Ecole Nationale Supérieure de Mécanique et d'Aérotechnique

Ecole Doctorale des Sciences et Ingéniérie pour l'Information

THESE

pour l'obtention du grade de

DOCTEUR DE L'ECOLE NATIONALE SUPERIEURE DE MECANIQUE ET D'AEROTECHNIQUE

(Diplôme National — Arrêté du 7 Août 2006)

Ecole Doctorale : Sciences et Ingéniérie pour l'Information

Secteur de Recherche : INFORMATIQUE et APPLICATIONS

Présentée et soutenue publiquement devant la Commission d'Examen
le 7 Décembre 2009 par :

Sybille Caffiau

****

Approche dirigée par les modèles pour la conception et la validation des applications interactives : une démarche basée sur la modélisation des tâches

****

Directeurs de Thèse : ***Patrick GIRARD et Dominique SCAPIN***

JURY

Rapporteurs :	Gaëlle CALVARY	Professeur, Université de Grenoble
	Jean VANDERDONCKT	Professeur, Université catholique de Louvain
Président :	Rémi BASTIDE	Professeur, Centre Universitaire de Formation et de Recherche Jean-François Champollion, Castres
Examinateurs :	Laurent GUITTET	Maître de Conférences, LISI-ENSMA, Poitiers
	Patrick GIRARD	Professeur, LISI-ENSMA, Université de Poitiers
	Dominique SCAPIN	Directeur de Recherche, INRIA, Rocquencourt

Remerciements

Il m'est très difficile d'écrire les remerciements, non pas qu'il manque de personnes à remercier mais parce qu'il me faut trouver des formules qui me permettent d'exprimer ma gratitude. Cet exercice est d'autant plus difficile que je sais qu'une partie des gens que je souhaite remercier ne lira que cette page de ma thèse alors qu'une autre est citée dans les remerciements des thèses plusieurs fois par an. Je vais tout de même m'y essayer en réclamant votre indulgence.

Je souhaiterais en tout premier lieu remercier mes encadrants de thèse : **Patrick Girard** et **Dominique Scapin**. Vous avez tous deux apporté deux points de vue distincts qui ont rendu la réalisation de ce travail très enrichissant. Dominique merci d'avoir eu la patience de composer avec notre point de vue parfois un peu trop *informatique*. Patrick, vous m'avez fait découvrir le domaine de l'IHM, de la recherche, de l'enseignement en ayant toujours pour mes idées une oreille attentive et en essayant toujours de voir mon intérêt avant tout, il n'y a pas de mots pour vous exprimer ma gratitude pour tout ce que vous avez fait pour moi. **Laurent Guittet**, bien que tu ne sois pas un de mes encadrants, tu as grandement participé à mes travaux, merci pour ta présence et les nombreux échanges que nous avons eus.

Vous avez été tous les trois mes principaux interlocuteurs pour mes réflexions mais tout au long de ma thèse, j'ai également bénéficié de la présence de plusieurs *familles* de personnes. La première est la famille constituée des membres du LISI ; merci à **Guy Pierra** et **Yamine Aït-Ameur** de m'y avoir accueilli ; merci à tous ses membres de contribuer à ce que le LISI soit un laboratoire où il est agréable de faire une thèse, et particulièrement à **Fred** et **Claudine** qui constituent, sans nul doute, la colonne vertébrale de l'organisation de cette famille. La seconde famille est celle constituée des membres de la communauté francophone d'IHM. Ma première rencontre avec elle a eu lieu pendant mon stage de Master recherche et par la suite, lors de chacune des rencontres (RJC et Conférences) j'ai pu prendre pleinement conscience de l'esprit de communauté qui s'y exerce. Merci à cette famille (et surtout à ces fondateurs) d'exister.

Appartenant à cette famille, merci particulièrement à **Jean Vanderdonckt** et **Gaëlle Calvary** pour l'intérêt que vous voulez bien porter à mes travaux. Cet intérêt est illustré par l'honneur que vous me faites d'avoir accepté d'être rapporteurs de cette thèse mais également par les échanges que nous avons eus que ce soit par emails ou de vive voix lors de nos rencontres. Merci également à **Rémi Bastide** d'avoir accepté d'être membre du jury de cette thèse.

La dernière famille que je souhaiterais remercier est celle qui m'a accueillie en tout premier lieu, et sur laquelle je peux toujours compter pour partager mon stress, mes problèmes, ma mauvaise humeur (parfois) mais aussi tous les bons moments. Maman, Papa merci de toujours me soutenir, d'avoir toujours fait les choix que vous avez faits *pour* nous. Ma petite Laurianne, merci d'être toi ; un instant emportée et irrationnelle et l'instant d'après adorable et joyeuse. Baptiste, chaque fois que je te demande d'être présent tu es là, même si c'est en râlant, merci. Merci à vous quatre, cette thèse est un peu la vôtre car elle n'aurait jamais pu exister sans vous. Pour vous aider dans votre tâche de m'aider à me changer les idées, j'ai toujours pu compter sur l'appui de ceux

que je considère comme étant mes grands-parents, Papy Guy et Mamy Simone et sur les deux derniers membres de ma famille (mais pas les moins actifs) Tanguy et Phebe. Merci à vous.

Merci à mes amis qu'ils soient du labo ou de l'extérieur, que nous nous soyons rencontrés il y a des années ou seulement quelques mois, les moments que nous passons ensemble sont, pour moi, autant de bons moments associés à cette thèse ; alors merci à vous : Cécile, Olivier K., Olivier D., Sylvie A., Denis, Loé -mon grand-frère dans la famille IHM- qui a toujours été présent pour répondre à mes questions, Nico, Sylvie G. (collègue à distance qui a souvent collaboré à mes publications en me relisant). Merci à Françoise et Jean-François pour l'intérêt que vous avez toujours bien voulu me porter. Je n'oublie pas Chimène, qui a partagé avec moi toutes les étapes de réalisation de nos thèses, de la préparation des examens en Master à la rédaction de ce document mais aussi tous les moments de joie qui nous sont arrivés pendant ces quatre années (de nos premières publications à l'arrivée de Phebe dans nos vies).

Enfin, merci à Françoise et Patrick qui forment avec leurs enfants Sylvie, Yannick et Claire, et Mamy, un foyer prêt à accueillir qui le veut. Dans votre refuge, nous nous savons toujours bien accueilli.

Comment remercier nominativement toutes les personnes qui ont joué un rôle dans l'accomplissement de cette thèse ? C'est impossible et je ne prendrai pas le risque d'en oublier. Cependant, en plus des personnes sus-citées, je voudrais particulièrement remercier l'ensemble de mes enseignants qui, depuis que je suis scolarisée m'ont appris tant de chose et m'ont encouragée à continuer à apprendre. Merci à vous, Mme Desbrousses, M. Moreau, Annie Geniet, Hélène Tomlinson (qui, avec patience, continue à m'aider à améliorer mon anglais) et tous les autres.

Résumé

Actuellement, les applications interactives sont utilisées dans de nombreux domaines (guichets automatiques, tours de contrôle, …), par des publics très différents (enfants, experts, handicapés, …) et par un nombre important d'utilisateurs (interfaces de téléphones portables, …) ou au contraire très spécifiques (logiciels conçus spécifiquement pour une entreprise, …). Elles sont de ce fait très diverses et impliquent la recherche de techniques de conception adaptées. Afin de s'adapter à cette diversité, des recherches sont menées pour adapter les avancés technologiques en développant de nouvelles techniques d'interaction, de nouveaux supports d'interaction…

De par ce besoin en systèmes interactifs adaptés et la multiplicité des paramètres à prendre en compte (plateforme, interaction, utilisateur), la conception et le développement des applications interactives sont devenus très coûteux. Afin de réduire ces coûts, des recherches sont actuellement menées sur le processus de conception. Cette thèse s'inscrit dans ces travaux.

L'un des axes étudiés pour réduire le coût de production des applications interactives est la détection des erreurs le plus en amont possible pendant le processus de conception. Nous proposons de faciliter la vérification et la validation de la dynamique des applications (plus spécifiquement dénommée dialogue) tout au long de la conception, en fonction des spécifications recueillies auprès des futurs utilisateurs, exprimées sous forme de modèles de tâches. Les modèles de dialogue et les modèles de tâches représentent deux points de vue différents et complémentaires pour une même application. Nous proposons une approche de vérification de cohérence entre ces deux modèles tout au long du cycle de vie de l'application. Pour cela, nous avons défini des règles de cohérence entre les modèles que nous vérifions formellement en utilisant une méta-modélisation des formalismes que nous avons choisis après évaluation de leur utilisation pour une conception centrée-utilisateur. Cette thèse fait appel à des connaissances dans les domaines de l'ergonomie, de l'Ingénierie Dirigée par les Modèles (IDM) et de l'Interaction Homme-Machine.

Abstract

Nowadays, interactive applications are used in many contexts (automatic cash dispenser, traffic-control, …), by different user types (children, experts, disabled people, …), for all users (phone interfaces…) or on contrary for very specific users (software designed for a specific company). They therefore are very different and they imply specific design research techniques. In order to enable such diversity, research is performed to integrate new technologies by developing new interaction techniques, new interaction supports, …

Due to this need of adapted interactive software and the multiplicity of the parameters that the designer has to take into account (platform, interaction, user…), the cost of the design and the development of interactive application increases. In order to decrease these costs, studies are performed on the design process, such as the work presented in this thesis.

One of the issues studied to decrease the interactive application production cost is to detect errors as early in the design process as possible. We propose to facilitate the checking and the validation of the dynamics of applications (named *dialog*) in the whole design, according to the design requirements gathered from the future users and expressed by task models. Dialog and task models express two different and complementary viewpoints on a same application. We propose an approach to check the coherence between both models during the whole life cycle of applications. This approach uses coherence rules defined during the thesis, and formally checked using a meta-modelling of both formalisms chosen after evaluation of their use in a user-centred design. This thesis uses knowledge from the fields of Ergonomics, Model-Driven Engineering (MDE) and Human-Computer Interaction.

Table des matières

Table des Figures

Liste des Tableaux

Chapitre 1. Introduction

1.1. Introduction générale

Aujourd'hui, nous sommes tous confrontés à l'utilisation des interfaces pour accomplir les tâches de tous les jours, que ce soit dans le domaine professionnel ou dans le domaine personnel : recherche d'itinéraire à l'aide d'un GPS, programmation d'un enregistreur vidéo, élaboration d'un contrat, organisation d'une réunion, achat d'un billet de train sur un automate ou sur Internet…

Cette omniprésence des applications interactives entraîne une variation importante des types d'applications disponibles, qui s'applique :

- aux types de plateforme : tablette, ordinateur, téléphone…
- aux techniques d'interaction : WIMP, manipulation directe,…
- aux dispositifs d'entrée et de sortie : sonores, visuels…
- au nombre de plateformes et de techniques d'interaction utilisées : réalité virtuelle, application mixtes…

Ces multiples types d'applications interactives sont nécessaires pour s'adapter aux besoins multiples des domaines d'application (médicale, domotique, trafic aérien…), aux besoins de la prise en compte des profils utilisateurs (novice, expert, handicapé, enfant, senior…) et à la prise en compte des avancées techniques telles que le développement de nouvelles plateformes.

Considérer tous ces paramètres dans la phase de conception engendre une grande difficulté, nécessitant l'intervention de spécialistes ayant des connaissances distinctes, et multiplie par là-même les coûts. Par exemple, une conception centrée-utilisateur peut faire intervenir un ergonome exprimant un point de vue *utilisateur* et un développeur exprimant un point de vue *système* sur la même application. Bien que complémentaires, ces deux points de vue sont trop souvent réalisés avec un minimum d'interactions, ne permettant pas d'assurer les relations entre les différents points de vue. Chacun de ces intervenants ayant son (ou ses) domaine(s) d'expertise, il est indispensable de trouver des moyens pour permettre la communication pour l'échange des différents points de vue sur l'application.

Pour répondre à ce besoin d'échange d'information et proposer des méthodes facilitant la conception tout en limitant les coûts de production, de nombreuses études s'orientent vers l'utilisation de modèles. Nous souhaitons appliquer ces travaux dans un contexte de conception mêlant le point de vue de l'utilisateur et le point de vue du concepteur informatique sur la dynamique du système.

Un moyen d'exprimer le point de vue de l'utilisateur par un modèle est de concevoir un modèle de tâches exprimant les activités humaines pour lesquelles l'application sera conçue. Ce modèle permet de représenter les activités en les décomposant sous forme d'arbre de tâches. Ces arbres de tâches peuvent être simulés dynamiquement afin de produire des scénarios d'exécution. Cette simulation dynamique

semble très proche de la dynamique finale de l'application (Figure 1.1.1), qui est généralement représentée par un modèle de dialogue.

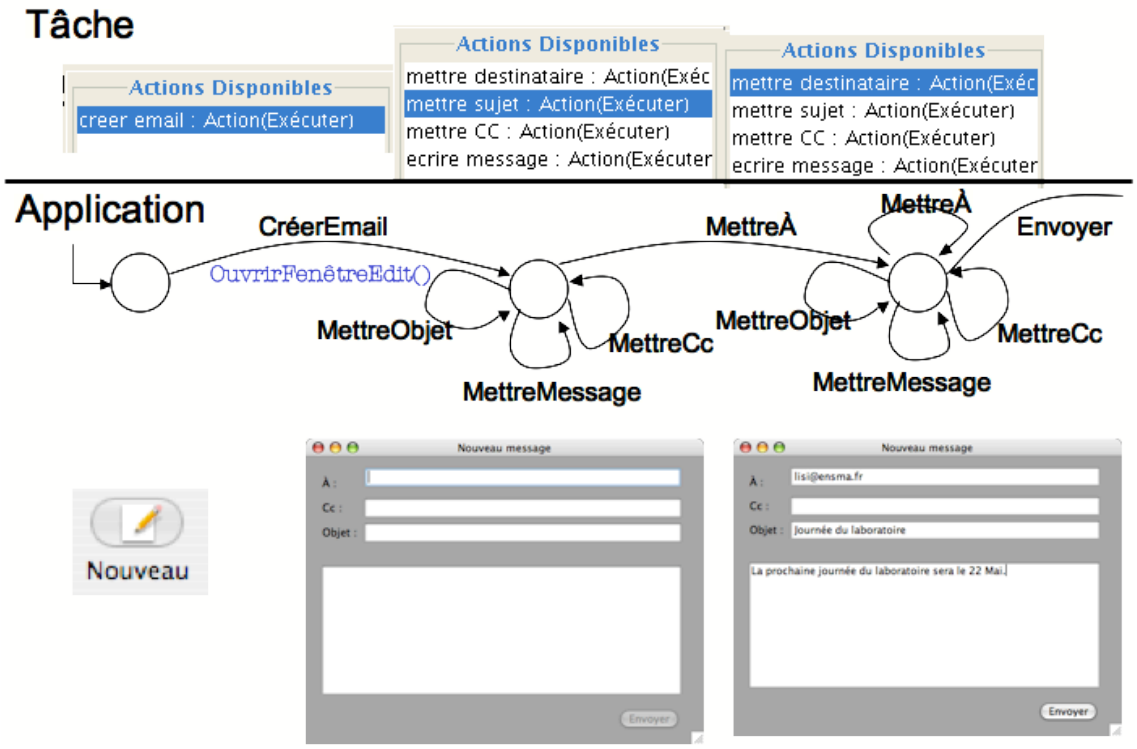


Figure 1.1.1 : Lien intuitif entre dynamique des modèles de tâches et modèle de dialogue de l'interface

L'étude proposée dans ce document a été menée pour permettre la conception et la validation de ces deux modèles de manière cohérente entre eux, tout en prenant en compte les différents paramètres de conception (dispositifs, techniques d'interaction, niveau de connaissance de l'utilisateur…).

1.2. Organisation de la thèse

Cette thèse est composée de 7 chapitres.

Le Chapitre 1 constitue l'introduction de cette thèse.

Le Chapitre 2 comporte un **état de l'art** ; Dans ce chapitre, nous montrons comment la conception centrée-utilisateur est devenue au fil des années l'approche phare pour la conception d'IHM. Nous présentons également, dans ce chapitre, le rôle des modèles dans cette conception, qui peut reposer sur les études menées dans le domaine de l'IDM.

Le Chapitre 3 décrit les **études préliminaires** qui situent notre problématique de recherche. Il a pour objectif de positionner le problème auquel nous nous attaquons dans la suite de la thèse. Nous y présentons les études préliminaires qui nous ont permis de

choisir les formalismes de modèle de tâches et de modèle de dialogue sur lesquels nous avons travaillé, et de déterminer les difficultés de conception des IHM à partir des données contenues dans les modèles de tâches.

Le corps de notre contribution originale, c'est-à-dire les études que nous avons menées pour **répondre à ce problème** en utilisant les travaux du domaine de l'Ingénierie Dirigée par les Modèles (IDM) est présenté dans les Chapitres 4, 5 et 6. Ces trois chapitres proposent de répondre à une partie de ces difficultés en se reposant sur la méta-modélisation des modèles choisis.

Nous présentons dans les chapitres 4 et 5 les travaux nous ayant permis de méta-modéliser K-MAD et le formalisme des Interacteurs Hiérarchisés (IH) qui sont les formalismes que nous avons choisis pour modéliser respectivement les tâches et le dialogue. Ces méta-modèles sont utilisés pour permettre la communication entre les deux modèles. Cette communication entre les modèles est utilisée dans une démarche de vérification en conception. Pour cela, nous procédons en plusieurs étapes :

- la définition des règles de cohérence entre les deux modèles
- l'établissement d'un méta-modèle global permettant la mise en correspondance des méta-modèles de tâches et de dialogue

Le Chapitre 6 illustre cette démarche par un exemple de conception d'application interactive : un *mailer*.

