,;'Boston')$ décompose Tp et Tpp, et $(ClickFrom\,;ClicBoston)$ décompose Tmd et $Tmdd$. A ce niveau de raffinement $T1 = ('From';' Boston')[]('From'$ $;' Boston')[](('From';' Boston')||('From';' Boston'))$. Ce raffinement est obtenu en appliquant la règle de l'opérateur de préfixage. Les tableaux 5.23, 5.24, 5.25, 5.26

$REFINEMENT\ raffinement3$
REFINES $raffinement2$
SETS
$EnsMod = \{parole, manipD, rien\}$
VARIABLES
$VarPre1$, $VarPre2$, $VarPre3$, $VarPre4$, Mod
INVARIANT
$Mod : EnsMod$ and
⋮
ASSERTIONS
⋮
INITIALISATION
$VarPre1 := 2 \parallel VarPre2 := 2 \parallel VarPre3 := 2 \parallel VarPre4 := 2 \parallel$
$Mod :\in EnsMod$

Tab. 5.23 – Modèle du troisième raffinement

et 5.27 montrent le modèle B événementiel correspondant. Les événements $EvtT1$ et $EvtTpTmd$ restent inchangés.

EVENTS

$EvtC1_{Tp} =$	$EvtC2_{Tp} =$	$EvtTp =$
SELECT	**SELECT**	**SELECT**
$G_6 \wedge varpre1 = 2 \wedge$	$G_7 \wedge varpre1 = 1 \wedge$	$G'_1 \wedge varpre1 = 0$
$mod = parole$	$mod = parole$	**THEN**
THEN	**THEN**	$Tp := TRUE$
$S_1 \parallel$	$S_2 \parallel$	**END** ;
$Varpre1 := 1$	$Varpre1 := 0$	
$mod :\in EnsMod$	$mod :\in EnsMod$	
END ;	**END** ;	

Tab. 5.24 – Modèle du troisième raffinement : événement Tp

$EvtC1_{Tmd} =$	$EvtC2_{Tmd} =$	$EvtTmd =$
SELECT	**SELECT**	**SELECT**
$G_8 \wedge varpre2 = 2 \wedge$	$G_9 \wedge varpre2 = 1 \wedge$	$G'_2 \wedge varpre2 = 0$
$mod = manipD$	$mod = manipD$	**THEN**
THEN	**THEN**	$Tmd := TRUE$
$S_1 \parallel$	$S_2 \parallel$	**END** ;
$Varpre2 := 1 \parallel$	$Varpre2 := 0 \parallel$	
$mod :\in EnsMod$	$mod :\in EnsMod$	
END ;	**END** ;	

Tab. 5.25 – Modèle du troisième raffinement : événement Tmd

Nous avons utilisé les modèles génériques de décomposition de la propriété de redondance afin de raffiner cette tâche. C'est au dernier raffinement que nous utilisons les modalités qui permettent de réaliser les événements concrets. On retrouve la même structure des événements de la propriété d'assignation pour les tâches Tp Tpp, Tmd et $Tmdd$.

$EvtC1_{Tmpp} =$	$EvtC2_{Tpp} =$	$EvtTpp =$
SELECT	**SELECT**	**SELECT**
$G_10 \wedge varpre3 = 2 \wedge$	$G_11 \wedge varpre3 = 1 \wedge$	$G'_4 \wedge varpre3 = 0$
$mod = parole$	$mod = parole$	**THEN**
THEN	**THEN**	$Tpp := TRUE$
$S_1 \parallel$	$S_2 \parallel$	**END** ;
$Varpre3 := 1$	$Varpre3 := 0$	
$mod :\in EnsMod$	$mod :\in EnsMod$	
END ;	**END** ;	

Tab. 5.26 – Modèle du troisième raffinement : événement Tpp

$EvtC1_{Tmdd} =$	$EvtC2_{Tmdd} =$	$EvtTmdd =$
SELECT	**SELECT**	**SELECT**
$G_1 2 \wedge varpre4 = 2 \wedge$	$G_1 3 \wedge varpre4 = 1 \wedge$	$G'_5 \wedge varpre4 = 0$
$mod = manipD$	$mod = manipD$	**THEN**
THEN	**THEN**	$Tmdd := TRUE$
$S_1 \parallel$	$S_2 \parallel$	**END** ;
$mod :\in EnsMod$	$mod :\in EnsMod$	
$Varpre4 := 1$	$Varpre4 := 0$	
END ;	**END** ;	
$EvtT1 =$	$EvtTpTmd =$	
...	...	

Tab. 5.27 – Modèle du troisième raffinement : événement Tmdd

5.4 Conclusion

Dans ce chapitre, nous avons montré l'utilisation du développement par raffinement ainsi que la vérification par la preuve des propriétés des IHM multimodales. La représentation des propriétés à l'aide des systèmes de transitions a permis la représentation de toutes les propriétés CARE en B événementiel.

La méthode de vérification utilisée avec B a permis non seulement de vérifier que la tâche concernée vérifie la propriété souhaitée mais qu'elle est valide.

Contrairement aux langages de SMV et Promela adaptés à l'expression des systèmes de transitions, le langage B nécessite la gestion explicite des systèmes de transitions en introduisant les variants. L'inconvénient que nous pouvons citer est la charge du système avec l'ensemble de ces variants. Dès que le système de transitions atteint une certaine taille, la gestion de ces variables devient complexe. Par contre, contrairement à SMV et Promela, B événementiel permet de traiter des systèmes de transitions de taille arbitraire et ne se heurte donc pas au problème de l'explosion combinatoire, mais la preuve n'est que semi-automatique.

Chapitre 6

Conclusion

6.1 Conclusion

Notre travail est motivé par la complexité de modélisation de l'interaction multimodale. Le développement dans ce domaine utilise des modèles informels voire semi-formels ce qui ne permet pas souvent de répondre a priori à des questions concernant les propriétés du système. En général, il faut attendre la fin du développement du système pour pouvoir tester et expérimenter le système interactif ainsi conçu. Ceci, impose souvent un retour aux premières étapes de la conception et induit un développement coûteux.

Après une étude bibliographique sur les travaux réalisés par la communauté des IHM3 pour la modélisation des systèmes interactifs multimodaux et leurs propriétés, nous nous sommes concentrés sur deux points. D'une part la définition d'un cadre méthodologique pour la modélisation de l'interaction multimodale et des propriétés d'utilisabilité et d'autre part la mise en oeuvre de différentes techniques formelles pour l'expression et la validation des modèles et des propriétés. Tout au long de ce travail nous nous sommes attachés à ne pas définir de nouveaux modèles mais plutôt à réutiliser les travaux réalisés dans le domaine de l'interaction multimodale comme source et origine de notre démarche de formalisation.

6.1.1 Méthodologie

L'étude bibliographique menée autour des modèles formels pour le développement des systèmes multimodaux, nous a permis de découvrir que tous les travaux menés dans ce domaine utilisent plusieurs modèles formels différents selon différents points de vue (sémantique, systèmes de preuve, mode de validation, etc.) et dont le domaine d'efficience est limité. En conséquence, il est souvent nécessaire de mettre en oeuvre différentes techniques pour atteindre différents objectifs. Ce type de démarche engendre une hétérogénéité des modèles et il n'est pas possible de lier les résultats obtenus sur des modélisations formelles différentes d'un même système in-

teractif multimodal. Ce problème n'est pas propre aux systèmes interactifs, mais est classique en génie logiciel.

Nous avons traité ce problème par la proposition d'un cadre général à sémantique formelle permettant de décrire les systèmes interactifs multimodaux en se focalisant sur la modalité en entrée.

6.1.1.1 Modélisation de systèmes interactifs multimodaux

La modélisation des systèmes interactifs multimodaux que nous avons définie est formelle et générique. Nous proposons un cadre unique de modélisation formelle pour la conception de l'interaction d'une part et pour la spécification des propriétés d'autre part. Ce cadre de modélisation se fonde sur les systèmes de transitions avec une sémantique formelle exprimée par des règles de sémantique opérationnelle et peut être translaté dans une technique formelle particulière utilisant ce modèle formel comme sémantique pour le comportement du système interactif à développer. Les modèles issus de cette approche sont indépendants de toute technique formelle.

L'originalité du cadre méthodologique générique que nous avons proposé réside dans la prise en compte du savoir-faire des développeurs de systèmes interactifs. En effet, le cadre méthodologique est fortement inspiré des espaces de conception proposés dans la littérature. Ce cadre permet la représentation de différents espaces de conception et donc de prendre en compte différentes points de vue lors de la conception d'une IHM3. En effet, nous avons proposé une conception de l'interaction mutli-modale en respectant les types de multimodalité définis par [Bellik, 1995]. Ces types définissent les contraintes sur l'utilisation des modalités du côté du système. Nous avons illustré ce travail sur une étude de cas non triviale fournie par le CLIPS/IMAG.

6.1.1.2 Expression et Vérification de propriétés

Par ailleurs, les contraintes associées aux utilisateurs sont définies par les propriétés CARE. Ces dernières ont également été modélisées dans le cadre méthodologique générique que nous avons proposé. Ces propriétés permettent d'évaluer l'utilisation qui peut être faite d'un système modélisé selon des systèmes de transitions. Nous notons que ces propriétés sont décrites dans le même cadre formel ce qui a pour conséquence de réduire l'hétérogénéité que l'on constate dans les autres modélisations formelles.

L'expression des propriétés CARE peut être effectuée de deux manières. La première dite opérationnelle consiste à exprimer un comportement attendu et à vérifier que celui-ci est bien compris dans l'ensemble des comportements du système. En d'autres termes, il faut exprimer une propriété par un système de transitions, et observer si le système de transitions associé au système global comprend bien le système de transitions de la propriété. La seconde approche, dite logique, consiste

à exprimer une propriété par une formule de logique (invariant, propriété exprimée en logique temporelle, etc.) et prouver que cette propriété est bien une propriété du système décrivant le système interactif multimodal.

Un autre avantage de ce cadre méthodologique est l'orthogonalité entre espaces de conception et propriétés (entre autres propriétés CARE). Il est possible de décrire les espaces de conception associés à un système multimodal indépendamment des propriétés qui sont vérifiées sur ce système. Cette orthogonalité permet la prise en compte de différentes techniques de conception et de différentes catégories de propriétés.

6.1.2 Mise en oeuvre de techniques formelles

La seconde partie de notre travail a consisté à formaliser des modèles issus du cadre générique en utilisant des techniques formelles différentes. Nous avons utilisé, avec les mêmes études de cas, trois techniques formelles : deux techniques fondées sur la vérification sur modèle et une technique fondée sur la preuve.

6.1.3 Techniques de vérification sur modèles ou model checking

Deux techniques de vérification sur modèle ont été utilisées. La première utilise une logique temporelle linéaire et l'autre une logique temporelle arborescente.

6.1.3.1 Modélisation

Dans les deux techniques, nous avons transcrit les systèmes de transitions issus du cadre méthodologique générique sans difficulté particulière malgré la différence des modélisations associées à chaque technique. L'une (SMV) utilise les états et les variables d'états pour décrire un système alors que l'autre (PROMELA) exprime explicitement les actions qui font transiter d'un état à un autre. Dans les deux cas, les modélisations ont été réalisées en totalité et tous les éléments issus du cadre méthodologique générique ont pu être représentés.

6.1.3.2 Expression et vérification de propriétés

Pour ce qui est de la vérification des propriétés dans les techniques de vérification sur modèle, les model-checkers associés aux techniques utilisées, n'ont pas posé de problème particulier pour ce qui est de la procédure de vérification. Par contre, l'expression des propriétés CARE n'a pas été complète. La propriété de redondance n'a pu être exprimée dans les deux cas. Cela est dû à l'impossibilité de décrire, à l'aide de ces techniques, une composition parallèle qui soit synchrone telle que l'exige notre expression de la propriété de redondance dans le cadre méthodologique

générique. Les techniques de model checking fondées sur les langages à flots de données synchrones devraient être évalués pour étudier la possibilité de traiter la propriété de redondance.

Nous n'avons pas mesuré la complexité des vérifications de propriétés ni l'impact des modélisations sur cette complexité. Par contre, l'abstraction que nous avons effectuée en n'utilisant que les variables exprimant les changements de modalités a réduit la taille des systèmes de transitions sur lesquels portait la vérification.

6.1.3.3 Techniques à base de preuve formelle

La technique formelle, fondée sur la preuve, utilisée est la méthode " B événementiel ". Cette technique consiste à exprimer le système par un ensemble d'événements gardés décrivant des changements d'états.

6.1.3.4 Modélisation

Contrairement aux techniques de vérification sur modèle où nous avons exprimé la totalité du système directement, la technique B événementiel permet de décrire le système pas à pas grâce à l'opération de raffinement qu'il fournit. Nous avons utilisé le travail de [Ait-Ameur & Baron, 2007]. Les opérateurs permettant de décrire l'enchaînement des actions du système sont utilisés pour décrire les raffinements de modèles B. En résultat, nous avons été en mesure d'exprimer la totalité des descriptions des systèmes interactifs sur plusieurs études de cas.

Par contre nous n'avons pas étudié l'impact de ce choix de méthode de raffinement par rapport à la difficulté de preuve des obligations de preuve générées. En fait, on peut se demander si d'autres décompositions, moins intuitives, ne permettraient pas d'obtenir des obligations de preuve dont la preuve serait moins coûteuse.

6.1.3.5 Expression et vérification de propriétés

L'expression des propriétés de systèmes multimodaux décrits avec B événementiel n'est pas dissociée de l'activité de modélisation du système lui même. Des propriétés de non blocage ainsi que des invariants sont décrits dans les clauses spécifiques de B. Par contre la description de propriétés est souvent effectuée de manière opérationnelle par la description de tâches encodant une propriété et ensuite le développeur s'assure que cette tâche est bien une tâche du système. De cette façon, nous avons été en mesure d'exprimer et de prouver formellement la totalité des propriétés CARE.

6.2 Perspectives

De la même manière que pour les résultats obtenus, les perspectives entrevues par nos travaux portent également sur la méthodologie et sur la formalisation de

modèles de systèmes interactifs multimodaux ainsi que la validation de propriétés de ces systèmes. Avant de rentrer dans les détails de ces perspectives, il nous paraît indispensable de valider cette proposition

- d'une part en utilisant d'autres études de cas non triviales décrites dans d'autres espaces de conception ou avec d'autres notations spécifiques au développement d'IHM3 et
- d'autre part en utilisant d'autres techniques formelles aussi bien pour la modélisation du système que pour la vérification des propriétés associées.

Une évaluation de l'efficacité de cette démarche pourra être ensuite réalisée.

6.2.1 Méthodologie

Le cadre méthodologique proposé a permis de montrer que la prise en compte de notations issues de concepteurs d'IHM et leur formalisation étaient du domaine du possible. Seulement, ce travail est insuffisant pour couvrir le cycle de développement d'une IHM3. Sur le plan méthodologique, des travaux en direction des points suivants devront être menés afin de compléter notre proposition. Nous nous y attèlerons à l'avenir.