Enfin, le Chapitre 7 présente une conclusion qui fait un bilan des travaux décrits dans ce document et présente les perspectives ouvertes.

Chapitre 2. La place des modèles dans la conception des systèmes interactifs

Depuis l'intégration des applications informatiques dans le milieu industriel puis dans le cadre familial, la variété des caractéristiques d'application s'est considérablement accrue. Les interfaces des applications sont passées de langages de commande à des interfaces graphiques, de l'interaction textuelle à la manipulation directe, en passant par les techniques basées sur l'utilisation de widgets. Les interfaces graphiques se sont largement développées, aussi bien en termes de variété de plateformes cibles que d'élargissement des populations d'utilisateurs (profils utilisateur très variés : adultes, enfants, professionnels, étudiants, handicapés, etc.).

Avec la multiplication des types d'applications disponibles, les coûts de développement ont également augmenté. Pour tenter de maîtriser ces coûts de production, les concepteurs d'applications ont développé divers modèles et méthodes.

Cette section est dédiée à un état de l'art de ces méthodes et modèles. La première section présente les approches développées pour la conception centrée utilisateur (§ 2.1). Dans le domaine du génie logiciel, une approche basée sur l'utilisation de modèles en ingénierie a été développée. En parallèle, ces méthodes ont été adaptées à l'Interaction Homme-Machine (IHM). Cette nouvelle approche est présentée dans la section 2.3.

2.1. Conception orientée-système

Le domaine du génie logiciel a depuis longtemps vu l'intérêt d'utiliser les modèles pour concevoir les applications informatiques. D'après la définition de Minsky [Minsky 1988] : « To an observer B, an object A* is a model of an object A to the extent that B can use A* to answer questions that interest him about A »[1].

Suivant cette définition, différents modèles ont été développés dans le but de répondre aux différentes questions posées lors de la conception. La première section de cette partie est dédiée aux modèles développés pour la conception des applications en suivant une démarche orientée système. Basées sur ces modèles de conception, des méthodes de développement ont été conçues.

2.1.1. Modèles de développement

Depuis le début des années 70, le domaine du génie logiciel développe des modèles visant à spécifier les différentes étapes de conception informatique.

[1] Pour un observateur B, A* est un modèle de l'objet A dès lors que B peut utiliser A* pour répondre aux questions qu'il se pose sur A »

L'établissement de tels modèles a pour but de décomposer le processus de conception, de prévenir les éventuels défauts de conception, le plus en amont possible, et de faciliter la maintenance du système conçu.

Issus des premiers travaux dans ce domaine, des modèles ont été proposés pour la conception de tout type d'application. Le premier fut le *modèle en cascade*. L'étude de l'utilisation de ce modèle a mis en évidence la nécessité de prendre en compte la vérification et la validation des étapes dans les modèles de conception afin d'être le plus à même de détecter les erreurs le plus rapidement possible. Cette observation a donné naissance à la définition de plusieurs modèles raffinant le *modèle en cascade* comme le *modèle en V*. Ces modèles n'ont pas été définis pour exprimer la conception d'applications interactives.

2.1.1.1. Le modèle en cascade

Le premier modèle développé, le *modèle en cascade* (Figure 2.1.1), divise le processus de conception des applications informatiques en quatre étapes différentes réalisées les unes après les autres, en fonction de l'étape précédente (d'où l'appellation « en cascade »).

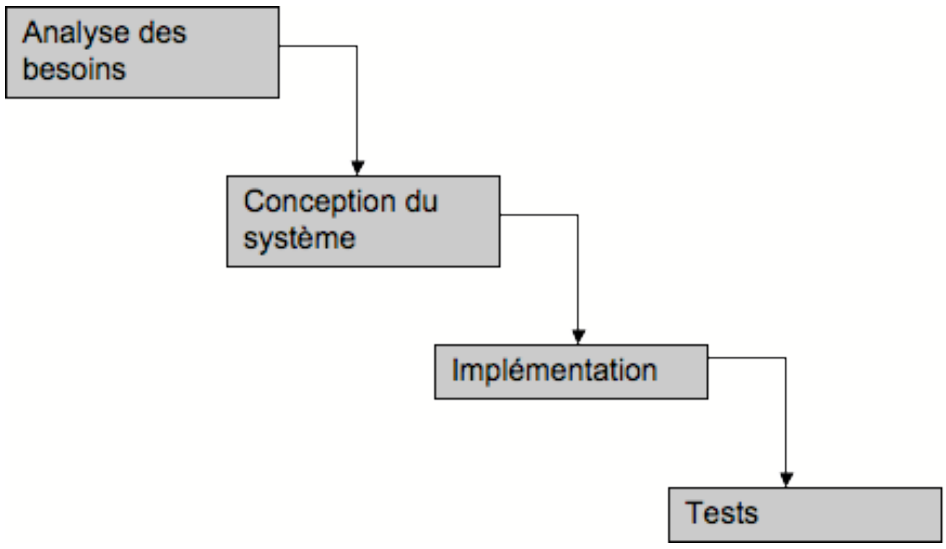


Figure 2.1.1 : Le modèle de conception en cascade

Les quatre étapes définies sont l'**analyse des besoins**, la **conception du système**, l'**implémentation** (ou mise en œuvre) et la phase de **tests**. L'étape d'analyse des besoins vise à recueillir les services requis par le système et les contraintes de développement de ce système. Lors de l'étape de conception du système, une solution matérielle et logicielle est définie répondant aux besoins recensés lors de la première étape. L'étape d'implémentation vise à coder cette solution. Enfin, des tests sont menés sur le code produit afin de fournir une solution livrable.

Suite à l'utilisation de ce modèle, des modifications ont été apportées dans le but de le rendre plus flexible et plus prompt à l'identification d'erreurs de conception. Ces modifications concernent la phase de conception du système et l'ajout de la possibilité de retour à l'étape précédente. L'étape de conception du système est ainsi subdivisée en trois étapes différentes afin d'y inclure une phase de conception globale de l'application et une phase de conception détaillée pour chacun des modules définis lors de la phase de conception globale.

Malgré l'ajout d'un retour possible à l'étape précédente, l'identification et la correction des erreurs reste un procédé difficile à mettre en place et qui arrive tard dans la conception (dernière des étapes du processus).

2.1.1.2. Le modèle en V

Le *modèle en V* (Figure 2.1.2) permet de mettre en avant la détection d'erreur le plus en amont possible lors des étapes de conception. Ce modèle reprend les principales étapes du *modèle en cascade* et y ajoute des étapes de tests correspondant à chacune des étapes de conception.

Cette approche est composée de deux pentes. La première est dédiée aux étapes de spécification, de conception et d'implémentation, et la seconde aux étapes de vérification du code obtenu. À chaque étape de spécification, une étape de vérification est associée.

La première étape est la **spécification**, c'est-à-dire l'expression, le recueil et la formalisation des besoins fonctionnels et de l'ensemble des contraintes. Une fois ces données récoltées, une étape de **conception globale** est réalisée. Cette étape consiste à mettre en place un modèle d'architecture afin de structurer l'application. Celle-ci est alors divisée en différents modules qui communiquent entre eux. Chaque module a un rôle qui lui est assigné et fait l'objet d'une **conception détaillée**. L'un de ces modules exprime la dynamique de l'application et est représenté sous forme d'un modèle de dialogue. C'est sur les modèles détaillés de ces différents modules qu'est basée l'**implémentation** de l'application.

Correspondant aux quatre étapes de la conception (sur la pente descendante du V), quatre étapes de vérifications ont été définies. La **conception détaillée** permet de mettre en place des **tests unitaires** sur chacun des modules. Une fois chaque module vérifié (plus aucun bogue n'est alors détecté lorsque les modules sont utilisés de manière isolée), les **tests d'intégration** peuvent être menés. Ceux-ci visent à s'assurer que l'intégration (et la communication) de tous les modules ensemble ne lève pas d'erreur. La dernière étape de vérification est réalisée sur l'application dans son entier. Elle correspond à la vérification de la **qualification** de l'application par rapport aux spécifications.

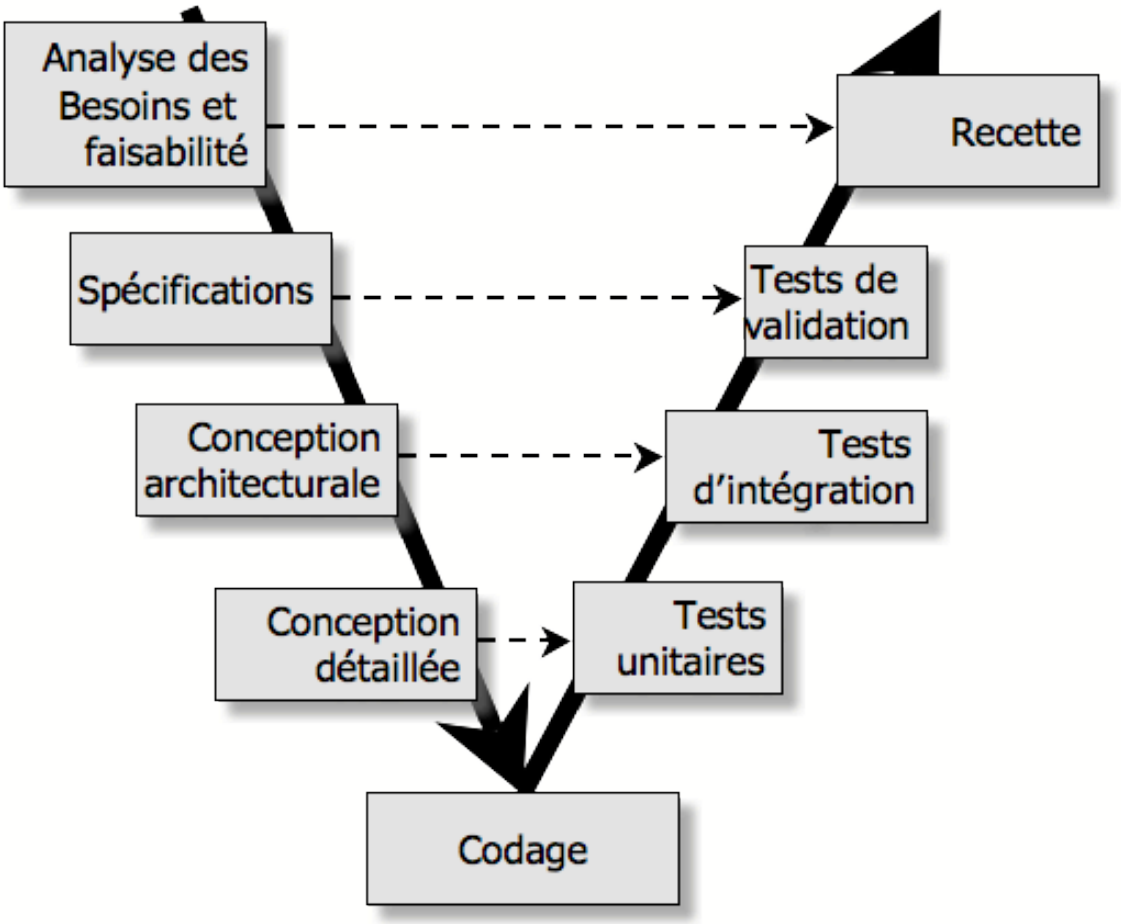


Figure 2.1.2 : Le modèle en V

Le *modèle en V* propose la prise en compte des étapes de vérification lors de chaque étape de la conception (plus en amont que dans le *modèle en cascade*). Cependant, tout comme le *modèle en cascade*, il exprime un processus de conception pour lequel chaque étape est réalisée à partir de la précédente et donc introduit un processus dans lequel la validité d'une étape dépend de la validité de l'étape précédente.

De plus, les besoins pris en compte lors de l'étape d'analyse des besoins sont les besoins du point de vue système mais ne prennent pas en compte les besoins liés à la composante « utilisateur ».

2.1.2. Les méthodes et modèles utilisés en génie logiciel

Tout l'enjeu de la conception d'application est de les développer en maîtrisant les coûts, les délais, tout en augmentant la fiabilité. Dans ce but, dès le début des années 60, des méthodes de génie logiciel ont été développées. Une ou plusieurs de ces

méthodes peuvent être utilisées pour chacune des phases de conception (§ 2.1.1) afin de supporter la conception d'un logiciel de qualité.

Le choix des méthodes utilisées est alors fait en fonction de différents critères comme le domaine d'application [Jaulent 1992]. Par exemple, pour la phase de spécification fonctionnelle du logiciel du cycle en V (§ 2.1.1.2), le concepteur peut choisir la méthode proposée par SADT [Marca et McGowan 1987] ou Merise [Tardieu, et al. 1983].

Les premières méthodes proposées sont des méthodes fonctionnelles. Elles ont pour but de soutenir la conception d'applications pour lesquelles le rôle de l'utilisateur dans l'exécution est minime.

De même que les langages de programmation sont passés du fonctionnel (comme C, ADA ou FORTRAN) à orientés objet (comme C^{++} ou JAVA), les méthodes de conception ont évolué pour proposer des approches orientées objet.

La conception basée sur ces approches repose sur les quatre principes [Graham 1991] :

- identifier les objets et le nom de leurs attributs et de leurs méthodes
- établir la visibilité de chaque objet en relation avec les autres objets
- établir l'interface de chaque objet et les exceptions manipulées
- implémenter et tester les objets

Parmi les méthodes orientées objets développées, trois (OMT, BOOCH et OOSE) ont fusionné pour donner naissance à UML (Unified Modeling Languages) [Object Management Group 1997]. L'OMG (Object Management Group) en a proposé une normalisation en 1997. Il est aujourd'hui un standard en conception logiciel.

UML 2.0 propose treize modèles (ou diagramme) différents, chacun étant une représentation graphique d'un point de vue sur le système. Les représentations graphiques manipulent des éléments dont les principaux sont représentés sur la Figure 2.1.3.

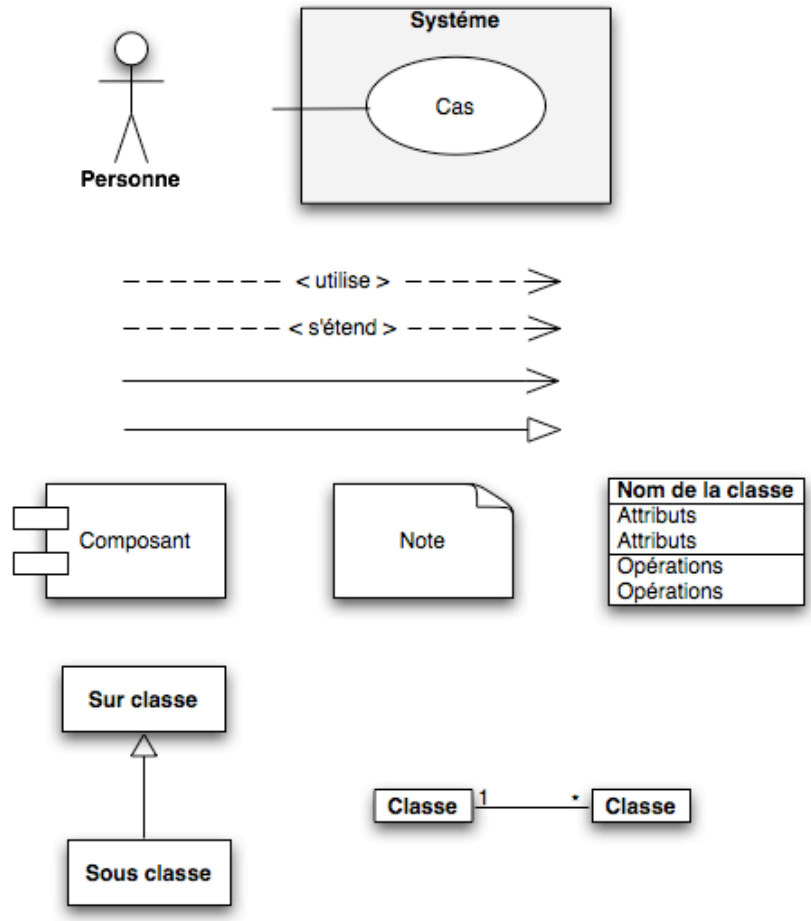


Figure 2.1.3 : Les principaux composants d'UML

Cinq vues différentes sont identifiables dans UML :

- Vue des **cas d'utilisation** : description du point de vue des acteurs du système (QUOI, QUI), comme le diagramme des cas d'utilisation.
- Vue **logique** : description du point de vue interne du système (COMMENT).
- Vue **d'implémentation** : description du point de vue de la dépendance des modules développés.
- Vue des **processus** : description du point de vue temporel et technique, comme le diagramme états-transitions.
- Vue du **déploiement** : description du point de vue physique des éléments du système (ordinateur, réseau…)

Chacune de ces vues nécessite l'utilisation d'un ou plusieurs modèles UML et un modèle peut appartenir à plusieurs vues.

Les treize diagrammes d'UML n'ont pas tous la même place dans le processus de conception. Ainsi si certains modèles sont peu utilisés, les concepteurs mettent souvent l'un des modèles au centre de la conception, comme le diagramme de classe ou celui des cas d'utilisation pour en concevoir d'autres (comme le diagramme états-

transitions). UML ne propose pas une méthode de développement mais un ensemble d'outils graphiques visant à aider à la conception de manière non formelle.