- **Décomposition / Composition.** Ce point concerne la description et la construction du système. Ce dernier peut être construit par des opérations de composition (conception ascendante) en assemblant des systèmes de transitions pour en faire de plus gros, ou bien en décomposant un système de transitions pour obtenir plusieurs autres systèmes. Ces opérateurs de décomposition (conception descendante) pourraient enrichir le cadre méthodologique formel proposé. Il faudra également veiller à y associer les conditions ou obligations de preuve qui garantissent la correction de cette composition / décomposition. Ensuite, on se posera également la question de la preuve de propriétés. Peut-on établir des propriétés localement à un composant de système de transitions et garantir que ces propriétés sont conservées par le système global ;
- **Architecture logicielle.** Le travail effectué dans cette thèse ne prend pas en compte d'architecture logicielle particulière. Il nous paraît nécessaire de relier ce cadre méthodologique à l'architecture logicielle. Un modèle d'interacteurs avec lequel notre modèle d'interation est attaché aux entrées, permet d'avoir des composants logiciels valides. La définition de règles formelles pour la composition des interacteurs permettra un developpement plus rigoureux et une réutilisabilité de composants déja validés ;
- **Validation de tâches.** A l'image des architectures logicielles, la validation de tâches utilisateurs n'a pas été prise en compte. Il est indispensable de compléter le cadre méthodologique par la proposition de l'expression de modèles de tâches. Ce travail ne nous paraît pas complexe au regard des opérateurs utilisés pour décrire les systèmes de transitions du cadre méthodologique proposé ;

- **Lien entre les modèles CASE et CARE.** Une étude formelle du lien entre l'espace de conception CASE et les propriétés CARE permet de déduire, dès la conception, les propriétés CARE que l'IMH3 peut satisfaire et permet ainsi de donner un autre cadre formel de conception centré utilisateur.
- **Réutilisation et composants génériques.** Ce point concerne la réutilisation de développement. Par exemple, la conception d'une interaction en mode alterné devrait être réutilisable dans différents contextes. Par réutilisation, nous entendons non seulement la description du système, mais également les propriétés prouvées sur un tel système. Ces composants génériques pourraient être raccrochés à des composants génériques existants. Nous pensons par exemple que ces modèles pourraient constituer la spécification formelle de composants génériques de la plate-forme ICARE ou d'éléments de boîtes à outils multimodales ;
- **Prise en compte de la modalité en sortie.** Enfin, nous n'avons pris en compte que la multimodalité en entrée. Il nous paraît indispensable de développer le même effort pour décrire la multimodalité en sortie afin de pouvoir exprimer la totalité d'une interaction multimodale.

6.2.2 Utilisation de techniques formelles

La mise en oeuvre de techniques formelles que nous avons effectuée a démontré la faisabilité de notre démarche tant avec des techniques de vérification sur modèle qu'avec des techniques formelles fondées sur la preuve. Cependant, ce travail n'est pas terminé. Il est nécessaire d'approfondir différents points importants dans cette utilisation de techniques formelles.

- **Expression de propriétés CARE.** Les différentes techniques utilisées n'ont pas été en mesure d'exprimer toutes les propriétés de la famille CARE. Il est nécessaire de faire un bilan autour des techniques formelles en vue d'indiquer quelles caractéristiques elles doivent satisfaire afin de supporter cette vérification. Cela permettra de guider le concepteur dans le choix de la technique à mettre en oeuvre ;
- **Utilisation de techniques d'abstraction.** La définition d'autres sources d'abstraction permettra une simplification des procédures de vérification de propriétés à l'image de ce que nous avons réalisé avec l'utilisation des variables de modalité ;
- **Génération de code.** Les travaux que nous avons réalisés sont incomplets de ce point de vue. Il faudrait effectivement traiter la génération de codes et de tests associés en utilisant des techniques permettant de telles générations ;
- **Coopération de techniques formelles hétérogènes.** Les différentes techniques utilisées sont hétérogènes en sémantique et en pouvoir d'expression de propriétés. Le cadre générique unifie les descriptions, mais nous ne pouvons

pas attester que les propriétés prouvées par une des techniques sont également valides dans l'expression d'une autre technique formelle. Il est nécessaire de définir des relations formelles entre le cadre méthodologique générique proposé et les représentations formelles que nous en faisons dans telle ou telle autre technique formelle.

Bibliographie

[Abowd *et al.*, 1995] Abowd G., Wang H. & Monk A. (1995). A formal technique for automated dialogue development. In *DIS '95 : Proceedings of the conference on Designing interactive systems*, p. 219–226, New York, USA : ACM Press.

[Abrial, 1996a] Abrial J.-R. (1996a). *The B Book : Assigning Programs to Meanings*. Cambridge University Press - ISBN 0521-496195.

[Abrial, 1996b] Abrial J.-R. (1996b). Extending b without changing it (for developing distributed systems). In H. Habrias, Ed., *First B Conference, Putting Into Pratice Methods and Tools for Information System Design*, p.$\tilde{2}$1, Nantes, France.

[Ait-Ameur,] Ait-Ameur Y. Cooperation of formal methods in an engineering based software developpement process. In *Integrated Formal Methods : Second International Conference, IFM 2000*, volume 1945/2000, Dagstuhl Castler, Germany : LNCS Springer Berlin/Heidelberg.

[Ait-Ameur *et al.*, 2006a] Ait-Ameur Y., Ait-Sadoune I. & Baron M. (2006a). Etude et comparaison de scénarios de développements formels d'interfaces multi-modales fondés sur la preuve et le raffinement. In *MOSIM 2006 - 6ème Conférence Francophone de Modélisation et Simulation. Modélisation, Optimisation et Simulation des Systèmes : Défis et Opportunités*, Rabat.

[Ait-Ameur *et al.*, 2006b] Ait-Ameur Y., Ait-Sadoune I., Baron M. & Mota J. (2006b). Validation et vérification formelles de systèmes interactifs multimodaux fondées sur la preuve. In *18° Conférence Francophone sur l'Interaction Homme-Machine (IHM)*, p. 123–130, Montréal.

[Ait-Ameur & Baron, 2007] Ait-Ameur Y. & Baron M. (2007). Formal and experimental validation approaches in HCI systems design based on a shared event b model. *International Journal on Software Tools and Technology Transfer (STTT)*, **8**(2), 1–17.

[Ait-Ameur *et al.*, 2003] Ait-Ameur Y., Baron M. & Kamel N. (2003). Utilisation de techniques formelles dans la modélisation d'Interfaces Homme-Machine. une expérience comparative entre b et promela/spin. In *6th International Symposium on Programming and Systems ISPS 2003*, p. 57–66, Alger.

[Ait-Ameur *et al.*, 2005] Ait-Ameur Y., Baron M. & Kamel N. (2005). Encoding a process algebra using the event B method. Application to the validation of user interfaces. In *IEEE ISOLA workshop on leveraging applications of formal methods, verification, and validation*, p. 109–125, Loyola College Graduate Center, Columbia, USA.

[Ait-Ameur *et al.*, 1998a] Ait-Ameur Y., Girard P. & Jambon F. (1998a). A Uniform approach for the Specification and Design of Interactive Systems : the B method. In P. Markopoulos & P. Johnson, Eds., *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'98)*, p. 333–352, Abingdon, UK.

[Ait-Ameur *et al.*, 1998b] Ait-Ameur Y., Girard P. & Jambon F. (1998b). Using the B formal approach for incremental specification design of interactive systems. In S. Chatty & P. Dewan, Eds., *Engineering for Human-Computer Interaction*, volume 22, p. 91–108 : Kluwer Academic Publishers.

[Ait-Ameur & Kamel, 2004] Ait-Ameur Y. & Kamel N. (2004). A generic formal specification of fusion of modalities in a multimodal HCI. In R. Jacquart, Ed., *IFIP World Computer Science*, p. 415–420, Toulouse, France : Kluwer Academics.

[Ait-Sadoune, 2005] Ait-Sadoune I. (2005). *Vérification et validation formelle dIHM Multimodales fondées sur la preuve. Utilisation de la Méthode B.* Mémoire d'ingénieur d'état en informatique, INI, Alger.

[Alty, 1991] Alty J. (1991). Multimedia - what is it and how do we exploit it ? In D. Diaper & N. Hammond, Eds., *Acte de la conférence Human-Computer Interaction*, p. 31–44 : People and Computers VI.

[Balbo, 1994] Balbo S. (1994). *Evaluation ergonomique des interfaces utilisateur : un pas vers l'automatisation.* Doctorat d'université, Université Josef Fourier.

[Baron, 2003] Baron M. (2003). *Vers une approche sûre du développement des Interfaces Homme-Machine.* Doctorat d'université, Université de Poitiers.

[Bass *et al.*, 1991] Bass L., Pellegrino R., Reed S., Sheppard S. & Szezur M. (1991). The arch model : Seeheim revisited. In *user Interface Developper's Workshop.*

[Bastide & Palanque, 1990] Bastide R. & Palanque P. (1990). Petri net objects for the design, validation and prototyping of user-driven interfaces. In *3rd IFIP conference Interact'90*, p. 625–631, North-Holland.

[Bellik, 1995] Bellik Y. (1995). *Interfaces Multimodales : concepts, modèles et architecture.* PhD thesis, LIMSI, Université d'Orsay.

[Bernsen & Dybkjaer, 2001] Bernsen N. & Dybkjaer L. (2001). Exploring natural interaction in the car. In *International Workshop on Information Presentation and Natural Multimodal Dialogue*, p. 75–79, Verona, Italy.

[Bersen, 1994] Bersen O. (1994). Foundations of multimodal representation. a taxonomy of representation modalities. *Interacting with Computer*, **6**(4), 347–371.

[Blattner & Dannenberg, 1990] Blattner M. & Dannenberg R. (1990). Chí90 workshop on multimedia and multimodal interface design. *SIGCHI Bulletin*, **22**(2), 54–58.

[Boehm, 1981] Boehm B. (1981). *Structuring the design space*. Prentice-Hall.

[Boehm, 1988] Boehm B. (1988). A spiral model of software developpemen and enhancement. *IEEE Computer*.

[Bolt, 1980] Bolt R. (1980). Put That There : Voice and Gesture at the Graphics Interface. In *SIGGRAPH'80 Proceedings*, volume 14, p. 262–270 : ACM Press.

[Bouchet *et al.*, 2006] Bouchet J., Madani L., Nigay L., Oriat C. & Parissis I. (2006). Formal testing of multimodal interactive systems. In *DSV-IS2006, the XIII International Workshop on Design, Specification and Verification of interactive systems* : Lecture Notes in Computer Science, Springer-Verlag.

[Bouchet *et al.*, 2004] Bouchet J., Nigay L. & Ganille T. (2004). ICARE Software components for rapidly developing multimodal interfaces. In *6th International conference on Multimodal Interfaces, ICMI'04*, p. 251–258, State College, PA, USA.

[Bourguet, 2002] Bourguet M. (2002). Outils de prototypage pour la conception et l'évaluation d'interfaces utilisateur multimodales. In *14eme conférence sur l'interaction Homme-Machine* : ACM Press.

[Bourguet, 2003] Bourguet M. (2003). Designing and prototyping multimodal commands. In R. e. a. M., Ed., *INTERACT'03*, p. 717–720 : IOS Press, (c)IFIP.

[Bourguet, 1992] Bourguet M.-L. (1992). *Conception et réalisation d'une interface de dialogue personne-machine multimodale*. Doctorat d'université, Institut National Polytechnique de Grenoble.

[Bryant, 1990] Bryant R. (1990). Graph-based algorithms for boolean function manipulation. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*.

[Burch *et al.*, 1994] Burch J., Clarke E., Long D., MacMillan K. & Dill D. (1994). Symbolic model checking for sequential circuit verification. *IEEE Transaction on Computer-Aided of Integrated Circuits and Systems*, **13**(4), 401–424.

[Caelen & Coutaz, 1991] Caelen J. & Coutaz J. (1991). Interaction homme-machine multimodale, problèmes généraux. In *IHM'91*, p. 41–58, Dourdon.

[Cansell, 2003] Cansell D. (2003). *Assistance au développement incrémental et à sa preuve*. Habilitation à diriger les recherches, Université Henri Poincaré.

[Caspi & Girault, 1992] Caspi P. & Girault A. (1992). An algorithm for distributing a finite transition system on a shared/distributed memory system. In *PARLE'92*, Paris.

[Clarke & Emerson, 1981] Clarke E. & Emerson E. (1981). Synthesis of synchronization skeletons for branching time temporal logic. In *Longics of Programs : Workshop*, volume 131, York-town Heights, New York : Springer-Verlag.

[Clarke *et al.*,] Clarke E., Grumberg O. & Long D. Model checking and abstraction. In *19th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, p. 343–354, Albuquerque, New Mexico, United States.

[Cleaveland & Steffen, 1993] Cleaveland R. & Steffen J. (1993). The concurrency workbench : A semantic-based tool for the verification of concurrent systems. *ACM Transactions on Programming Languages and Systems*, **15**(1), 36–72.

[Coutaz, 1987] Coutaz J. (1987). PAC an Implementation Model for Dialogue Design. In *Proceedings of INTERACT*, p. 431–437 : North Holland.

[Coutaz, 1990] Coutaz J. (1990). *Interface Homme-Ordinateur, Conception et Réalisation*. Dunod Informatique- Paris.

[Coutaz *et al.*, 1993a] Coutaz J., Faconti G., Nigay L., Paterno F. & Salber D. (1993a). A comparison of approaches for specifying multi-modal interactive systems. In *ERCIM Workshop on Multimodal Human-Computer Interaction*, p. 165–174.

[Coutaz *et al.*, 1993b] Coutaz J., Faconti G., Paterno F., Nigay L. & Salber D. (1993b). *MATIS : a UAN description and lesson learned*. Rapport interne, SM/WP14, ESPRIT BRA 7040 Amodeus-2.

[Coutaz *et al.*, 1995a] Coutaz J., Nigay L., Salber D., Blandford A., May J. & Young R. (1995a). Four easy pieces for assessing the usability of multimodal interaction : The CARE properties. In *IFIP Int. Conf. on Human-Computer Interaction Interact'95*, p. 115–120, London : Chapman & Hall.

[Coutaz *et al.*, 1995b] Coutaz J., Nigay L., Salber D., Blandford A., May J. & Young R. (1995b). Four easy pieces for assessing the usability of multimodal interaction : the CARE properties. In *Proceedings of Human Computer Interaction - Interact'95*, p. 115–120 : Chapman and Hall.

[de Campos, 1999] de Campos J. F. C. F. (1999). *Automated Dedution and Usability Reasoning*. Phd thesis, University of York.