Le modèle EXPRESS a été développé pour décrire des données et permettre l'expression de règles (RULES) vérifiables sur ces données ou sur les relations définies entre elles. Le modèle graphique EXPRESS-G permet d'associer à un modèle EXPRESS, une vue graphique. Ce modèle utilisé dans la suite de nos travaux est décrit en annexe (Chapitre 9). Il permet d'exprimer l'équivalent des données du diagramme de classe UML auxquelles sont ajoutées les contraintes d'OCL.

UML et EXPRESS sont deux langages d'expression de données pour la conception des systèmes. Afin de soutenir le développement de système des outils de génie logiciel ont été développés. Ces outils permettent la vérification de preuve et l'expression suivant une manière formelle. Deux grandes familles ont été identifiées : les approches algébriques (ou fonctionnelle) et les approches non algébriques (nommées approches basée sur modèles, approches constructives, approches opérationnelles ou encore approches basées sur état explicite).

Les approches algébriques sont des approches pour lesquelles la sémantique est donnée par une algèbre. Le système est alors décrit sous forme d'équations exprimées avec des variables et des opérations issues de cette algèbre. Deux exemples de langages algébriques sont LOTOS [Systems 1984] et LUSTRE [Halbwachs, et al. 1991].

Les approches non algébriques expriment le système sous forme d'états évoluant en fonction des opérations appliquées [Aït-Ameur 2000]. Parmi les approches non algébriques développées nous pouvons citer les réseaux de Petri et B [Abrial 1996].

2.1.3. L'approche IDM (MDE) en génie logiciel

Au début des années 1980, des travaux ont été menés dans le but de développer des outils d'aide à l'ingénierie logiciel. Ces outils ont nécessité beaucoup d'efforts pour permettre la spécification de la conception sous forme graphique.

Ces représentations graphiques facilitent la communication entre les différents intervenants ainsi que le raisonnement sur une représentation abstraite (les graphes) du système.

Cependant, la multiplication des paramètres à prendre en compte pour la conception (comme les plateformes d'accueil) a rendu l'utilisation de ces outils complexe. De plus, l'utilisation de ces outils supporte mal le passage à l'échelle. Enfin, aucun lien direct en formel ne permet de passer directement des représentations graphiques à l'implémentation.

Pour pallier ces limitations d'utilisation des outils d'aide à la conception, un domaine de recherche s'est intéressé à l'utilisation des modèles pour l'ingénierie. L'Ingénierie Dirigée par les Modèles (IDM ou MDE pour *Model Driven Engineering*)

est une approche de conception qui propose d'utiliser une (ou des) représentation(s) simplifiée(s) d'aspects de l'application (modèle(s)) pour soutenir la conception.

L'idée est de mécaniser la définition des solutions qui correspondent à chacune des préoccupations de conception (fonctionnelles, sécuritaires…) et de fusionner (ou *tisser*) ces solutions pour concevoir l'application (voir Figure 2.1.4 issu de [Schmidt 2006]).

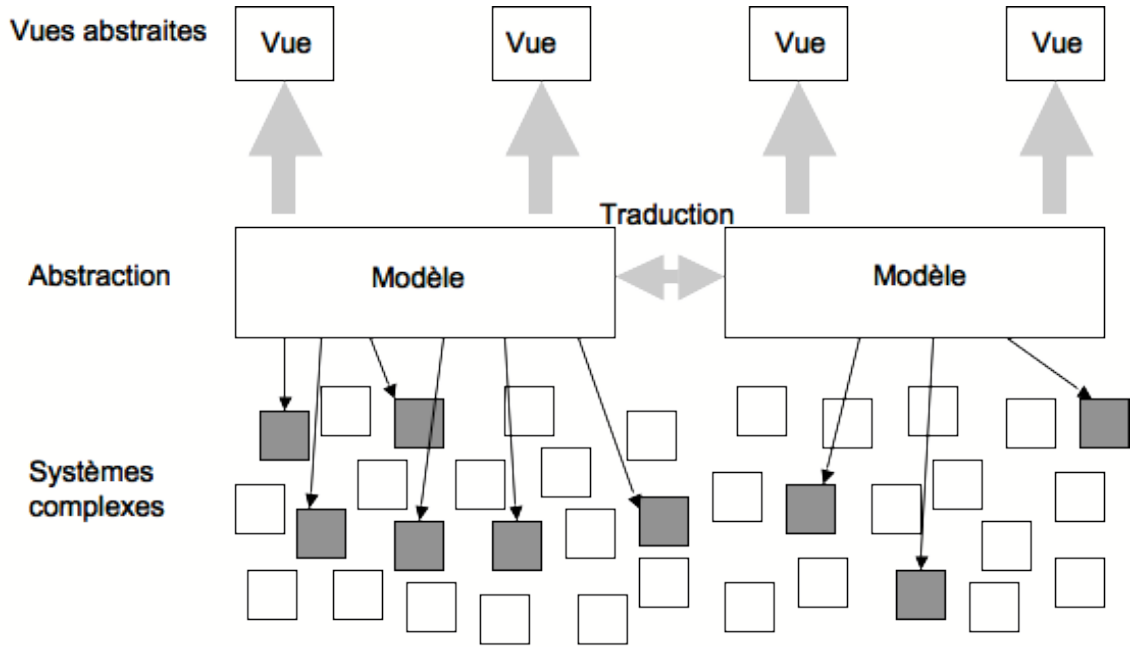


Figure 2.1.4 : Les relations entre les vues, les modèles et les méta-modèles, traduit de [Schmidt 2006].

Les préoccupations de conception sont l'ensemble des préoccupations des différents domaines d'expertise auxquels la conception du système fait appel. Ces préoccupations sont exprimées conformément à un méta-modèle.

Ces sont ces méta-modèles qui sont exploités dans la seconde étape de l'approche IDM : l'étape de transformation et de génération. Lors de l'analyse des méta-modèles exprimés dans la première phase, les éléments sont sélectionnés en fonction de leur pertinence pour l'application développée et associés selon un mécanisme de tissage (représenté par les relations entre les méta-modèles).

Le développement de cette approche a connu son extension avec l'initiative MDA (*Model Driven Architecture*) de l'organisme OMG (Object Modeling Group)[2]. Cette étude porte sur l'utilisation des travaux de l'IDM appliqués à la conception du logiciel en fonction de la plate-forme d'accueil.

[2] http://www.omg.org/mda/

Les approches de l'IDM ont principalement été utilisées pour des processus de conception génératifs. Les applications sont partiellement ou totalement obtenues après une succession de différentes transformations de modèles. Chaque transformation prend un (ou plusieurs) modèle(s) en entrée et produit un modèle en sortie jusqu'à obtention de l'application finale. Ces transformations étant réalisées par des machines, il est nécessaire que les modèles utilisés et les transformations soient explicitement (et formellement) exprimés.

Afin de les décrire, des outils commerciaux et open source ont été développés. Parmi ces outils, nous pouvons citer les outils de la famille XML (comme XSLT et Xquery) qui sont des outils de transformations génériques qui peuvent être utilisés pour les transformations particulières de modèles. D'autres outils étaient spécifiquement développés pour la transformation de modèles. Enfin, les outils de méta-modélisation sont des outils pouvant être utilisés pour exprimer les transformations de modèles. Les transformations sont alors réalisées par l'exécution d'un méta-programme orienté objet. Les deux outils majeurs pour méta-modéliser les formalismes sont le diagramme de classe d'UML et le langage EXPRESS.

2.1.4. Conclusion sur l'approche orientée-système

Dans cette section (§ 2.1), nous avons présenté les méthodes et les principales approches de conception des applications centrée-système. Ces méthodes de conception ont donc pour point central de préoccupation le système que ce soit du point de vue interne (les caractéristiques techniques font parties des spécifications desquelles découle l'ensemble du processus de conception) que du point de vue de la communication avec l'*opérateur*.

Ce type de conception ne permet pas de prendre en compte les raisons humaines de la conception. Par exemple, UML ne permet pas de représenter le POURQUOI de la conception alors que les modèles expriment le QUAND, QUOI, QUI et OU.

De plus, les processus proposés ne valident que les aspects fonctionnels sans permettre l'évaluation du rapport des humains avec la machine. Ce rapport peut être source d'erreurs qui ne sont alors pas détectées.

La conception centrée système n'est donc pas uniquement une conception dans laquelle le système joue le rôle majeur mais une conception dans laquelle l'utilisateur n'est pas considéré du tout. Cette conception n'est donc pas adaptée pour tout système ayant des aspects liés à l'utilisateur (y compris les avancées techniques comme par exemple les systèmes mixtes).

2.2. La conception centrée utilisateur

Au fil des années, la place de l'utilisation dans le processus de conception s'est faite de plus en plus importante. La validité d'une application n'est plus seulement

jugée qu'à son bon déroulement du point de vue fonctionnel mais également à sa facilité d'utilisation et de prise en main ; c'est pourquoi l'utilisateur ne veut plus être uniquement l'opérateur qui devra se plier au fonctionnement de l'application mais souhaite faire prévaloir son point de vue. Ce changement d'état d'esprit de l'utilisateur a conduit à l'émergence d'un courant de conception baptisé « centré utilisateur », dont le point d'orgue est à trouver dans la conception participative, qui prône l'intervention directe de l'utilisateur tout au long de la phase de conception.

Ce nouveau point de vue de la conception se fonde sur des études spécifiques aux quelles nous nous référerons souvent dans la suite de ce mémoire. Les études portent sur le facteur humain et ont permis de définir un nouveau type de conception, la conception centrée utilisateur (§ 2.2.1). Ce nouveau processus de conception a nécessité la définition de nouveaux modèles de conception (§ 2.2.2) et le développement de modèles spécifiques au domaine de l'IHM (§ 2.2.3).

2.2.1. Des facteurs humains à la conception centrée-utilisateur

Prendre en compte l'utilisateur dans la conception des systèmes interactifs semble aujourd'hui une évidence. Différents travaux ont été menés afin de représenter les données humaines. Parmi eux, trois sont particulièrement remarquables pour le domaine des IHM.

Le premier est le modèle du processeur humain [Card, et al. 1983]. Ce modèle est illustré sur la Figure 2.2.1. Il décompose le *processeur humain* en trois systèmes distincts :

- le système sensoriel (perception non interprétée de l'environnement)
- le système moteur (mouvement)
- le système cognitif (mémoire à court et à long terme, transformations des informations symboliques en actions)

Ce premier modèle est une tentative d'expression de données psychologiques avec une terminologie informatique. Il a conduit à la définition de modèles prédictifs d'évaluation des systèmes [Card, et al. 1983].

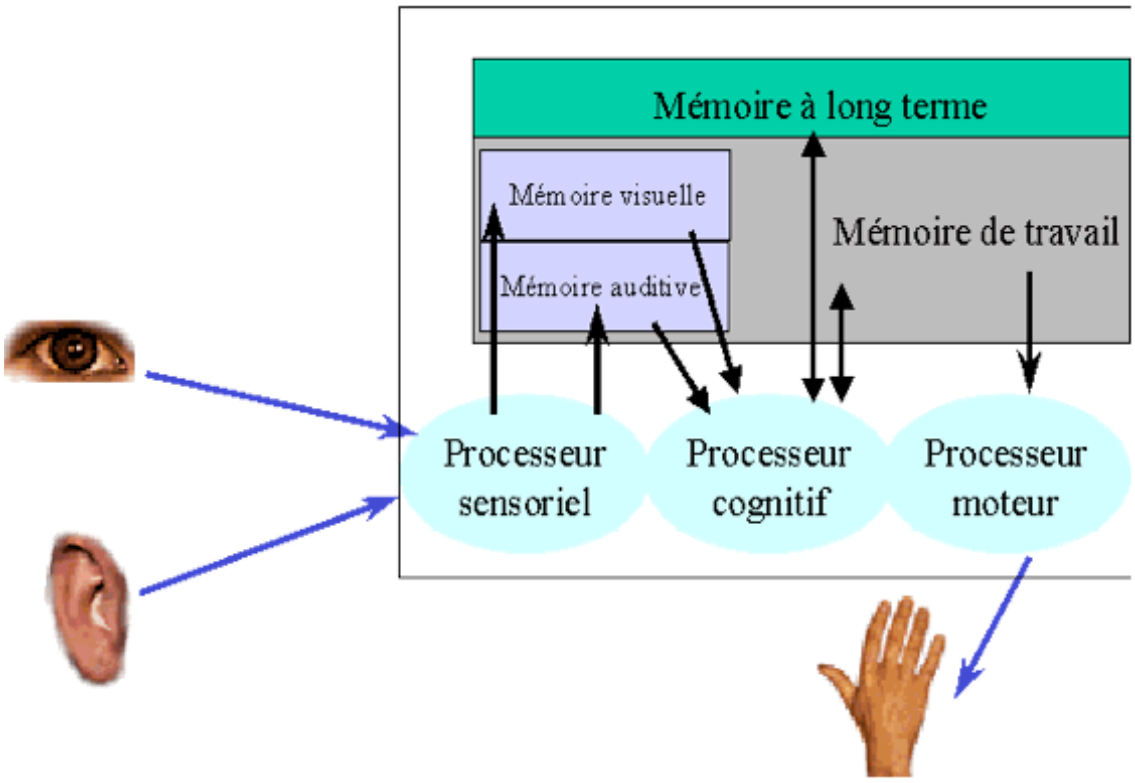


Figure 2.2.1 : le modèle du processeur humain

Une autre étude essentielle [Norman et Draper 1986] s'attaque à la notion d'activité. Elle propose une structuration des actions en suivant une décomposition des objectifs des actions en sous-objectifs (répondant à la question : « comment cet objectif est-il atteint ? »). Ces travaux précisent une méthode de structuration de l'accomplissement d'une tâche (établir un but, formation d'une intention, spécification de la suite d'actions, perception de l'état du système, interprétation et évaluation).

De plus, ces travaux définissent la notion de distance entre le système et les buts de l'utilisateur. Ces distances sont au nombre de deux (Figure 2.2.2). La distance d'exécution est la distance nécessaire au passage du but de l'utilisateur à l'expression de la commande sur le système. La seconde distance est la distance issue de la confrontation de l'interprétation du feedback du système par rapport aux buts de l'utilisateur.

La réduction des distances mise en évidence par Norman [Norman et Draper 1986] entre le système et les buts que l'utilisateur souhaite atteindre est considérée aujourd'hui comme un objectif majeur pour la conception des systèmes interactifs.

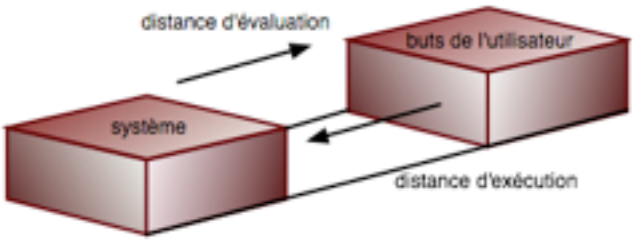


Figure 2.2.2 : Les distances entre le système et les buts de l'utilisateur

Enfin, le dernier modèle sur les travaux cognitifs que nous présentons dans cette thèse est celui de Rasmussen [Rasmussen, et al. 1994] (Figure 2.2.3). Ce modèle complète la théorie de l'action en proposant une décomposition des comportements humains en trois niveaux : les comportements basés sur les connaissances (*savoir*), les comportements basés sur les règles (*procédés*) et les comportements basés sur les habiletés (*réflexes*).

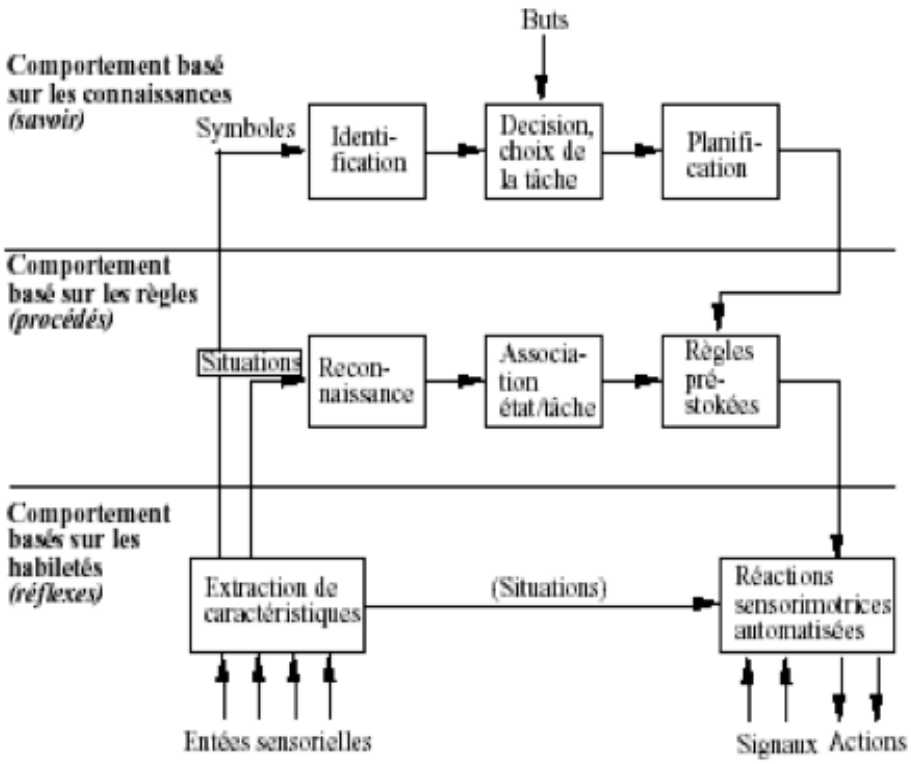


Figure 2.2.3 : Le modèle de Rasmussen

C'est dans [Norman et Draper 1986] que le terme de « *user-centered design*[3] » fut pour la première fois employé pour exprimer un type de conception dans lequel l'utilisateur de l'application (appelé « *end-user* ») influence la conception elle-même.