[Dijkstra, 1976] Dijkstra E. (1976). In *A Discipline of Programming*. Prentice-Hall, Englewood Cliffs.

[Dix *et al.*, 1993] Dix A., Janet, Abowd G. & Beale R. (1993). *Human-Computer Interaction*. Prentice Hall.

[du Bousquet *et al.*, 1999] du Bousquet L., Ouabdesselam F., Richier J.-L. & Zuanon N. (1999). Lutess : a specification driven testing environment for synchronous software. In *international Conference of software engineering*, p. 267–276 : ACM Press.

[Duke *et al.*, 1994] Duke D., Barnard P., Duce D. & May J. (1994). *Syndetic model for human-computer interaction*. Rapport interne, ID/WP35, ESPRIT BRA 7040 Amodeus-2.

[Duke & Harisson, 1997] Duke D. & Harisson M. (1997). Mapping user requirements to implementations. *Software Engineering Journal*, **10**(1), 54–75.

[Duke & Harrison, 1993] Duke D. & Harrison M. (1993). Abstract interaction objects. In *EUROGRAPHICS'93, computer Graphics Forum*, volume 12, p. 26–36 : Hubbold RJ., Juan R(eds.).

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

[Elseaidy *et al.*, 1997] Elseaidy W., Cleaveland R. & Jr. J. (1997). Modeling and verifying active structural control systems. *Science of Computer Programming*, **29**, 99–122.

[Faconti & Paterno, 1990] Faconti G. & Paterno F. (1990). An approach to the formal specification of the components of an interaction. In C. Vandoni & D. Duce, Eds., *Eurographics*, p. 481–494.

[Fekete, 1996] Fekete J. (1996). *Un modèle multicouche pour la construction d'applications graphiques interactives*. Doctorat d'université, Université Paris XI, Orsay.

[Fernandez *et al.*, 1992] Fernandez J., Caravel H., Mounier L., Rasse A., Rodriguez C. & Sifakis J. (1992). A toolbox for the verification of lotos programs. In *ICSE'92, 14th International conference on software engineering*, Malbourne.

[Foley *et al.*, 1984] Foley J., V.Wallance & Chan P. (1984). The human factors of computer graphics interaction techniques. *IEEE computer Graphics and Applications*, **4**(11), 13–48.

[Frohlich, 1991] Frohlich D. (1991). The design space of interface,. In *Multimedia Systems, Interaction and Applications, 1st Eurographics Workshop*, p. 53–69 : L. Kjelldahl(Ed.), Springer Verlag.

[Goldberg, 1984] Goldberg A. (1984). *Smalltalk-80 : The Interactive programming Environment*. Addison-Wesley.

[Gram & Cockton, 1996] Gram C. & Cockton G. (1996). *Design principles for Interactive Software*. Chapman & Hall.

[Halbwachs *et al.*, 1991] Halbwachs N., Caspi P., Raymond P. & Pilaud D. (1991). The synchronous dataflow programming language lustre. *Proceedings of the IEEE*, **79**(9).

[HALL, 1991] HALL P. (1991). *OSF/Motif Programmer's Guide*. Open source foundation.

[Harel, 1987] Harel D. (1987). Statecharts : a Visual Formalism for Complex Systems. *Science of Computer Programming*, **8**(3), 231–274.

[Hartson & Gray, 1992] Hartson H. & Gray P. (1992). Temporal aspects of tasks in the user action notation. *Human-Computer interaction*, **7**, 1–45.

[Hoare, 1969] Hoare C. (1969). An axiomatic basis for computer programming. *Communications of ACM*, **12**, 576–580.

[Hoare, 1985] Hoare C. (1985). *Communicating Sequential Processes*. Prentice Hall International.

[Holzmann, 1991] Holzmann G. J. (1991). *Design and Validation of Computer Protocols*. Prentice Hall Int.

[ISO84, 1984] ISO84 (1984). *ISO - Information Processing Systems - Definition of the temporal ordering specification language lotos*. Rapport interne, TC 97/16 N1987, ISO.

[ISO8807, 1989] ISO8807 (1989). *Information processing systems - open systems interconnection - LOTOS - A formal description technique based on the temporal ordering of observational behavior*. Rapport interne, 8807, ISO.

[Jambon, 2002] Jambon F. (2002). From formal specification to secure implementations. In *Computer-Aided Design of User Interfaces, (CADUI'2002)*, p. 43–54, Valenciennes, France : Kluwer Academics.

[Jones, 1990] Jones C. (1990). *Systematic software Developement Using VDM*. Prentice-Hall, Upper Saddle River, NJ 07458, USA.

[Jourde *et al.*, A paratre 2006] Jourde F., Nigay L. & Parissis I. (A paraître 2006). Test formel de systèmes interactifs multimodaux : couplage ICARE - Lutess. In *ICSSEA2006, 19ème journées Internationales "génie logiciel & Ingénierie de Systèmes et leurs Applications" Globalisation des services et des systèmes*.

[Juster & Roy, 2004] Juster J. & Roy D. (2004). Elvis : Situated speech and gesture understanding for a robotic chandelier. In *Sixth International Conference on Multimodal Interfaces*, p. 90–96 : ACM Press.

[Kamel, 2004] Kamel N. (2004). Utilisation de SMV pour la vérification de propriétés d'IHM multimodales. In *16° Conférence Francophone sur l'Interaction Homme-Machine (IHM'2004)*, p. 219–222, Namur, Belgique : ACM Press.

[Kamel & Ait-Ameur, 2005] Kamel N. & Ait-Ameur Y. (2005). Mise en oeuvre d'IHM Multimodales dans un système de CAO. Une approche fondée sur les méthodes formelles. *Revue internationale d'ingénierie numérique*, **1**(2), 235–256.

[Kurshan, 1994] Kurshan R. (1994). *Computer-aided verification of coordinating Processes : The Automata-Theoretic Approach*. Princeton University Press.

[Loer & Harrisson, 2001] Loer K. & Harrisson M. (2001). Formal interactive systems analysis and usability inspection methods : two incompatible worlds. In P. Palaque & F. Paterno, Eds., *7th international Workshop on Design Specification and Verification of Interactive Systems (DSV-IS 2000)*, volume 1946, p. 169–190 : Lecture Notes in Computer Science, Springer Verlag.

[MacColl & Carrington, 1998] MacColl I. & Carrington D. (1998). *Testing MATIS : a case study on specfication based testing of interactive systems*. Formal Aspects of HCI (FAHCI98), ISBN 0-86339-7948.

[Manna & Pnueli, 1992] Manna Z. & Pnueli A. (1992). *The Temporal Logic of reactive and Concurrent Systems : Specification*. Springer-Verlag.

[Markopoulos, 1995] Markopoulos P. (1995). On the expression of interaction properties within an interactor model. In *DSV-IS95 : Design, Specification, Verification of Interactive Systems*, p. 294–311 : Springer Verlag.

[Markopoulos, 1997] Markopoulos P. (1997). *A compositional model for the formal specification of user interface software*. Phd thesis, University of London.

[Markopoulos *et al.*, 1996] Markopoulos P., Rowson J. & Johnson P. (1996). Dialogue modelling in the framework of an interactor model. In *the 3rd International Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS '96*, Namur, Belgique.

[Martin, 1997] Martin J. (1997). TYCOON : Theoritical Framework and Software Tools for multimodal interfaces. In *Intelligence and Multimodality in Multimedia Interfaces* : AAAI press.

[Martin & Beroule,] Martin J. & Beroule B. Multimodal interfaces based on types and goals of cooperation between modalities. In *IMMI-1, 1st International Workshop on intelligence and Multimodality in Multimedia Interfaces : research and applications*, Edinburgh.

[Martin, 1995] Martin J.-C. (1995). *Coopération entre modalités et liage par synchronie dans les interfaces multimodales*. Doctorat d'université, Ecole Nationale Supérieure des Télécomunications, Paris.

[McDermid & Ripkin, 1984] McDermid J. & Ripkin K. (1984). *Life Cycle Support in the ADA environnement*. Cambridge University Press.

[Merloz *et al.*, 2000] Merloz P., Lavallee S., Tonnetti J. & Pittet L. (2000). Image-guided spined surgery : Technology, operative technique and clinical practice. *Operative Techniques in orthopaedics*, **10**(1), 56–63.

[Milner, 1980] Milner R. (1980). A calculus of comunicating systems. New York, NY, USA : LNCS 92, Springer-Verlag.

[Myers, 1995] Myers A. B. (1995). User interface software tools. *ACM Transaction on Computer Human Iteraction*, **2**(1), 64–103.

[Navarre *et al.*, 2005] Navarre D., Palanque P., Bastide R., Schyn A., Winckler M., Nedel L. & Freitas C. (2005). A formal description of multimodal interaction techniques for immersive virtual reality applications. In *INTERACT 2005*, p. 25–28, Roma, Italy : Lecture Notes in Computer Science, Springer Verlag.

[Nedel *et al.*, 2003] Nedel L. P., Freitas C., Jacob L. & Pimenta M. (2003). Testing the use of egocentric interactive techniques in immersive virtual environments. In *INTERACT 2003 - Ninth IFIP TC13 International Conference on Human-Computer*, p. 471–478, Amsterdam : IOS Press.

[Nigay, 1994] Nigay L. (1994). *Conception et modélisation logicielle des Systèmes Interactifs : Application aux interfaces Multimodales*. Doctorat d'université, Université de Joseph Fourier, Grenoble.

[Nigay & Coutaz, 1993a] Nigay L. & Coutaz J. (1993a). A design space for multimodal interfaces : concurrent processing and data fusion. In *Proceedings of INTERCHI-93 - INTERCHI-93*, p. 172–178 : ACM Press.

[Nigay & Coutaz, 1993b] Nigay L. & Coutaz J. (1993b). A design space for multimodal systems : Concurrent processing and data fusion. In *INTERACT'93*, p. 172–178, Amesterdam.

[Nigay & Coutaz, 1996] Nigay L. & Coutaz J. (1996). Espaces conceptuels pour l'interaction multimédia et multimodale. *TSI, Spécial Multimédia et Collecticiel*, **15**(9), 1195–1225.

[Nigay & Coutaz, 1997] Nigay L. & Coutaz J. (1997). Multifeature systems : The CARE properties and their impact on software design. In *Intelligence and Multimodality in Multimedia interfaces* : AAAI Press.

[Normand, 1992] Normand V. (1992). *Le modèle SIROPO : de la spécification conceptuelle des interfaces à leur réalisation*. Doctorat d'université, Université Joseph Fourier, Grenoble.

[Olsen, 1990] Olsen D. (1990). Propositional production systems for dialogue description. In *Human factors in Computing Systms : CHI'90*, p. 57–63 : ACM Press.

[Palanque *et al.*, 1995] Palanque P., Bastide R. & Sengès V. (1995). Validating interactive system design through the verification of formal task and

system models. In L. J. Bass & C. Unger, Eds., *IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95)*, p. 189–212, Grand Targhee Resort (Yellowstone Park), USA : Chapman & Hall.

[Palanque & Schyn, 2003] Palanque P. & Schyn A. (2003). A Model-based for Engineering Multimodal Interactive Systems. In *9th IFIP TC13 International Conference on Human Computer Interaction (Interact'2003)*.

[Paternò *et al.*, 2001] Paternò F., Mori G. & Galimberti R. (2001). CTTE : An environment for analysis and development of task models of cooperative applications. In *ACM CHI 2001*, volume 2, Seattle : ACM/SIGCHI.

[Paterno, 1995] Paterno F. (1995). *A Method for formal specification and verification of interactive systems*. PhD thesis, Departement of computer system, Université de York.

[Paterno, 2001] Paterno F. (2001). *Model-Based Design and Evaluation of Interactive Application*. Springer.

[Paterno & Faconti, 1992] Paterno F. & Faconti G. (1992). On the use of LOTOS to describe graphical interaction. p. 155–173 : Cambridge University Press.

[Paterno & Mezzanotte, 1994] Paterno F. & Mezzanotte M. (1994). *Analysing MATIS by Interactors and ACTL*. Rapport interne, Amodeus Esprit Basic Research Project 7040, System Modelling/WP36.

[Paterno *et al.*, 1992] Paterno F., Mori G. & Galimberti R. (1992). UIMS. In *The UIMS Workshop Tool Developers : A Metamodel for the Runtime Architecture of an Interactive System*, volume 24, p. 32–37.

[Peterson, 1981] Peterson J. L. (1981). *Petri Net Theory and the Modeling of Systems*. ISBN : 0-13-661983-5.

[Pfaff, 1985] Pfaff G. (1985). *User Interface Management Systems*. Springer-Verlag.

[Plotkin, 1981] Plotkin G. (1981). *A Structural Approach to Operational Semantics*. Rapport interne, Department of of computer Science, University of Arhus DAIMI FN 19.

[Queille & Sifakis, 1981] Queille J. & Sifakis J. (1981). Specification and verification of concurrent systems in caesar. In *5th International Symposium on Programming*, p. 337–351 : Springer Verlag.

[Quemada *et al.*, 1993] Quemada J., Pires J. F., Marias J., Azcorra A. & Robles T. (1993). *Introduction to Lotos*, In K. Turner, Ed., *Using formal description techniques - an introduction to Estelle, lotos and SDL*, p. 47–83. Wiley.

[Robert, 1992] Robert P. (1992). *Le Petit Robert1, dictionnaire alphabétique et analogique de la langue française*. Edition Les dictionnaire Robert-Canada S.C.C.

[Roche, 1998] Roche P. (1998). *Modélisation et validation d'interface homme-machine*. Doctorat d'université, Ecole Nationale Supérieure de l'Aéronautique et de l'Espace.

[Rousseau *et al.*, 2004] Rousseau C., Bellik Y., Vernier F. & Bazalgette D. (2004). Architecture framework for output multimodal systems design. In *Proceedings of OZCHI'04*, Wollongong, Australia.

[Royce, 1970] Royce W. (1970). Managing the development of large software systems. In *IEE WESTCON*, p. 1–9.

[Ruys & Langerak, 1997] Ruys T. & Langerak R. (1997). Validation of Bosch mobile communication network architecture with SPIN. In *3rd SPIN Workshop, Enshede, NL*.

[Ryan *et al.*, 1991] Ryan M., Fiadeiro J. & Maibaum T. (1991). Sharing actions and attributes in modal action logic. In T. Ito & A. Meyer, Eds., *Theoretical aspect of computer software*, volume 526, p. 569–593 : Lecture Notes in Computer Science, Springer Verlag.

[Scapin & Pierret-Golbreich, 1989] Scapin D. & Pierret-Golbreich C. (1989). Towards a method for task description : MAD. In D. Berlinguet L. Berthelette, Ed., *Conférence of Work with Display Units WWU'89*, p. 27–34, Amsterdam : Elsevier Science.