Dans [Norman 1988], Norman définit quatre recommandations issues de ses recherches sur l'implication de l'humain dans le processus de conception :

- à tout moment, rendre visible quelles sont les actions possibles
- rendre les choses visibles le plus possible, y compris les actions alternatives et le résultat des actions
- rendre facile l'évaluation de l'état courant du système
- suivre une relation logique entre les intentions et les actions attendues, entre les actions et les effets résultat ; et entre les informations qui sont visibles et les interprétations de l'état du système

Ces recommandations placent l'utilisateur au centre de la conception. La prise en compte des utilisateurs dans le processus de conception a permis la modifications des types d'applications conçues. La prise de conscience de ce facteur s'est particulièrement exprimée dans la conception des interfaces. Celles-ci sont passées de l'état de quasi-inexistant (communication Humains-Systèmes via des lignes de commandes) à des interfaces graphiques sur lesquelles les utilisateurs peuvent interagir [Myers 1998].

Les premières interfaces graphiques reposent sur les concepts de métaphores du monde réel et de WYSIWYG (What You See Is What You Get). Les interfaces développées sont alors caractérisées par le paradigme WIMP (Windows, Icons, Menus and Pointers).

Cependant, ce type d'interfaces a montré ses limites. C'est pourquoi de nouveaux paradigmes d'interaction ont été définis comme celui de la manipulation directe [Shneiderman 1983, Shneiderman 1998]. La manipulation directe est définie sur trois principes de base :

- rendre visible les objets d'intérêt et des actions possibles
- avoir des actions rapides, réversibles et incrémentales
- remplacer la syntaxe complexe des langages de commande par la manipulation directe de l'objet d'intérêt

Shneiderman illustre dans [Shneiderman 1998] les intérêts de la *manipulation directe* pour l'apprentissage, la productivité et la réduction des erreurs utilisateur.

L'application de ces principes d'interaction dans la conception des interfaces a introduit un nouveau type d'interfaces : les interfaces post-WIMP. Ces interfaces peuvent notamment intégrer des instruments pour faciliter la manipulation des objets du domaine via l'utilisation d'instruments ou de méta-instruments.

La multiplicité des moyens d'interaction sur les interfaces apporte de nouvelles difficultés à la conception des IHM.

[3] conception centrée-utilisateur

De plus, l'approche de conception centrée-utilisateur a également nécessité le développement de moyens pour introduire l'utilisateur dans le processus de conception ; comme la définition de méthode d'évaluation, de récolte des besoins... De nouveaux modèles ont également été développés pour les exprimer.

2.2.2. Les modèles de conception

Pour définir un modèle de conception adapté à la prise en compte des besoins utilisateur, Coutaz dans [Coutaz 1987] propose une adaptation du *modèle en V* pour y incorporer des points d'entrée du point de vue utilisateur. Nous les présentons sur la Figure 2.2.4. De plus, nous y avons ajouté des exemples des modèles conçus pour supporter chacune des étapes.

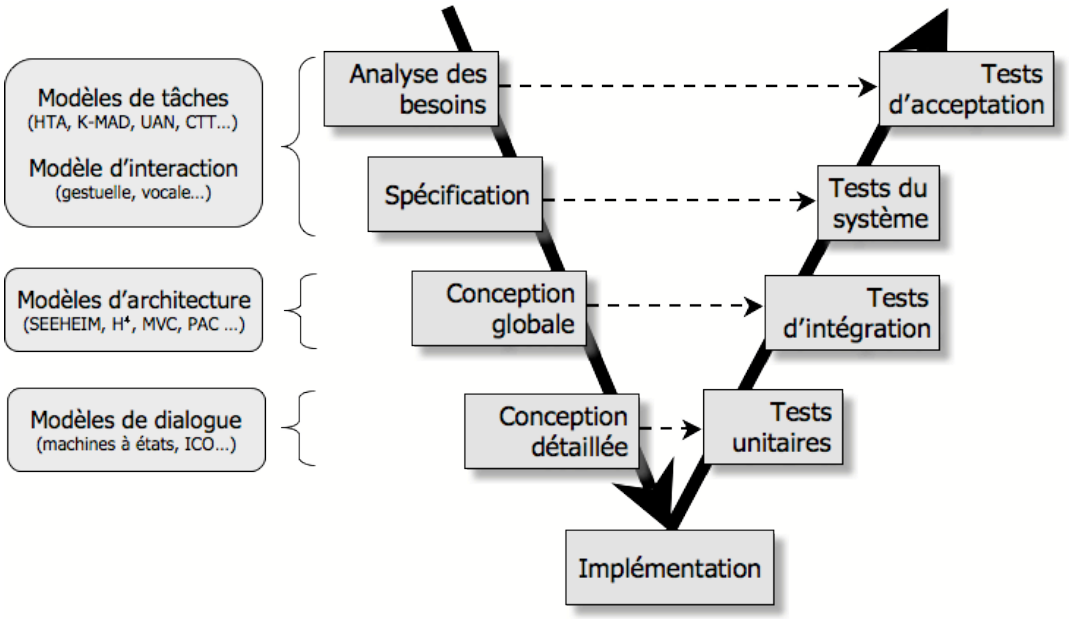


Figure 2.2.4 : Cycle en "V"

En plus de la prise en compte des besoins utilisateur, comme le préconise la norme ISO 13407 qui détermine les exigences auxquelles un projet doit répondre pour être considéré comme centré utilisateur, les modèles de conception des applications interactives doivent permettre l'expression d'un conception itérative.

Or, ce modèle en V présente un processus de conception linéaire bien que de nombreuses études aient montré que les applications interactives étaient développées par une succession d'itérations des étapes de conception, suivant donc un processus de conception itératif. Suivant cette observation, les modèles *itératifs* sont plus enclins à être adaptés à la conception des applications interactives. Parmi ces modèles, nous présenterons ici le *modèle en spirale* ou le *modèle en étoile*.

Le *modèle en spirale* cherche à proposer un processus de conception adapté à la conception itérative. Il propose un processus de conception dans lequel chacune des étapes est répétée jusqu'à l'obtention du système. Ce processus est basé sur l'analyse de risque. Chaque itération vise à supprimer les risques relevés préalablement. Le processus est réalisé tant que des risques sont relevés, il en résulte un nombre d'itération pouvant être important et donc un processus potentiellement coûteux. Le coût du modèle en spirale, ainsi que le fait que le point de départ de chaque itération soit l'analyse des risques, font de ce modèle un modèle principalement adapté à la conception d'applications pour lesquelles la sûreté de l'application est un élément primordial.

L'étude de l'utilisation de processus de conception correspondant aux modèles séquentiels et linéaires (comme les modèles *en cascade* et *en V*) a révélé que les concepteurs préféraient réaliser simultanément plusieurs étapes, choisissant leurs tâches de conception de manière opportuniste. Le *modèle en étoile* [Hix et Hartson 1993] (illustré sur la Figure 2.2.5) propose une conception centrée sur l'évaluation et pour laquelle l'ordre d'exécution des étapes n'est pas fixé.

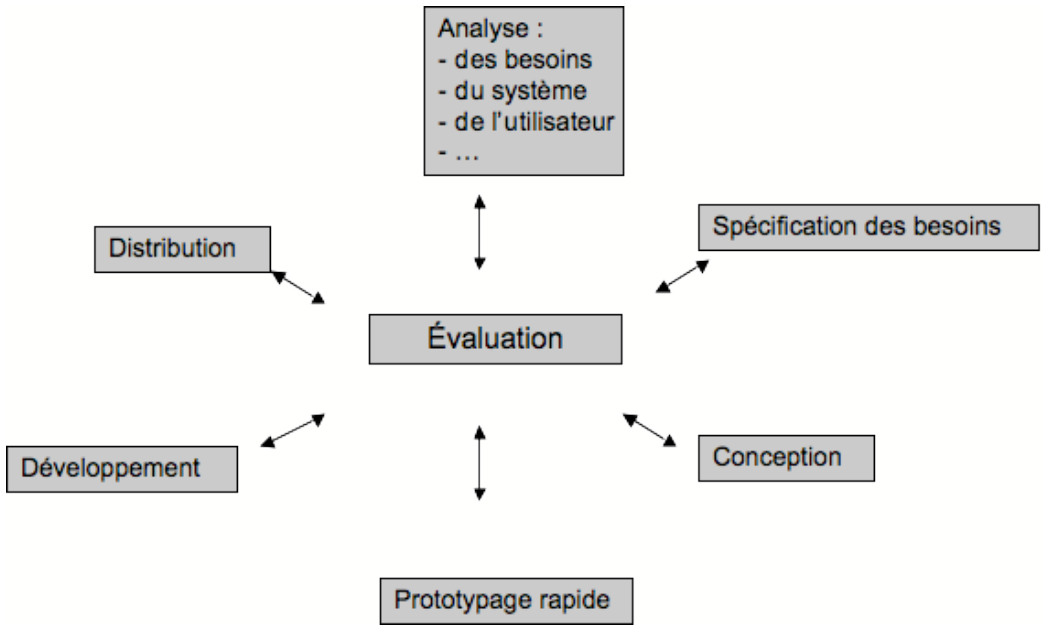


Figure 2.2.5 : Le modèle en étoile

Pour la conception des applications interactives, les modèles de conception ont permis d'identifier différentes étapes pouvant être associées à différents niveaux de connaissance. Ainsi, lors de la phase de spécification, ce sont les connaissances métiers (humaines) qui sont exploitées alors que lors de la phase d'implémentation, ce niveau de connaissance est dépassé et a permis la définition de solutions qui vont être, quant à elles, exploitées pour implémenter le système.

De plus, l'évolution des modèles de conception a accru la place faite au point de vue humain, que se soit en définissant de nouvelles étapes comme l'analyse des besoins utilisateur, ou en augmentant les phases de tests pour permettre la validation par les

utilisateurs (le plus en amont possible). En effet, l'efficience et la satisfaction de l'utilisateur peuvent être considérées dès le début de la conception du système, dans le but d'augmenter le niveau d'acceptation des applications interactives.

Des modèles ont été développés pour soutenir chacune de ces étapes (voir un exemple sur la Figure 2.2.4). Dans la section précédente, nous avons présenté des modèles sous-tendant les connaissances métiers (pour les phases de spécification et/ou s'analyse des besoins utilisateur), dans la suite nous présentons les modèles permettant la modélisation du système informatique (pour les phases plus en aval de la conception comme l'implémentation).

Parmi les modèles développés pour représenter les systèmes interactifs, nous avons choisi d'en présenter ici deux types : les modèles qui structurent les applications (les modèles d'architecture) et des modèles exprimant le lien entre les actions de l'utilisateur et la réaction du système dans l'application (le modèle de dialogue).

2.2.3. Modèles spécifiques de l'IHM

Afin d'intégrer les utilisateurs dans le processus de conception (et permettre une conception centrée-utilisateur), il est nécessaire de permettre l'expression des activités humaines. Les représentations utilisées pour l'analyse sont souvent les mêmes que celles utilisées pour les représentations des méthodes de production. Parfois, ces méthodes ont une étape de traduction, donc différentes représentations de méthodes de production peuvent être utilisées, comme les méthodes proposées par la notation UML [Object Management Group 1997]. Cependant, des notations spécifiques ont été développées pour représenter les tâches : les **modèles de tâches**. Les modèles de tâches sont issus de deux domaines de recherche : les sciences cognitives et les sciences informatiques.

Ces modèles permettent l'expression des tâches que l'utilisateur accomplit pendant l'activité modélisée. Ils permettent d'analyser les activités dans le but de concevoir une application ou de détecter d'éventuels dysfonctionnements. Des méthodes ont été proposées pour réaliser la phase de modélisation d'activités (comme la méthode du « comment ?/pourquoi ? » [Sebillotte 1991]). La décomposition des tâches des modèles est le plus souvent hiérarchique, appliquant au concept de tâches le principe de décomposition des objectifs en sous-objectifs proposé par Norman [Norman et Draper 1986].

Dans le contexte de la modélisation des tâches, plusieurs définitions du concept de **tâche** ont été proposées, nous pouvons en citer deux, issues de chacun des deux domaines :

- du domaine des sciences cognitives : une tâche est « ce qui est à faire, ce qui est prescrit » [Falzon 2004]. Dans cette définition, Falzon différencie le concept de tâche du concept d'activité qui est « ce qui est fait, ce qui est mis en jeu par le sujet pour effectuer la tâche [Falzon 2004].

- du domaine informatique : une tâche est « une activité dont l'accomplissement par un utilisateur produit un changement d'état significatif d'un domaine d'activité donné dans un contexte donné » (définition trouvé dans [Samaan 2006])

Normand dans [Normand 1992] propose une définition de la tâche dans le contexte de l'IHM. Une tâche y est définie comme étant « un but que l'utilisateur vise à atteindre assorti d'une *procédure* (ou plan) qui décrit les moyens pour atteindre ce but ([Normand 1992] lu dans [Lucquiaud 2005b]).

2.2.3.1. Les modèles de tâches

Pour répondre aux besoins d'expression de ces différentes définitions des tâches, nombre de modèles de tâches ont été développés. Sans faire une présentation exhaustive de ces modèles, nous allons en décrire certains.

Chaque modèle a été développé dans un contexte et pour répondre à un besoin spécifique. Par exemple, le modèle GOMS (Goal, Operator, Method, Selector) [Card, et al. 1983] a été défini dans un but d'évaluation. L'activité est décrite sous forme de buts à atteindre (*Goal*), d'opérations élémentaires (*Operator*), d'algorithmes utilisés pour atteindre un but (*Method*) et de règles de sélection des méthodes (*Selector*). Plusieurs versions de ce modèle ont été développées. KLM (Keystroke Level Model) est une version simplifiée de GOMS qui décrit l'activité par les opérations sur les touches. Ce modèle peut être utilisé pour évaluer le temps nécessaire à l'exécution d'une tâche.

Au contraire de KLM, NGOMS et CPM-GOMS sont des extensions du modèle GOMS. Ils permettent d'ajouter des informations aux constituants du modèle GOMS (comme le nombre d'étapes dans une méthode, les erreurs).

Ces modèles (GOMS et composés) sont des modèles dans lesquels l'activité est décrite en procédant par décomposition hiérarchique des tâches.

La description de l'activité peut être faite sous forme d'un modèle linguistique non hiérarchique comme le modèle UAN (User Action Notation) [Hartson et Gray 1992]. Ce modèle décrit chaque tâche sous forme de tableau à trois colonnes : les actions utilisateur (actions physiques exécutées par l'utilisateur), le retour d'information fourni par le système (feedback) et le nouvel état de l'interface. UAN ne permet la description des tâches qu'au niveau élémentaire. Pour combler cette limitation, une extension du modèle a été proposée : XUAN [Gray, et al. 1994]. En plus de la description de tâches non élémentaires, XUAN ajoute des pré et post conditions et des notions temporelles.

La plupart des modèles proposent une décomposition hiérarchique des tâches permettant ainsi d'obtenir une représentation graphique hiérarchisée facilement compréhensible et lisible des activités comme MAD* [Gamboa et Scapin 1997, Scapin

et Bastien 2001], CTT [Paternò, et al. 1997], GTA [van der Veer 1996] et K-MAD [Lucquiaud 2005a].

MAD* (Modèle Analytique de Description de tâcheS orienTé spécificAtion d'inteRfaces) est une extension proposée du modèle MAD [Gamboa 1998]. La Méthode Analytique de Description (MAD) [Scapin et Pierret-Golbreich 1989] est issue de travaux à l'intersection des domaines de l'ergonomie, de l'informatique et de l'intelligence artificielle. Les travaux initiaux avaient plusieurs objectifs [Gamboa 1998] : considérer les moyens par lesquels les utilisateurs représentent les tâches ainsi que la logique de data-processing ; prendre en compte les aspects conceptuels et sémantiques et non seulement les aspects syntaxiques et lexicaux ; obtenir une structuration des tâches de manière uniforme ; réaliser des descriptions à partir de points de vue déclaratifs (états du monde) ou procéduraux (moyens d'atteindre ces états) ; permettre le parallélisme et pas seulement la représentation séquentielle (synchronisation des tâches) ; et être évaluables. MAD* complète le modèle MAD en y ajoutant aux opérateurs disponibles dans MAD (et qui spécifient la décomposition des tâches) uns sémantique des objets manipulés. Ce modèle est implémenté dans les outils ALACIE [Gamboa 1998] et IMAD [Gamboa et Scapin 1997].

Certains modèles ont été développés pour exprimer des activités spécifiques comme GTA (Groupware Task Analysis). GTA [van der Veer 1996] est une méthode d'analyse de tâches qui a été développée pour la modélisation d'environnements complexes où différentes personnes interagissent avec les systèmes interactifs. Comme MAD*, ce modèle décompose hiérarchiquement les tâches.