[Schyn, 2005] Schyn A. (2005). *Une approche fondée sur les modèles pour l'ingénierie des systèmes interactifs multimodaux*. Doctorat d'université, Toulouse III.

[Schyn *et al.*, 2003] Schyn A., Navarre D., Palanque P. & Nedel L. P. (2003). Description formelle dune technique dinteraction multimodale dans une application de réalité virtuelle immersive. In *Proceeding of the 15th French Speaking conference on human-computer interaction (IHM'2003)*, p. 25–28, Caen, France.

[Spivey,] Spivey J. *Understanding Z : A Specification Language and its Formal Semantics*, volume 3. Cambridge Tracts in Theoretical Computer Science, Cambridge, Cambridge University Press.

Annexe

Annexe A

Mise en oeuvre de la technique de model-checking avec SMV pour un système de CAO

Nous présentons dans cette annexe une mise en oeuvre utilisant la technique de model-checking avec SMV, pour un système de CAO (Conception Assisté par Ordinateur) [Kamel & Ait-Ameur, 2005]. Cette mise en oeuvre montre l'intérêt de la modélisation formelle de l'interaction dans un système de CAO. Nous avons choisi de modéliser les interactions d'une application de CAO simple. Un utilisateur choisit des formes géométriques 3D simples (cube, cylindre, etc) pour les assembler pour concevoir des formes plus complexes. Plusieurs possibilités sont offertes à l'utilisateur pour formuler ses requêtes. Il utilise pour cela, la manipulation directe avec la souris, la parole ou les deux. L'interface contient un ensemble d'icônes que l'utilisateur peut sélectionner pour désigner le type d'objet de base à manipuler (cylindre, cube, etc). Des boutons sont également disponibles, des boutons sur lesquels l'utilisateur peut cliquer pour déclencher une commande telle que ajouter un objet, le supprimer ou le déplacer.

Nous avons choisi de modéliser l'IHM3 selon deux types. Un type exclusif où chaque commande doit être réalisée avec une seule modalité, et un type parallèle alterné où chaque commande peut être réalisée avec plusieurs modalités en parallèle entrelacé. La tâche que nous considérons consiste à concevoir un objet simple composé d'un cube au dessus duquel est placé un cylindre. Avant de présenter les deux modélisations, nous présentons les ensembles d'actions de base du modèle :

- $ASouris = \{ClicCube, ClicCylindre, ClicAjou, ClicSup, Clic(x, y)\}$
- $Aparole = \{'Supprimer', 'Ajouter', 'Cube', 'Cylindre', 'a', 'ici', 'positionxy'\}$

x, y sont des réels obtenus par le clic de la souris dans Asouris ou bien reconnus par la voix pour Aparole. Les actions générées par la modalité souris sont :

- *ClicCube*, *ClicCylindre* : clic de la souris respectivement sur les objets cube et cylindre présents sur l'interface ;

- *ClicAjou*, *ClicSup* : clic de la souris respectivement sur les boutons Add et Del ;
- $Clic(x, y)$: clic de la souris sur le point de coordonnées (x, y) dans l'espace de conception de l'IHM3. En réalité, c'est un ensemble d'événements de même nature. A chaque couple (x, y) correspond une action. Pour notre étude de cas, $(x1, y1)$ et $(x2, y2)$ sont les coordonnées où doivent être placés le cube et le cylindre. Nous considérons qu'une conversion des points 2D en points 3D est disponible. Nous n'entrons pas dans les détails de cette conversion qui n'est pas essentielle à la compréhension de cette étude de cas.

Les actions de l'ensemble *Aparole* sont des mots prononcés par l'utilisateur et reconnus par le système. Pour chacun des deux modèles choisis pour modéliser notre IHM, nous donnons les expressions des énoncés en fonction des actions de base qui les composent. Ensuite, l'expression de la tâche qui conçoit l'objet en fonction des énoncés qui la composent, et enfin, l'expression de la même tâche en fonction des événements de base qui la composent en remplaçant chacun des énoncés par son expression.

A.1 Modélisation du type exclusif

La modélisation d'une IHM3 de type exclusif où chaque énoncé est produit par une seule modalité est d'abord présentée. Les énoncés sont composés en séquence avec l'opérateur du choix ou de la séquence pour formuler des tâches.

A.1.1 Les énoncés :

Nous définissons quatre énoncés $E1$, $E2$, $E3$ et $E4$ composé chacun d'un ensemble d'actions de base des ensembles *Aparole* et $Asouris$. Pour réduire l'espace pris par l'ecriture des expressions, nous avons renommé les actions de base pour chaque énoncé Ei en un ensemble d'événements eij.

$$\begin{array}{|l|}\hline E1 = clicAjou; clicCube; clic(x1, y1) = e11; e12; e13 \\ E2 = clicAjou; cliccylindre; clic(x2, y2) = e21; e22; e23 \\ E3 = 'Ajouter'; 'Cube'; 'positionx1y1' = e31; e32; e33 \\ E4 = 'Ajouter'; 'Cylindre'; 'positionx2y2' = e41; e42; e43 \\ \hline \end{array}$$

A.1.2 Les tâches :

Nous définissons deux tâches à réaliser *"placer le cube et placer le cylindre"*. La tâche PlacerCube consiste à placer un cube aux coordonnées $(x1, y1)$. Elle est réalisée soit avec l'énoncé $E1$ soit avec l'énoncé $E3$. Elle est le résultat de la composition

de ces deux énoncés par l'opérateur du choix ([]). Le même raisonnement est appliqué pour la tâche PlacerCylindre. Les expressions des deux tâches en fonctions des énoncés sont données par

$$\boxed{\begin{array}{l} PlacerCube = E1[]E3 \\ PlacerCylindre = E2[]E4 \end{array}}$$

La tâche *Concevoir* est la tâche globale. Elle consiste à composer l'objet complexe sur l'espace de conception. Elle est réalisée par les deux tâches Placercube et PlacerCylindre. L'utilisateur peut placer le cube ensuite le cylindre ou bien, le cylindre et ensuite le cube. Ceci se traduit par l'expression suivante

$$\boxed{\begin{array}{l} Concevoir = (PlacerCube >> PlacerCylindre) \\ \qquad\qquad\qquad [](PlacerCylindre >> PlacerCube) \end{array}}$$

En fonction des énoncés, cette expression est dépliée en :

$$\boxed{Concevoir = ((E1[]E3) >> (E2[]E4))[]((E2[]E4) >> (E1[]E3))}$$

et en fonction des actions de base, elle correspond à l'expression

$$\boxed{\begin{array}{l} Concevoir = \\ (((e11; e12; e13))[](e31; e32; e33)) >> ((e21; e22; e23)[](e41; e42; e43))) \\ [] \\ (((e21; e22; e23)[](e41; e42; e43)) >> ((e11; e12; e13)[](e31; e32; e33))) \end{array}}$$

Le système de transitions correspondant à cette expression est obtenu par la composition des systèmes de transitions des énoncés qui le composent. Le système résultant est décrit sur la figure A.1. Nous utilisons des flèches pleines pour les transitions réalisées avec les clics de la souris et des flèches en pointillés pour celles réalisées par la parole.

A.2 Modélisation de l'IHM3 de type alternée

Nous donnons ici, la modélisation d'une IHM3 de type parallèle alterné où chaque énoncé est produit par les deux modalités en séquence ou en parallèle entrelacé. Nous n'avons pas utilisé l'opérateur parallèle au niveau des tâches.