CTT (*ConcurTaskTrees*) est défini par ses auteurs comme étant une notation pour spécifier des modèles de tâches couvrant les limitations des autres notations utilisées pour la conception des applications interactives [Paternò, et al. 1997]. Son principal but est de fournir une notation facile à utiliser, qui permet de représenter les différentes relations temporelles d'un même niveau d'abstraction et qui peut être utilisée par un novice en modélisation de tâches. CTT est une notation focalisée sur les activités.

Elle est basée sur une structure hiérarchique des tâches représentée par une structure d'arbre. Chaque tâche est graphiquement représentée dans un format de nœud d'arbre. Les relations temporelles sont exprimées par les opérateurs LOTOS [Paternò et Faconti 1992, Systems 1984] qui sont capables d'exprimer les relations temporelles entre les tâches d'un même niveau d'abstraction.

Enfin, pour palier la diversité des modèles développés une étude a été menée pour définir un modèle constitué de tous les composants proposés dans les autres modèles de tâches. Ce modèle est nommé K-MAD (Kernel of Model for Activity Description) [Lucquiaud 2005b]. Les tâches de ce modèle sont hiérarchiquement décomposées et un outil a été développé pour permettre l'utilisation du modèle.

Certains des modèles ont été introduit dans un processus de conception en les intégrant comme origine de génération comme GLADIS (une évolution de GOMS) qui est un des modèles utilisé dans ALADIN [Ricard et Pollet 1994], Diane dans la

méthodologie Diane+ [Tarby et Barthet 2001] ou TKS dans ADEPT [Johnson, et al. 1988].

Devant la diversité des modèles de tâches proposés, diverses classifications ont été proposées : on se réfèrera ici aux deux principales. Dans la première, Balbo [Balbo, et al. 2004] propose une comparaison pour aider le concepteur à choisir une notation adaptée à ses besoins. A partir de cette étude et de l'utilisation de différents modèles de tâches, une taxonomie est proposée suivant six axes : le but, l'utilisation pour la communication, l'utilisation pour la modélisation, l'adaptabilité, la couverture et l'extensibilité. La seconde classification [Limbourg et Vanderdonckt 2003] suit une approche très différente : cette comparaison des composants des modèles de tâches vise à concevoir un méta-modèle sous la forme d'un modèle entités-associations. Ces travaux visent à identifier les concepts communs des modèles de tâches. Ces deux études montrent à quel point le domaine de la modélisation des tâches est vaste et varié.

Les modèles de tâches permettent de décrire les activités en suivant différents points de vue correspondants aux besoins de leur conception (modélisation de systèmes, d'activité, support d'analyse ou de conception…). En suivant la définition des modèles de Minsky [Minsky 1988], le modèle d'un objet doit permettre à un observateur de répondre aux questions qu'il se pose sur cet objet. Cette définition sous-entend que la sémantique du modèle soit claire, et qu'une grande aide soit apportée par l'association d'outils qui doivent être capables de renforcer l'utilisation du modèle. Assurer la consistance du modèle, permettre son exploration dynamique (à l'aide de la simulation par exemple), donner l'opportunité de questionner le modèle et de comparer différentes conceptions, faciliter les liens entre les différents modèles, sont autant de point qu'un outil doit permettre de réaliser et ce, à partir d'une sémantique non-ambiguë qu'il exploite.

Les modèles de tâches ont cependant vite montré leurs limites, en particulier du fait de la séparation entre modèles de tâches et objets de l'application. Pourtant, depuis de nombreuses années, la prise en compte d'objets dans ces modèles à été jugée nécessaire à une description de l'activité suffisamment détaillée pour permettre une conception complète à partir du modèle de tâches [van Welie, et al. 1998]. Malgré cela, ils sont rarement exploités lors de la conception du système, en particulier en raison de leur non prise en compte par les outils associés aux modèles de tâches. Seuls quelques travaux les utilisent pour définir les objets interactifs [Pribeanu et Vanderdonckt 2002], mais à un niveau d'abstraction très bas, ne permettant pas de vérification lors des phases de conception les plus abstraites.

De plus, ces études montrent également que ces modèles sont peu formels comme nous l'illustrerons dans la section 3.1.

Enfin, il est à noter que les auteurs [Barthet 1988] distinguent deux types de modèles de tâches : les modèles de tâches prescrits et les modèles de tâches effectifs (ou réels). Les modèles de tâches prescrits sont composés des tâches telles qu'elles sont prévues par le concepteur alors que les modèles de tâches effectifs expriment les tâches telles qu'elles sont réellement exécutées par l'utilisateur.

2.2.3.2. Les modèles d'architecture

Les modèles d'architecture ont été développés pour structurer l'application développée. Ils permettent de décomposer l'application en plusieurs modules ayant chacun un rôle qui lui est attribué. Cette décomposition peut suivre trois procédés distincts (développés dans cette section) : une décomposition de l'application dans son ensemble (modèles globaux), une décomposition bi-axiale (modèle multi-agents) et une décomposition combinant les avantages des deux premières décompositions (modèles hybrides).

Les modèles globaux. Les modèles globaux structurent l'application en la considérant dans son ensemble. Ils permettent d'établir l'architecture d'une application en fonction des différentes facettes jouées par le code.

Le premier modèle d'architecture ayant été défini est le modèle de Seeheim [Pfaff 1985]. Celui-ci, conçu en 1983, trouve son origine dans la nécessité de remplacer les commandes textuelles par les interfaces graphiques sans avoir à re-définir le noyau fonctionnel des applications existantes. Il compose l'interface en trois modules (en plus du noyau fonctionnel) : l'adaptateur au noyau fonctionnel, qui contient les fonctionnalités permettant d'appliquer les fonctions du noyau fonctionnel à cette interface ; la présentation, qui est la partie dédiée à l'interface graphique de l'application ; et enfin le dialogue, qui permet de lier les deux autres parties en maintenant la cohérence entre les deux. Ce modèle a permis la définition du vocabulaire qui est utilisé dans l'ensemble des autres modèles. Cependant, il ne précise pas comment réaliser les différents modules ni comment les liens entre les différents modules doivent être implémentés.

Afin de tenir compte des évolutions techniques, le modèle Seeheim a été complété donnant naissance au modèle ARCH (Figure 2.2.6). Ce modèle est composé des mêmes modules que Seeheim auxquels est ajouté un autre module et le noyau fonctionnel qui est intégré au modèle d'architecture. Ce module ajouté est la « boîte à outils » (Toolkit) sur laquelle repose l'implémentation de la présentation.

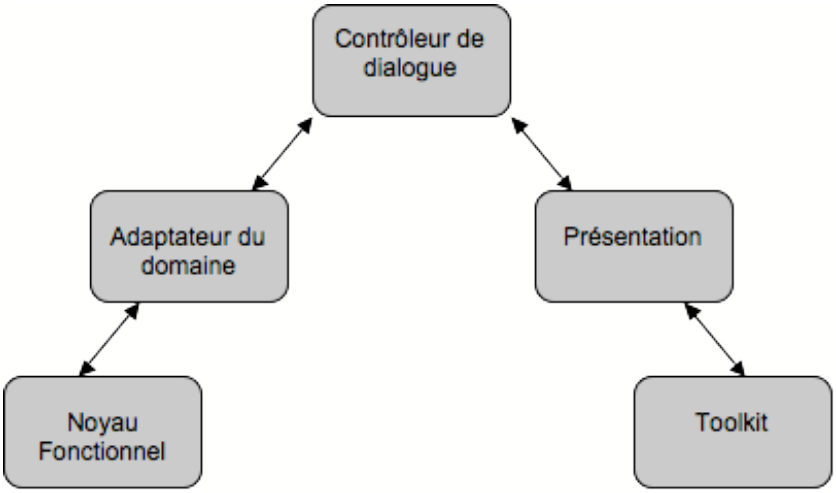


Figure 2.2.6 : Le modèle d'architecture ARCH

Bien que le modèle ARCH permette de prendre en compte les avancées dans le domaine (et est, de ce fait, plus utilisé), il a les mêmes inconvénients que le modèle de Seeheim : il ne précise ni la structure interne à chaque module, ni les méthodes d'échanges entre eux.

Les modèles multi-agents. Les modèles multi-agents procèdent en réalisant deux décompositions successives de l'application suivant deux axes. Tout d'abord une décomposition en agent (d'où le nom), c'est-à-dire que l'application globale se compose d'un ensemble de *packs* correspondant chacun à un concept (objet). Ces *packs* communiquent entre eux et sont décomposés en trois modules chacun remplissant un rôle déterminé. Les définitions de ces modules sont très proches de celles des modules de modèles globaux.

Ainsi, en suivant le modèle PAC chaque agent est décomposé en trois parties : la présentation, l'abstraction et le contrôle. La présentation du modèle PAC définit le comportement perceptible de l'agent c'est-à-dire les aspects visuels, sonores… ainsi que la réception des actions des utilisateurs. L'abstraction d'un agent PAC définit les fonctionnalités de l'agent (indépendamment de l'interaction). Cette partie correspond au noyau fonctionnel du modèle globale Seeheim. Enfin, la cohérence entre la Présentation et l'Abstraction d'un agent PAC est assurée par son Contrôle. L'ensemble des agents PAC d'une application communiquent via leur Contrôle, qui est donc également en charge de la cohérence entre les différents agents de l'application. C'est pourquoi la partie contrôle est indispensable de tout agent PAC alors que les deux autres modules peuvent être absents d'un agent PAC.

Les modèles multi-agents sont apparus avec les langages de programmation objets et sont aujourd'hui les modèles sous-jacents des langages utilisés pour la programmation d'interface. Par exemple les objets de la librairie Java/Swing sont implémentés via le modèle d'architecture Modèle-Vue-Contrôleur (MVC).

Tout comme le modèle PAC, le modèle MVC (Figure 2.2.7) propose une décomposition en trois parties. Cependant, dans ce modèle, le contrôleur n'a pas le même rôle que le contrôle de PAC ou de Seeheim. Le contrôleur de MVC contrôle les entrées des utilisateurs alors que la cohérence de l'application est assurée par la vue (pour la partie graphique) et le modèle (pour la partie sémantique).

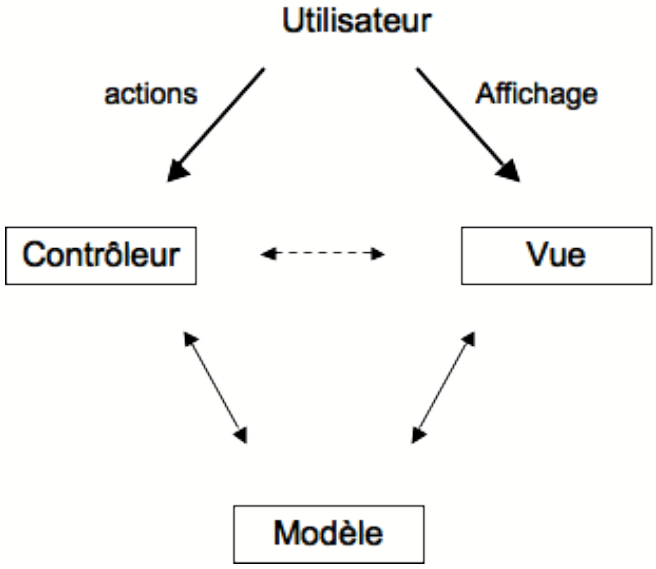


Figure 2.2.7 : Le modèle d'architecture MVC

Les modèles multi-agents permettent de proposer une structure interne de chaque agent de manière précise cependant, aucun de ces modèles ne donne de directives sur l'organisation globale des agents entre eux, c'est-à-dire sur la structure globale de l'application dans son ensemble.

Les modèles hybrides. Les modèles globaux offre une structuration globale de l'application mais pas de chacun des modules la constituant alors que les modèles multi-agents proposent une structuration de chaque agent mais pas de l'application dans son ensemble. Afin de tirer parti des avantages de ces deux types de modèles, les modèles hybrides ont été développés.

Les modèles hybrides sont des modèles qui globalement respectent un des modèles globaux (ou toute autre structuration proposant de « modulariser » l'application dans son ensemble) et pour lesquels chacun des modules issus de la décomposition globale est structuré, le plus souvent en suivant un autre modèle d'architecture.

Parmi les modèles hybrides existants, nous pouvons citer PAC-SEEHEIM [Coutaz et Nigay 1991] qui applique le modèle global Seeheim et le modèle multi-agents PAC sur chacun des modules du modèle de Seeheim.

Un autre modèle d'architecture multi-agents, nommé H^4 [Depaulis, et al. 2006] (Figure 2.2.8), est basé sur le modèle global ARCH (bien que les noms des modules différent sensiblement) pour lequel les cinq modules sont précisément décrits. H^4 propose une structure pour quatre des cinq modules du modèle : le noyau fonctionnel, le contrôleur de dialogue, l'adaptateur de présentation et la présentation. Ces quatre modules sont structurés de manière hiérarchique, chaque niveau étant une spécification du niveau d'abstraction supérieur.

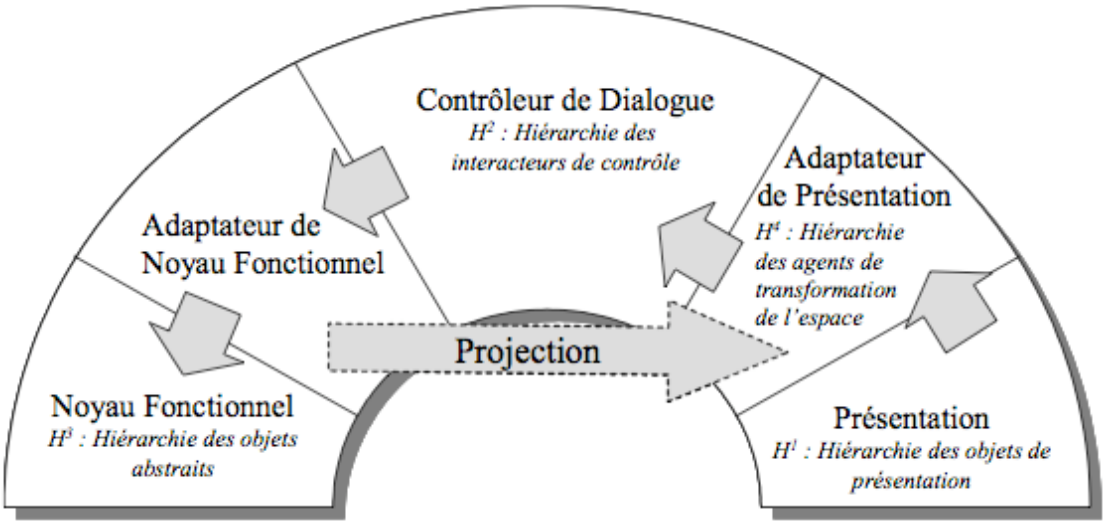


Figure 2.2.8 : Les différentes couches du modèles H[4] (issu de [Depaulis, et al. 2006]).

2.2.3.3. Les modèles de dialogue

Selon la classique définition introduite par le modèle de Seeheim, la dynamique d'une application interactive est contrôlée par la syntaxe du langage d'interaction utilisé. Dans ce modèle, cette fonction est dévolue à un composant logiciel dénommé *dialog control*, traduit généralement par *contrôleur de dialogue*. Il est décrit dans [Dix 1991] comme une double liaison :

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

Au cœur de l'interaction, le *dialogue* a été l'objet de nombreuses études. Ainsi, un nombre important de formalismes ont été utilisés pour décrire le dialogue [Olsen 1992]. L'ensemble de ces formalismes peut être classé en trois groupes [Green 1986] :

- les formalismes basés sur les grammaires [Olsen 1992],
- les formalismes à base d'événements [Green 1986] (utilisés dans la plupart des boîtes à outils comme Java/Swing),
- les formalismes basés sur les états tels que les machines à états [Jacob 1982], les *statecharts* [Harel 1987], ou les réseaux de Petri de haut niveau [Palanque 1992, Sibertin-Blanc 1985].

Les formalismes basés sur les grammaires. Une grammaire est un ensemble de quatre éléments :

- un alphabet de symboles terminaux,
- un alphabet de symboles non-terminaux,

- un symbole non-terminal particulier nommé axiome, qui est le symbole par lequel l'analyse commence
- un ensemble fini de règles qui permettent de vérifier qu'une séquence d'événements (entrés par l'utilisateur) est correcte et d'appeler les actions correspondantes.

Par exemple, les règles présentées sur la Figure 2.2.9 sont un extrait des règles de la grammaire exprimant le dialogue d'un *mailer*. Cet extrait est particulièrement centré sur le dialogue relatif à l'envoi d'un email. Les non-terminaux sont en majuscule alors que les terminaux sont en minuscule. Le symbole « | » permet d'exprimer le choix et les actions à réaliser sont entre accolades.