A.2.1 Les énoncés :

Les deux modalités participent à la réalisation des énoncés $E1$, $E2$, $E3$ et $E4$. Les énoncés $E3$ et $E4$ sont réalisés en combinant la parole et les clics souris en

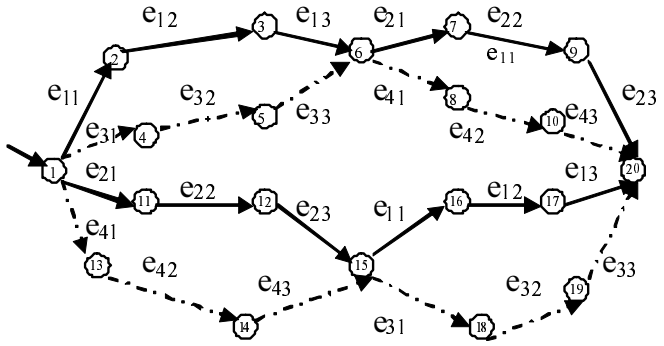

Fig. A.1 – Système de transitions de type exclusif

parallèle entrelacé. Dans cet exemple et contrairement au précédent. Par exemple, l'utilisateur prononce la phrase *'Ajouter ça ici'* pendant qu'il clique sur l'objet à ajouter et son emplacement sur l'écran. Les coordonnées récupérées par les clics souris sont fusionnées avec la commande *'Ajouter ça ici'*. Notons que les clics de la souris peuvent précéder la phrase ou inversement ou bien intervenir pendant qu'elle est prononcée.

$$
\begin{array}{|l|}
\hline
E1 =' Ajouter';' Cube'; Clic(x1, y1) = e11; e12; e13 \\
E2 = Ajouter';' Cylindre'; Clic(x2, y2) = e21; e22; e23 \\
E3 = ('Ajouter';' a';' ici')|||(ClicCube; Clic(x1, y1) \\
= (e31; e32; e33)|||(e34; e35) \\
E4 = ('Ajouter';' a';' ici')|||(ClicCylindre; Clic(x2, y2) \\
= (e41; e42; e43)|||(e44; e45) \\
\hline
\end{array}
$$

A.2.2 Les tâches :

nous avons gardé les mêmes tâches que celles utilisées dans l'IHM3 de type exclusif. Les expressions des tâches *PlacerCube* et *PlacerCylindre* restent identiques.

$$
\begin{array}{|ll|}
\hline
PlacerCube & = (E1[]E3) \\
PlacerCylindre & = (E2[]E4) \\
Concevoir & = (PlacerCylindre >> PlacerCube)[] \\
 & (PlacerCube >> PlacerCylindre) \\
 & = ((E1[]E3) >> (E2[]E4))[]((E2[]E4) >> (E1[]E3)) \\
\hline
\end{array}
$$

L'expression de la tâche *Concevoir* est donnée en fonction des actions de base comme suit

$$
\begin{array}{|ll|}
\hline
Concevoir = & (((e11;e12;e13)[](e31;e32;e33)|||(e34;e35)) \\
& >> \\
& ((e21;e22;e23)[](e41;e42;e43)|||(e44;e45)))) \\
& [] \\
& (((e21;e22;e23))[](e41;e42;e43)|||(e44;e45))) \\
& >> \\
& ((e11;e12;e13)[](e31;e32;e33)|||(e34;e35)))) \\
\hline
\end{array}
$$

A.3 Expression des propriétés d'utilisabilité

Nous avons choisi d'exprimer et de vérifier les deux propriétés d'utilisabilité qui sont l'équivalence et la complémentarité, sur les deux IHM3.

$$
\begin{array}{|l|}
\hline
\left.\begin{array}{l}(\mathbf{AG}(((etat=1)and(modalite=parole)) \Rightarrow \\ \mathbf{E}((action=parole)\bigcup(Etat=20)))))\end{array}\right\}(1) \\
and \\
\left.\begin{array}{l}(\mathbf{AG}(((etat=1)and(modalite=manipulation)) \Rightarrow \\ \mathbf{E}((modalite=manipulation)\bigcup((etat=20))))))\end{array}\right\}(2) \\
\hline
\end{array}
$$

Tab. A.1 – Propriété d'équivalence de *parole* et *manipulation directe*

La formule présentée sur le tableau A.1 exprime l'équivalence des deux modalités. Elle indique formellement que

1. sur toutes les exécutions, si un état est identifié comme l'état initial (etat=1) de la tâche considérée, alors il existe un chemin allant de cet état tel que tous les états sont atteints par la modalité *parole* jusqu'à atteindre l'état final de la tâche (etat=20) et
2. sur toutes les exécutions, si un état est identifié comme l'état initial (etat=1) de la tâche considérée, alors il existe un chemin allant de cet état tel que tous les états sont atteints par la modalité *manipulation directe* jusqu'à atteindre l'état final de la tâche (etat=20).

$$
\begin{array}{|l|}
\hline
\left.\begin{array}{l}(\mathbf{EG}((etat=1) \\ \Rightarrow \mathbf{E}(((modalite=parole)or(modalite=manipulation))\bigcup(etat=20)))\end{array}\right\}(1) \\
\text{and} \\
\left.\begin{array}{l}not\mathbf{EG}((etat=1)\Rightarrow\mathbf{E}((modalite=parole)\bigcup(etat=20)))\end{array}\right\}(2) \\
\text{and} \\
\left.\begin{array}{l}not\mathbf{EG}((etat=1)\Rightarrow\mathbf{E}((modalite=manipulation)\bigcup(etat=20)))\end{array}\right\}(3) \\
\hline
\end{array}
$$

Tab. A.2 – Propriété de complémentarité de *parole* et *manipulation directe*

La formule présentée sur le tableau A.2 exprime la complémentarité des deux modalités. Elle indique formellement que

1. il existe une exécution telle que, si un de ses états est identifié comme l'état initial (etat=1) de la tâche considérée, alors il existe un chemin allant de cet état tel que, tous ses états sont atteints par la modalité *parole* ou *manipulation directe*, jusqu'à atteindre l'état final de la tâche (etat=20) ; et
2. il n'existe pas d'exécution telle que, toutes les transitions, entre l'état initial et l'état final, de la tâche sont effectuées par la modalité *parole* ; et
3. il n'existe pas d'exécution telle que toutes les transitions, entre l'état initial et l'état final de la tâche, sont toutes effectuées par la modalité *manipulation directe*.

Nous avons codé les deux systèmes ainsi que les propriétés associées en SMV. La propriété d'équivalence des deux modalités *manipualtion directe* et *parole* est vérifiée par le système de l'IHM3 de type exclusif, mais ne l'est pas par le système de l'IHM3 de type parallèle alterné. La propriété de complémentarité est vérifiée pour l'IHM3 de type parallèle alterné, mais ne l'est pas par l'IHM3 de type exclusif.

L'intérêt de la propriété d'équivalence est que si une des deux modalités est absente pour une raison ou une autre, ou bien si l'utilisateur ne peut pas utiliser la souris pour une raison d'handicap, l'IHM3 est toujours utilisable. Par ailleurs dans le type parallèle alterné, l'utilisateur profite des avantages des deux modalités en même temps en accélérant ainsi le processus de conception en utilisant la phrase *'Ajouter ça ici'* et les deux clics $(ClicCube, Clic(x1, y1))$ en même temps pour réaliser sa tâche. Cependant, l'inconvénient est que si une des deux modalités est absente, alors l'IHM3 n'est plus utilisable. L'utilisation du type parallèle dans un système de CAO implique une réduction des chemins d'interaction et une utilisation plus efficace et plus ergonomique de ces systèmes.