Le dialogue d'une application interactive est spécifié comme une séquence de terminaux. Les non-terminaux ont pour rôle de factoriser des sous-dialogues. Dans les grammaires, l'itération est réalisée à l'aide de la récursivité, c'est le cas de MODIFIER_BROUILLON à la fin des deux premières règles de MODIFIER_BROUILLON.

```
MAILER := DIALOG_ENVOI | DIALOG_CONSULTER | QUITTER
DIALOG_ENVOI := IDENTIFIER_EMAIL envoyer {send();}
IDENTIFIER_EMAIL :=  NOUVEL_EMAIL | EMAIL_BROUILLON
EMAIL_BROUILLON := select_brouillon {afficher();} MODIFIER_BROUILLON
MODIFIER_BROUILLON := modifier_dest{mettreDest();}MODIFIER_BROUILLON
      | modifier_message{mettreMessage();}MODIFIER_BROUILLON
      | ∅
```

Figure 2.2.9 : Extrait des règles de la grammaire d'un mailer

La grammaire présentée sur la Figure 2.2.9 indique que l'utilisation du MAILER se fait à travers DIALOG_ENVOI ou DIALOG_CONSULTER. La partie du dialogue illustrée sur la Figure 2.1.10 ne présente pas la règle de DIALOG_CONSULTER. DIALOG_ENVOI n'a qu'une seule production : *IDENTIFIER_EMAIL envoyer {send() ;}*, cela signifie que le seul terminal accepté à la fin d'une séquence est *envoyer* et que l'action qui lui correspond est *send()*. Le premier élément de la production de DIALOG_ENVOI est le non-terminal IDENTIFIER_EMAIL. Ce sont les règles correspondant à ce non-terminal qui sont alors applicables. Donc le premier terminal possible est celui issu de l'application de la règle IDENTIFIER_EMAIL. Celle-ci est constituée de deux non-terminaux NOUVEL_EMAIL et EMAIL_BROUILLON. Ce sont les règles de ces deux non-terminaux qui s'appliquent alors. Seul la règle de EMAIL_BROUILLON est décrite dans cet extrait. La règle de EMAIL_BROUILLON est composée d'un terminal *select_brouillon{afficher() ;}*, qui est donc le seul non-terminal par lequel une séquence de dialogue peut commencer (en fonction de l'extrait de la Figure 2.14) et du non-terminal MODIFIER_BROUILLON. Après le terminal *select_brouillon,* les terminaux possibles sont donc modifier_dest et modifier_message. Les actions correspondantes sont alors réalisées (*mettreDest()* et *mettreMessage()*). L'action correspondante est alors réalisée et la prochaine règle applicable est celle correspondant au non-terminal MODIFIER_BROUILLON (pour exprimer le caractère itératif). Enfin, le non-terminal MODIFIER_BROUILLON peut être remplacé par le terminal « ∅ » pour mettre fin à l'itération.

La spécification du contrôleur de dialogue à l'aide d'une grammaire ne permet comme vérification que celle de la validité d'une séquence de terminaux. Ainsi, la séquence « select_brouillon modifier_message envoyer » est une séquence d'actions autorisées par les règles de grammaire du mailer de l'exemple. À l'inverse, « select_brouillon envoyer modifier_dest » est une séquence non supportée par cette grammaire et par conséquent un dialogue impossible. Cependant, ce mode de représentation du dialogue ne permet pas de s'assurer qu'un email contient obligatoirement une adresse de destinataire pour pouvoir être envoyé.

Des précisions sur les grammaires sont disponibles dans [Olsen 1992].

Les formalismes à base d'événements. Contrairement aux modèles présentés précédemment, les modèles à base d'événements ne décrivent pas les différents états dans lesquels le système peut se trouver, mais la manière d'y parvenir.

Lorsqu'un événement se produit, il est traité. Un gestionnaire d'événement le dirige alors vers une procédure de traitement qui, en fonction de ces paramètres, exécute un traitement particulier.

Ce modèle est utilisé par de nombreux langages de programmation, comme Java/Swing[4] car il possède un important pouvoir d'expression et d'exécutabilité. Cependant, les modèles à base d'événements ne permettent pas la vérification de propriétés ni la représentation explicite des états du dialogue.

Dans son étude sur le sujet, Green [Green 1986] concluait sur la supériorité du modèle événementiel pour décrire la dynamique des systèmes interactifs, ce qui s'est vérifié pour la programmation par la généralisation de ce modèle dans les boîtes à outils. Cependant, plusieurs points tendent à remettre en cause cette suprématie, ou tout au moins à envisager une définition à deux niveaux. Par exemple, la généralisation des approches à bases de modèles (MDA, IDM [mde]), qui s'appuient le plus souvent sur les modèles définis autour de la norme UML [Object Management Group 1997], font la part belle aux formalismes graphiques, et utilisent extensivement les formalismes à états.

Les formalismes à base d'états. Une machine à états est le formalisme à base d'états qui a, le premier [Jacob 1982], servi à représenter le dialogue d'une application. Depuis, de nombreux modèles à base d'états ont été développés pour exprimer le dialogue des applications interactives. Dans la suite de cette section, nous explicitons l'usage général de ces modèles dans le contexte des IHM, et nous reviendrons plus spécifiquement sur les formalismes concrets dans le chapitre suivant.

Les machines à états appelées aussi automates à états ou systèmes de transition étiquetés (STE) sont exprimables sous forme graphique. De plus, elles sont basées sur des modèles mathématiques ; par conséquent, elles permettent le raisonnement et donc, la validation et la vérification de nombreuses propriétés du dialogue dans les IHM telles que les propriétés de sûreté ou de vivacité.

[4] Sun Microsystems : http://www.sun.com

Une machine à états permet de représenter le dialogue d'une application par un ensemble de quatre éléments :

- un ensemble d'états
- un état initial
- un ensemble de transitions
- un ensemble d'étiquettes associée à ces transitions

Les transitions permettent de passer d'un état à un autre et sont activées par des événements tels qu'un mouvement de la souris ou la fin d'une horloge. À chaque transition peuvent être associées des gardes et des actions.

Une garde est une condition de réalisation d'une transition. Elle doit être évaluée à « vrai » pour que la transition soit exécutée. Dans le cas où une garde n'est pas précisée, elle est implicitement vraie.

L'action d'une transition est le code associé à cette transition.

À chaque état, deux actions peuvent être associées, la première nommée *action d'entrée,* est réalisée lorsque l'état devient l'état courant, et la seconde *action de sortie*, est exécutée lorsque l'état cesse d'être l'état courant.

Le fonctionnement d'une machine à état se déroule de la manière suivante. Lorsqu'un événement se produit, s'il existe une transition correspondant au traitement de cet événement et partant de l'état courant avec sa garde à vrai, alors les actions : *de sortie* de l'état courant, *de la transition, d'entrée* du nouvel état courant sont exécutées. Sinon, l'événement est ignoré.

L'état courant au début de l'exécution de la machine à état est l'*état initial*.

De plus, un automate peut être représenté graphiquement, chaque état étant représenté par une ellipse et chaque transition par une flèche. L'état initial est, quant à lui, spécifié par une petite flèche sur le côté gauche de l'ellipse. La Figure 2.2.10 est la représentation graphique de la machine à état exprimant le dialogue de l'envoi d'un email sauvegardé dans les brouillons.

Au début de l'exécution de la machine à états, l'état courant est donc l'état nommé *init* (état spécifié par la flèche à gauche). Lorsqu'un email en brouillon est sélectionné, les gardes des transitions déclenchables par cet événement à partir de l'état initial sont évaluées (ici les gardes de *selectBrouillonSans* et *selectBrouillonAvec*). Si l'email sélectionné a une adresse de destinataire (valide ou non) alors la garde de la transition *selectBrouillonAvec* est évaluée à vrai et l'état courant devient *avecDest* sinon, c'est la transition *selectBrouillonSans* qui est exécutée est l'état courant devient *sansDest*. À partir de celui-ci deux transitions sont exécutables *modifMessage* (qui ne change pas l'état courant) et *mettreDest* qui met le système dans l'état *avecDest*. De cet état, quatre transitions sont déclenchables : *modifierMessage* et *modifierDest* (qui ne modifient pas l'état courant) et *supprimerDest* et *envoyer*. La transition *supprimerDest* est déclenchée lorsque l'adresse du destinataire est supprimée, l'état courant redevient alors *sansDest*. Enfin, *envoyer* est déclenchée lorsque l'envoi de l'email est déclenché. Le système revient alors dans son état initial (avant qu'un email n'ait été sélectionné).

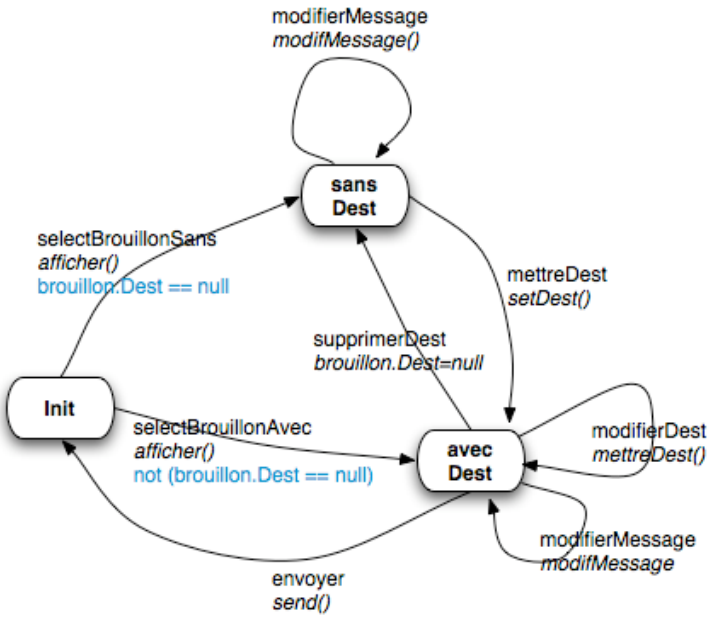


Figure 2.2.10 : Machine à état du dialogue de l'envoi d'un brouillon avec un mailer

Le formalisme des machines à états permet une représentation graphique du dialogue d'une application. Cependant, lorsque le dialogue à formaliser est dense, la machine à états produite est d'une taille importante ce qui la rend illisible.

Dans les grammaires, les non-terminaux ont la même utilité que les sous-dialogues dans les machines à états (voir le paragraphe précédent), alors que les terminaux sont utilisés dans l'ordre dans lequel ils apparaissent et sont traités directement.

De plus, depuis le milieu des années 2000, les machines à états peuvent être directement implémentées dans l'application grâce à une bibliothèque développée : SwingStates [Appert et Beaudouin-Lafon 2006, Appert et Beaudouin-Lafon 2006]. Cette bibliothèque repose sur une implémentation en JAVA/Swing.

Enfin, pour exprimer les techniques d'interaction utilisées pour permettre la communication entre l'utilisateur et le système (les moyens d'entrées des commandes des utilisateurs) différentes techniques d'interaction peuvent être utilisées. Les différents aspects des paradigmes d'interaction (du point de vue de l'utilisateur) sont détaillés sous forme de *modèles d'interaction*.

Beaudoin-Lafon définit dans [Beaudouin-Lafon 1997] un modèle d'interaction comme étant « un ensemble de principes, de règles et de propriétés qui guident la conception d'une interfaces ». Ces modèles ne sont pas définis en suivant de formalisme d'expression.

2.2.4. Conclusion sur la conception centrée utilisateur

En conclusion, dans cette section, nous avons décrit des approches de conception des applications. Les premières approches étaient dédiées à une conception orientée système puis avec le développement de la diversité des applications interactives des approches orientées utilisateur ont été développés.

Nous avons décrit quelques-uns des modèles utilisés dans la conception d'Interfaces Homme Machine développés pour les besoins de la prise en compte de l'utilisateur dans la conception des IHM. Ces modèles sont les modèles décrivant le processus de conception et des modèles permettant de décrire les deux points de vue de la conception d'un système interactif : le point de vue humain (modélisé par les modèles de tâches) et le point de vue système (en partie modélisé par les modèles de dialogue).

Lors de la conception, d'autres modèles peuvent être utilisés comme les modèles de tâches ou les modèles de dialogue. L'aspect itératif de la conception implique donc que ces modèles puissent être itérativement modifiés tout au long de la conception.

De plus, l'importance du besoin de validation exprimé dans les modèles de conception met en lumière le besoin de validation des modèles utilisés dans le processus de conception.

Le point de vue de l'utilisateur sur le système en conception peut être exprimé par des modèles de tâches. Plusieurs notations ont été développées pour remplir ce rôle. Ceux-ci représentent l'activité supportée de manière hiérarchique. Cette représentation correspond à la manière itérative dont sont conçues les décompositions des tâches. L'étude des modèles de tâches présentée dans montre que ces modèles manquent généralement d'aspects formels pouvant être exploités pour la validation du modèle et que les outils-support de ces modèles sont souvent incomplets. Une étude antérieure [Lucquiaud 2005b] a eu pour but de développer un modèle (K-MAD) et son outil (K-MADe) qui comblent ces deux points. Pour cette raison, K-MAD (utilisé avec K-MADe) est donc le modèle le plus adapté à une utilisation formelle du modèle de tâches, lors de la conception.

Enfin, les derniers modèles présentés dans cette section sont les modèles représentant le point de vue du système. L'utilisation de modèles d'architecture pour structurer l'application impose que tous les modules communiquent entre eux et aient une définition non ambiguë de leur rôle.

L'un des modules a pour but de gérer la dynamique de l'application : le dialogue. Celle-ci peut être exprimée sous forme de modèles –nommés modèle de dialogue- formels réalisés à partir des spécifications (dont une partie est exprimée sous forme de modèles de tâches). Cependant, leur conception et leur validation sont, le plus souvent, réalisés « à la main » par le concepteur/développeur.

Cette thèse a pour but l'étude de l'exploitation des aspects formels des modèles pour la conception du dialogue à partir des modèles de tâches en prenant en compte l'aspect itératif du processus de conception.

2.3. IDM et IHM

Pour concevoir les applications interactives, certaines approches proposent de se baser sur les modèles. Cette approche n'est pas nouvelle pour la conception d'IHM pour laquelle les concepteurs ont depuis longtemps cherché à exprimer leurs connaissances par des modèles. Cependant, les approches basées sur modèles visent à automatiser et formaliser le processus de conception.

Les premières approches proposées furent les *model-Based Systems* qui proposaient d'inclure l'utilisateur dans le processus de conception en lui faisant manipuler les modèles (§ 2.3.1).

Avec la multiplication des plateformes d'exécution des interfaces, les concepteurs d'IHM ont vu dans l'utilisation des modèles le moyen d'exprimer de manière générique les concepts de l'application tout en spécifiant les composants particuliers de chaque plateforme. Pour relever ce challenge, les approches de conception dirigée par les modèles (Model-Driven Approaches) proposent d'exploiter les modèles par l'utilisation de méta-modèles, un langage et un formalisme d'expression[5]. L'automatisation de la sélection des composants particuliers pour la réalisation de chaque interface adaptée a donné naissance aux approches *génératives* (§ 2.3.2).

Enfin, de nouvelles approches visent à utiliser l'IDM pour délaisser la générations des interfaces au profit d'une conception dans lesquelles le concepteur est plus présent (et de ce fait, les connaissances de conception également). Suivant cette démarche, le projet CAMELEON propose un cadre conceptuel adapté à la spécification d'interfaces qui s'adaptent à la plateforme d'accueil (§ 2.3.3).

2.3.1. Les Approches Basées sur Modèles (MBA)

De par la multiplicité des plateformes, des techniques d'interaction et des publics visés, les systèmes informatiques sont de plus en plus complexes. Parallèlement à cela, le domaine industriel souhaite réduire les coûts de conception et le temps de développement des applications.

Afin de maîtriser cette complexité, les chercheurs en conception d'IHM, se sont tournés vers les travaux menés dans le domaine des modèles, en faisant de l'IDM « *sans le savoir* » (voir la section 2.1.3). Les premières études en conception qui proposent d'utiliser les modèles sont des travaux sur la génération des interfaces à partir de description de haut niveau d'abstraction. Le domaine de ces travaux est nommé *Model Based Systems* [Girard 2000], *Model Based Development* [Luyten 2004] ou *Model-Based user interface Design* [Pinheiro da Silva 2000, Puerta 1997, Szekely 1996]. Tous ces termes peuvent être regroupés sous le terme de *Model Based Approach* (MBA).

[5] http://ormsc.omg.org/mda_guide_working_page.htm

2.3.1.1. Principe

Les approches basées sur modèles (MBA) sont composées de plusieurs éléments, le principal étant le modèle. Une structuration de ce modèle par niveau d'abstraction (Figure 2.3.1) a été défini dans le cadre de référence CAMELEON voir 2.3.3). Bien que des variations peuvent être observées sur la définition ou le contenu de chacun des niveaux d'abstraction, cette structuration est largement adoptée dans le domaine de la conception d'IHM. Ce modèle est composé de trois niveaux d'abstraction :

- les modèles de tâches et de domaine
- la spécification abstraite des interfaces
- la spécification concrète des interfaces

Le **modèle de tâches** est la décomposition des tâches, que l'utilisateur veut pouvoir réaliser, en sous-tâches. Les tâches sont décomposées jusqu'à ce que toutes les tâches obtenues soient des opérations que l'application peut réaliser (voir 2.2.3.1). Le **modèle de domaine** est l'ensemble des données et des opérations que l'interface peut réaliser.

Le niveau de **spécification abstraite** des interfaces est composé :

- des objets d'interaction abstraits c'est-à-dire les tâches de plus bas niveau comme la sélection d'un élément d'un ensemble
- des éléments d'information, c'est-à-dire les données présentées à l'utilisateur, comme le texte d'un label
- des unités de présentation c'est-à-dire l'abstraction des fenêtres.

La spécification abstraite est l'abstraction des informations qui seront présentées à l'utilisateur, c'est à ce niveau que le dialogue est contenu dans les MBA.

Le niveau de **spécification concrète** des interfaces représente l'interface en termes de primitives des éléments des boîtes à outils (fenêtres, boutons, menus…) et des agents de placement des éléments de la fenêtre.

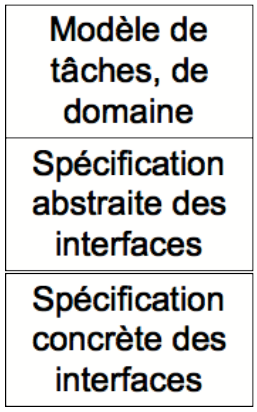


Figure 2.3.1 : Le modèle des approches basées sur modèles (structuration du cadre de référence CAMELEON (voir 2.3.3)

D'autres éléments interviennent dans les environnements de développements basés sur modèles :

- les outils de modélisation : ils aident les développeurs à construire le modèles
- les critiques de graphisme et les conseillers : ce sont des outils d'évaluation du graphisme et de propositions d'amélioration
- les outils de graphisme automatique : ils réalisent une partie du modèle que l'utilisateur n'aura pas à spécifier
- les outils d'implémentation : ils transforment la spécification concrète de l'interface en une représentation qui peut être directement utilisée par une boîte à outils ou un constructeur d'interface.

Ces outils permettent la génération ou la semi-génération d'une partie du modèle. Ils sont organisés avec le modèle comme sur la Figure 2.3.2. Cependant, tous les environnements de développement des interfaces basés sur modèles n'offrent pas tous ces outils.

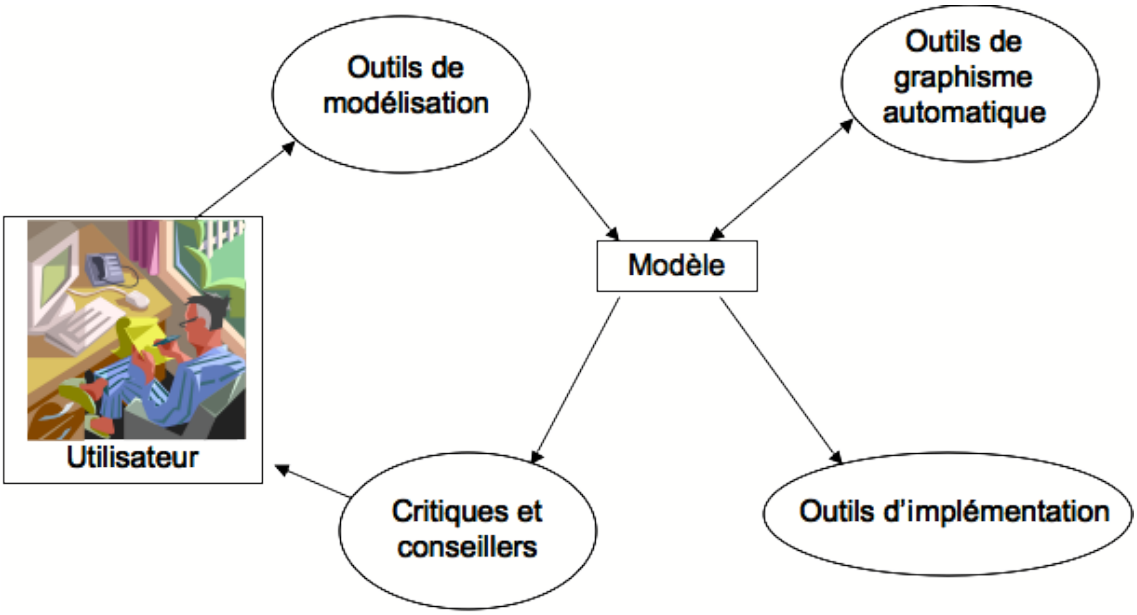


Figure 2.3.2 : Organisation des différents outils et du modèle

Les approches basées sur modèles nécessitent la conception des modèles a priori de la génération. De ce fait, chaque modification d'une partie du modèle rend nécessaire la reconduction complète du processus de génération. Or, comme nous l'avons indiqué précédemment, les modifications (ou complétion) de spécifications sont courantes tout au long du processus de conception ce qui le rend coûteux. Cependant, ils permettent également de déléguer au système les étapes automatiques de transformation d'un modèle à l'autre. C'est pourquoi, des approches utilisant les modèles souhaitent proposer une alternative de conception en utilisant les modèles comme support d'aide au concepteur et non plus comme substitution à celui-ci en appliquant des transformations automatiques ciblées (comme pour s'adapter au contexte d'utilisation par exemple).

Nous présentons ci-dessous quelques-uns des outils basés sur modèles et des approches IDM pour l'IHM.

2.3.1.2. MOBI-D

MOBI-D [Puerta 1997] (MOdel-Based Interface Designer) est la suite du projet Mecano [Puerta 1996], qui permet de générer automatiquement des interfaces à partir des modèles de domaine. MOBI-D représente une nouvelle génération des environnements de développement basés sur modèles en y apportant trois innovations importantes :

- l'utilisation d'un seul et unique langage textuel pour définir le contenant, la structure et les relations entre les différents éléments de l'interface,
- une définition formelle du graphisme de l'interface. Celui-ci devient un ensemble de relations entre les objets de l'interface. Le graphisme de l'interface devient donc un ensemble de relations entre les différents objets.
- la production du graphisme de l'interface en fonction des choix faits par l'utilisateur et des développeurs.

MOBI-D propose une conception basée sur l'utilisateur. L'interface est alors produite en fonction de la description des tâches que l'utilisateur veut qu'elle accomplisse.

La Figure 2.3.3 (de [Puerta 1997]) présente le déroulement de la conception d'une interface développée à l'aide de cet outil.

L'utilisateur final fait une description textuelle des tâches qu'il souhaite pouvoir réaliser grâce à un outil interactif de MOBI-D. Cet outil aide l'utilisateur à définir les tâches à l'aide de mot clés. Puis, à partir de cette description, le développeur construit parallèlement les modèles de domaines et les modèles de tâches utilisateur et les intègre.

À partir de cette intégration, MOBI-D recommande une représentation et des interactions afin de guider le développeur lorsque celui-ci réalise le graphisme et l'implémentation de l'interface, et vérifie que toutes les données et toutes les tâches sont prises en compte par les interfaces.

Lorsque toutes ces étapes ont été réalisées, l'utilisateur teste l'interface produite.

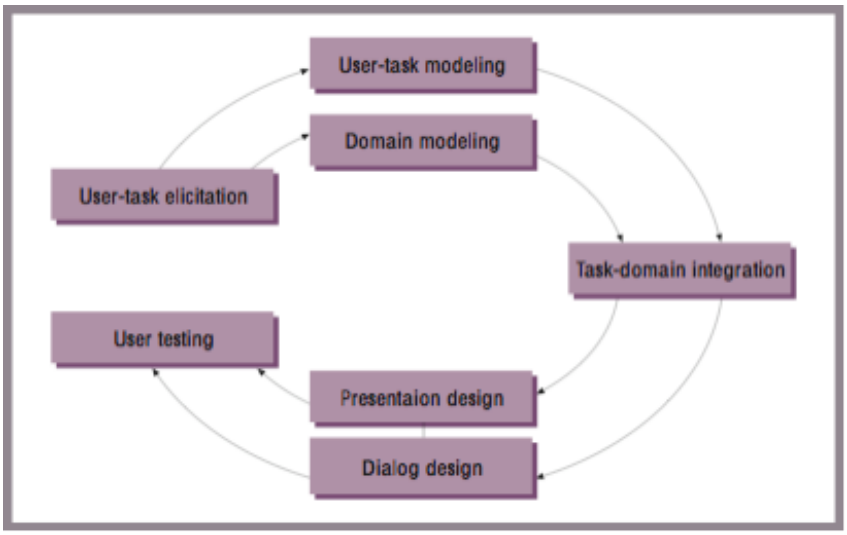


Figure 2.3.3 : Le cycle de développement, centré utilisateur, d'une application interactive issu de [Puerta 1997]

2.3.1.3. Mastermind

Mastermind (Models Allowing Shared Tools and Explicit Representation to Make Interfaces Natural to Develop) [Szekely, et al. 1995] est une MBA née de la fusion de deux projets : Humanoïde [Szekely, et al. 1993] et UIDE. Chacun d'entre eux avait ses points forts :

- pour Humanoïde : le modèle de présentation, les outils de modélisation et la performance
- pour UIDE : le modèle de dialogue, les critiques graphiques et les outils de génération d'aide

Mastermind possède donc toutes ces caractéristiques.

Cette approche génère automatiquement des interfaces à partir de tâches spécifiées par les utilisateurs finaux et de trois autres informations :

- le contenant
- la structure et le placement des écrans
- le rôle de chaque élément présent à l'écran dans l'exécution des tâches [Szekely, et al. 1995].

Le modèle de Mastermind est divisé en trois parties : le modèle d'application, le modèle de tâches et le modèle de présentation. Ces trois modèles sont décrits au moyen du modèle objet de CORBA (Common Object Request Broker Architecture).

Le modèle d'application définit les capacités de l'interface. Ceci lui permet d'avoir comme point fort, l'obtention d'interfaces d'applications distribuées. CORBA permet la définition de classes (et l'héritage) contenant des attributs et des méthodes ainsi que le traitement des exceptions. À ces caractéristiques, Mastermind a ajouté :

- la notion de pré-condition à l'appel aux méthodes
- la notion de rapport : n'importe quel objet peut enregistrer et mettre à jour les rapports, être informé qu'un changement a été effectué sur un autre objet et changer d'état si nécessaire

Le modèle de tâches est modélisé à l'aide d'un des outils de Mastermind, l'outil *Application Modeling Suite*. Dans l'arbre des tâches représentant hiérarchiquement les tâches, les tâches sont de trois types différents : utilisateur, présentation ou application. Le dialogue est défini par ce modèle de tâches. En effet, les tâches utilisateur de plus bas niveau spécifient les techniques d'interaction alors que les tâches présentation représentent des modifications d'un des attributs de l'interface.

Le modèle de présentation, quant à lui, est l'apparence visuelle de l'interface.

2.3.2. L'approche générative

Pour permettre la conception d'applications interactives capables de s'adapter à plusieurs plateformes d'accueil, de nombreux travaux proposent l'utilisation de modèles communs pouvant être spécifiés lors d'une étape de génération. Parmi ces travaux, un grand nombre se basent sur le langage de méta-modélisation XML [Souchon et Vanderdonckt 2003].

Les modèles utilisés peuvent être de nature différente (et combinés par la suite) : modèles de tâches, de domaine, de présentation et spécifient l'interface finale en suivant un procédé à base de *transformation* (au sens IDM du terme) des modèles jusqu'à l'obtention de l'interface. Nous présentons dans les sections 2.3.2.1 et 2.3.2.2, deux outils de génération d'interfaces. Le point de départ des transformations de modèles est dans ces deux approches un modèle de tâches.

Limbourg et al. dans [Limbourg, et al. 2001a] proposent une utilisation des méta-modèles pour générer des interfaces issues de la spécification sous forme d'un ensemble de modèles de tâches. Cette approche est composée d'une étape en amont de la génération dont le but est de disposer d'une ensemble de modèles de tâches décrit en XML quel que soit l'ensemble de modèles de tâches initial. Nous présenterons cette approche dans la section 2.3.2.3.

2.3.2.1. TERESA

TERESA [Mori, et al. 2003, Mori, et al. 2004] (Transformation Environment for inteRactivE Systems representAtions) a pour but la génération ou la semi-génération des interfaces indépendamment de la plateforme. Pour cela, l'outil utilise la notation CTT. D'après Paternò : « *L'idée basique est de capturer toutes les exigences au niveau du modèle de tâches et d'être alors capable d'utiliser quelques informations pour générer les UI adaptées pour chaque type de plateforme considérée.* ». En effet, certaines propriétés assurent l'utilisabilité d'une application interactive sur une plate-forme donnée. Ces propriétés sont, par exemple, la taille de l'écran, l'absence de multi-fenêtrage, le système d'exploitation, les boîtes à outils.

Le développement d'interface, selon l'approche de TERESA consiste en plusieurs étapes :

- la réalisation d'un modèle de tâches de haut-niveau décrivant une application multi-contexte.
- le développement d'un modèle de tâches pour chaque plate-forme considérée.
- le passage du modèle de tâches à une interface abstraite.
- la génération de l'interface.

Le modèle de tâches de haut niveau décrivant une application multi-contexte.

À cette étape, le concepteur ne produit qu'un seul modèle de tâches. Dans celui-ci, seuls les besoins concernant les activités logiques et les relations les liant entre elles sont spécifiées.

Le modèle de tâches pour chaque plate-forme considérée.

Le concepteur filtre le modèle de tâches réalisé à l'étape précédente en précisant pour chaque tâche la (ou les) plate-forme(s) sur laquelle (lesquelles) elle devra s'accomplir. À la fin de la réalisation de cette étape, un modèle de tâches est défini pour chaque plate-forme.

Le passage du modèle de tâches à une interface abstraite.

À partir des modèles de tâche obtenus, un ensemble de présentations abstraites qui compose une description abstraite de l'interface utilisateur. Ces présentations abstraites sont obtenues automatiquement du modèle de tâches à l'aide des ensembles des tâches activables (ETS) et les différents opérateurs. Une fois ces présentations réalisées, une analyse des opérateurs temporels liant les tâches décrits dans le modèle de tâches permet d'obtenir les transitions permettant de passer de l'une à l'autre.

La génération de l'interface.

La dernière étape consiste à générer une interface adaptée à la plate-forme sur laquelle elle sera utilisée en utilisant ses propriétés spécifiques.

TERESA propose de générer la présentation et le dialogue d'une application à partir d'un modèle de tâches sous forme d'un diagramme CTT. Cette transformation est réalisée en plusieurs étapes :

- Calculer les tâches activables à un moment donné, les ETS (Enabled Task Sets), en analysant les relations temporelles entre les tâches. Pour cela, l'algorithme utilisé est le même que celui utilisé pour la simulation dans l'outil CTT). L'heuristique de base de TERESA est que les éléments techniques qui permettent la réalisation des tâches de chaque ETS sont naturellement candidats pour appartenir à la même présentation.
- Une fois les ETS calculés le concepteur peut choisir, s'il le juge pertinent, d'appliquer une des quatre heuristiques supplémentaires qui permettent de regrouper des ETS différentes dans la même présentation. Les ensembles obtenus sont nommés des PSs (Presentation Sets).
- Chaque interface contient une ou plusieurs présentations, chacune d'entre elles a :
    - une structure (les différents éléments qui la composent), c'est ce qui définit la partie présentation graphique
    - une ou plusieurs connexions, c'est-à-dire un ou plusieurs ensembles composé d'un des éléments de la structure et d'une présentation. Un élément est l'objet d'interaction qui permet de « passer » à la présentation associée. Ce sont ces connexions qui permettent de déterminer le dialogue. Le couple (elt, pres) signifie donc que l'élément « elt » permet de passer à la présentation « pres ».

Le dialogue est défini de la manière suivante, pour chaque ensemble de présentation P, on définit, parmi les tâches qu'elles peuvent accomplir celles qui sont des transitions, nommées *transitiontasks*, ce sont donc les tâches dont l'exécution (dans l'abstract user interface) permet le passage à un autre ensemble de présentation P'.

Bien que ce ne soit pas dit, du fait de la manière dont les *Presentation sets* sont obtenus, le dialogue à l'intérieur de chacune d'entre elles n'est pas décrit. En effet, chaque *Presentation set* est une boîte de dialogue dans laquelle toutes les tâches sont réalisables en parallèle. Le modèle de tâches initial n'est donc plus pris en compte. De ce fait, des informations sont perdues.

2.3.2.2. Dygimes framework

Le Dygimes (DYnamically Generating user Interfaces for Mobile computing devices and Embedded Systems) framework [Luyten 2004, Luyten, et al. 2003], a pour but, tout comme TERESA de réaliser pour une même application une présentation pour chaque plate-forme sur laquelle elle tournera.

Comme il est dit dans [Luyten 2004] : *The main differences between the TERESA tool and the Dygimes framework, are that the latter support runtime creation of UIs and the possibility to use different widget libraries in addition to mark-up*

languages. De plus, Dygimes framework met l'accent sur la génération automatique des interfaces alors que TERESA réalise une production semi-générée (en partenariat avec l'utilisateur). Enfin, Dygimes framework offre un environnement de développement moins évolué que TERESA

Afin d'obtenir les différentes présentations générées et le dialogue d'une application, Dygimes framework repose sur le diagramme CTT de cette application. À partir de ce diagramme, des widgets sont associés à chaque tâche dont la réalisation en nécessite un, par exemple, un bouton « Quit » qui est associé à un tâche *Quit*.

La Figure 2.3.4 présente l'association des widgets pour une application qui permet l'envoi d'email. Elle est tirée, ainsi que les Figures 2.3.5 et 2.3.6, de [Luyten 2004] dans laquelle l'ensemble de l'exemple est traité.

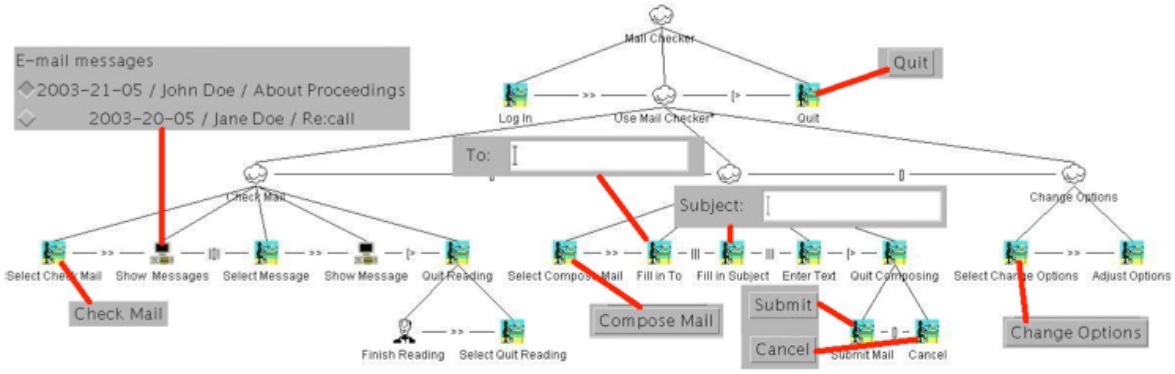


Figure 2.3.4 : Exemple d'un diagramme CTT annoté avec les widgets associés aux tâches

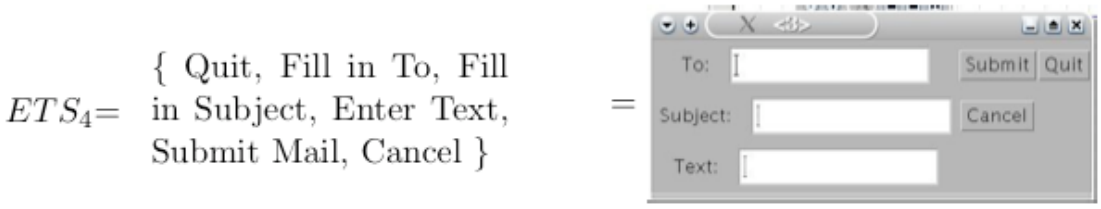


Figure 2.3.5 : Exemple de la représentation graphique d'un état

Dans la machine à états définissant le dialogue de l'application, chaque ETS est un état, les transitions étant les événements associés à chaque widget. On obtient alors la machine à état d'une application qui permet l'envoi de mails présentée sur la Figure 2.3.6.

Cette approche a choisi de ne prendre en compte que le dialogue inter-fenêtre, elle ne se préoccupe pas de ce qui peut se produire dans une fenêtre (activation d'un bouton, par exemple).

Dans cette approche ainsi que dans celle de TERESA, le modèle de dialogue est réalisé en calculant les ETS afin de former les états, puis les transitions sont déterminées parmi les tâches de chaque ETS. Cependant, dans Dygimes, la transformation du modèle de tâches décrit dans le formalisme CTT à la description du dialogue sous forme d'automate est détaillée alors que l'outil TERESA ne met à notre disposition que des fichier XML générés.

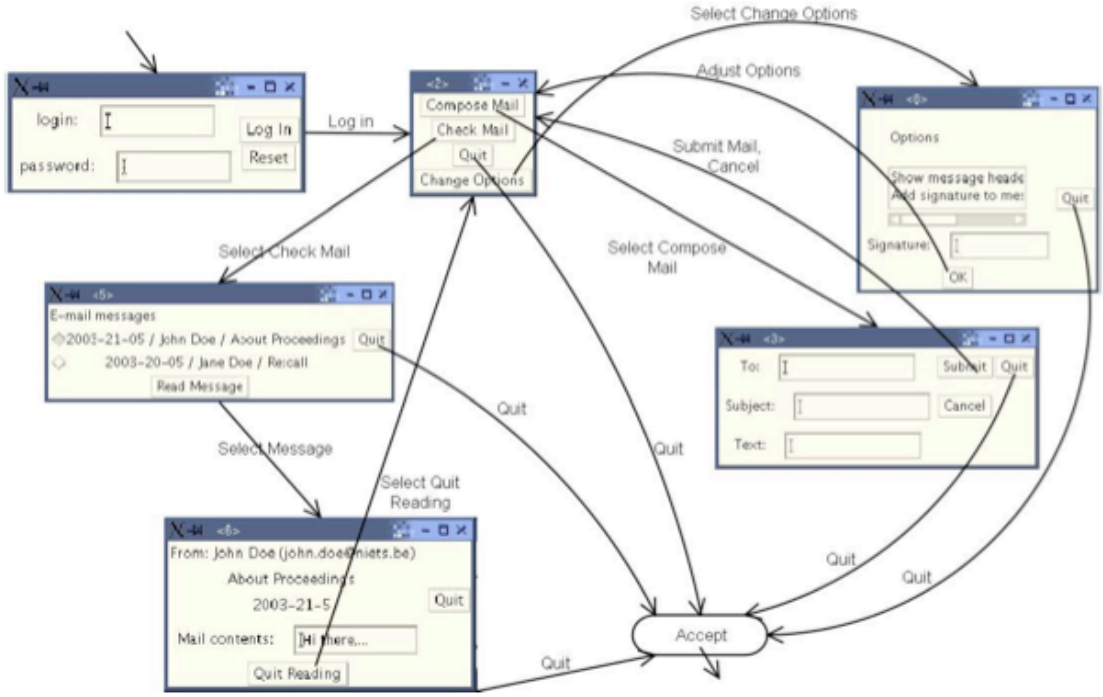


Figure 2.3.6 : L'automate de l'application exemple

2.3.2.3. DOLPHIN

DOLPHIN (Domain Ontologies Leading to the Production of Human-computer INterfaces) [Limbourg et Vanderdonckt 2003] a pour but la production des modèles de dialogue et de présentation à partir de divers modèles de tâches.

L'obtention des modèles de dialogue et de présentation à partir de modèles de tâches différents spécifiés dans des fichiers se fait à travers plusieurs étapes :

- *Transformation d'un fichier en un fichier syntaxiquement équivalent à XML :* si le fichier de spécification de tâches n'est pas en format XML (eXtensible Markup Language) compatible (c'est-à-dire avec une structure compatible), alors il est transformé, en un fichier syntaxiquement équivalent

à XML. Cette transformation est réalisée à l'aide du méta-modèle correspondant (défini dans un fichier DTD (Document Type Definition)). Dans le cas où les spécifications des modèles de tâche seraient déjà dans un fichier XML compatible, alors cette étape n'est pas réalisée.

- *Uniformisation :* une fois que les modèles de tâche sont spécifiés dans un fichier XML ou équivalent, ils sont uniformisés afin de n'obtenir qu'un seul modèle de tâches. L'uniformisation est réalisée à l'aide de règles d'uniformisation décrites dans un fichier XSLT (eXtensible Stylesheet Language Transformations). Le fichier obtenu est en format XIML (eXtensible user Interface Markup Language) [Eisenstein, et al. 2001].

Production des modèles de dialogue et de présentation : en appliquant des règles de design et des recommandations au modèle de tâches du fichier XIML de l'étape précédente, les modèles de dialogue et de présentation sont obtenus. Cette étape est davantage détaillée dans [Limbourg, et al. 2001b].

La Figure 2.3.7 (traduite à partir d'un des schémas de [Limbourg, et al. 2001a]) présente l'architecture du logiciel DOLPHIN.

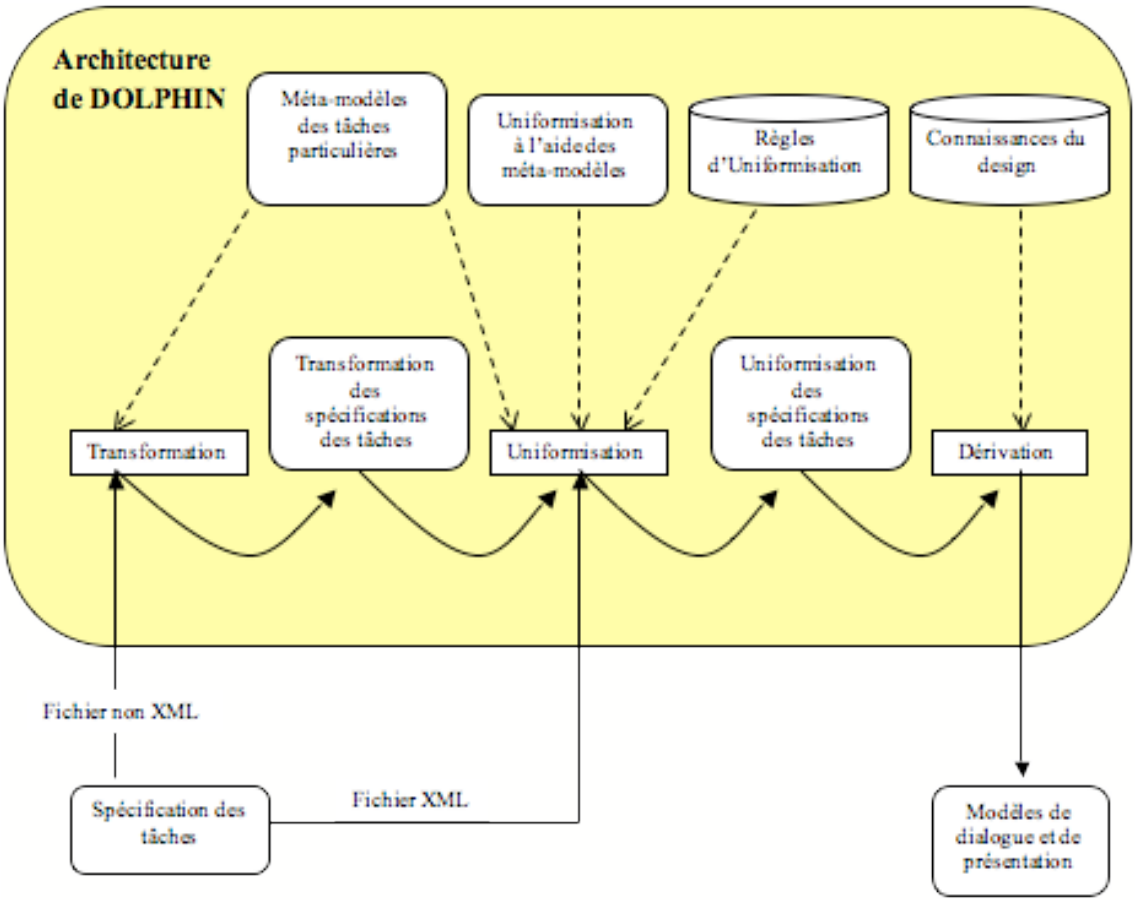


Figure 2.3.7 : Architecture de DOLPHIN

Cette approche permet la réutilisation de modèles de tâches et des outils qui leur sont associés ainsi que l'utilisation de futurs modèles de tâches via un ajout de faible coût : une méta-modélisation de ces futurs modèles.

2.3.3. Le cadre conceptuel CAMELEON

Le projet CAMELEON (Context Aware Modelling for Enabling and Leveraging Effective interactiON) a proposé un environnement de conception pour les applications plastiques. Les applications plastiques sont des applications capables de s'adapter au contexte d'utilisation. Pour permettre cette adaptation, cette approche se repose sur les modèles (Figure 2.3.8 issu de [Calvary, et al. 2003]).

Les premiers modèles (les *modèles initiaux*) sont produits par les concepteurs (modèles à gauche sur la Figure 2.3.8). Ces modèles sont ensuite précisés pour chacun des contextes. À partir de chacun de ces modèles spécifiques, différentes étapes de réification permettent de produire l'interface abstraite, l'interface concrète puis les interfaces finales adaptées aux différents contextes. Sur chacune de ces étapes, le concepteur peut intervenir et réaliser des modifications en revenant éventuellement à l'une des étapes précédentes.

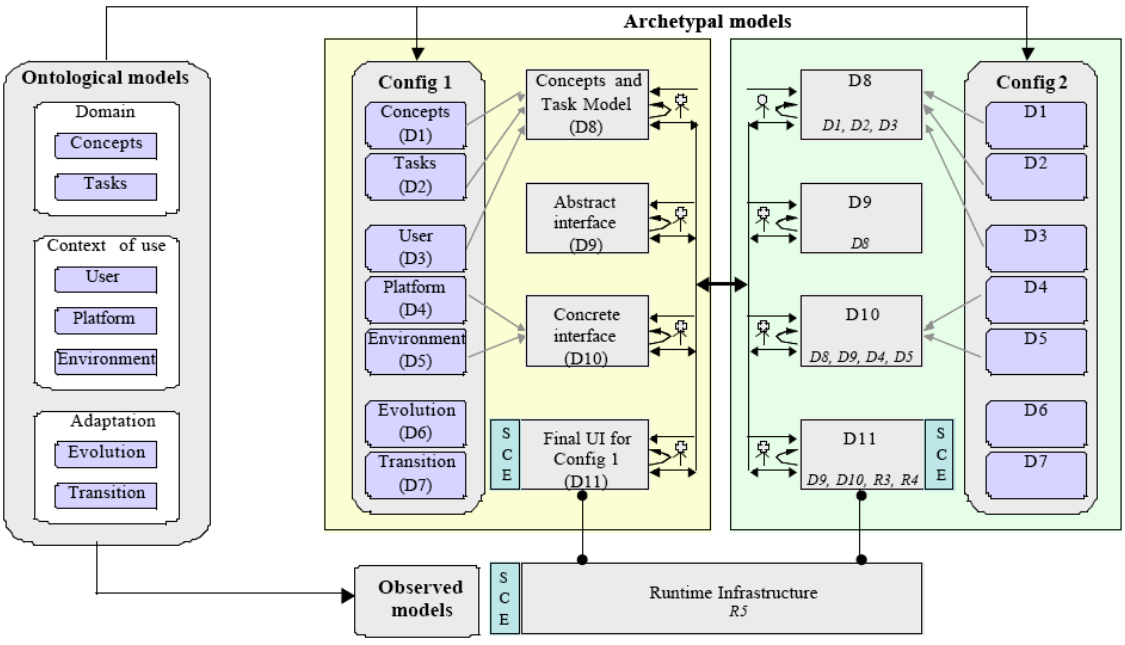


Figure 2.3.8 : Structure du projet CAMELEON issu de [Calvary, et al. 2003].

À partir de ce cadre de conception défini par le projet CAMELEON (Figure 2.3.8), le projet MORFEO[6] propose un ensemble de modèles de référence pour

[6] http://forge.morfeo-project.org

les applications interactives. Ces modèles sont méta-modélisés en UML et sont à la fois les modèles utilisés uniquement pour la partie interface (modèle de tâches, modèle d'interface abstraite, modèle d'interface concrète…) ainsi que ceux utilisés dans le reste de l'application interactive (modèle de domaine et modèle de contexte).

2.4. Conclusion sur l'état de l'art

Depuis de nombreuses années, on a cherché à définir des modèles et des méthodes pour construire des systèmes informatiques plus efficacement. Les premières approches, orientées système, ont conduit à des modèles de développement largement admis aujourd'hui (Cycle en cascade puis en V), et des méthodes diverses (SADT, Merise, OO, ...). Toutes ces méthodes promeuvent des modèles divers et variés : diagrammes à flots de données, diagrammes à états, modèles conceptuels, diagrammes d'interaction, diagrammes de séquences, méthodes formelles basées sur modèles, etc. Tous ces méthodes et modèles concourent à produire des systèmes plus fiables, plus facilement extensibles, et plus facilement maintenables. La démarche initiée par l'OMG autour de la « Model Driven Architecture » et reprise plus généralement dans la « Model-Driven Engineering », conduit vers une démarche automatisée de la production logicielle, à base de transformation et de coopération entre modèles.

Malheureusement, ce n'est pas si simple : l'introduction de l'utilisateur dans la boucle a profondément modifié la donne... L'apparition des « utilisateurs » a imposé de considérer un nouveau point de vue, celui de l'humain. Les spécifications du système, pierre angulaire de la conception logicielle, sont en fait le point faible, puisque très difficile à établir et valider. Alors que l'on est habitué de tous temps à parler de spécifications fonctionnelles, l'introduction de l'utilisateur nécessite de prendre en compte de nombreuses spécifications non-fonctionnelles, découlant de la nature humaine de l'intervenant (capacités cognitives limitées, distances de Norman, comportements et apprentissage, etc.). Tout cela demande de nouveaux modèles et de nouvelles méthodes. Les différences de point de vue entre les utilisateurs et le système sont représentées par les spécifications (fonctionnelles et humaines). Ces spécifications peuvent donc être vues comme la ligne frontière entre le monde de l'humain et le monde du système.

Les recherches en IHM ont produit à ce jour de nombreux résultats, conduisant à la notion générale de conception centrée-utilisateur. L'impossibilité de formaliser strictement les facteurs humains, associée à la nécessité de fréquents aller-retour entre concepteurs et utilisateurs pour s'assurer de la validité des besoins recueillis, ont conduit à modifier le processus de conception. Ces modifications sont illustrées par la proposition de modifications substantielles des modèles de développement, mettant en avant la nécessité d'une conception itérative avec une forte dose de prototypage (utilisé pour donner une représentation du système en conception à l'utilisateur et pouvoir recueillir ses remarques), et le développement conjoint des méthodes et outils pour la validation des systèmes. La prise en compte de l'activité de l'utilisateur dans l'activité de conception, et non plus seulement de ses besoins fonctionnels, ainsi que la nécessité de valider le système construit, et non pas seulement sa conception, ont conduit au développement de nouveaux modèles comme les modèles de tâches. La nécessité de mise en place de cycles itératifs, avec de forts besoins d'adaptabilité du code produit, a

entraîné le développement d'architectures logicielles adaptées. La complexité des dialogues homme-machine nécessaires a engendré de nombreux travaux sur leur formalisation et leur vérification. Les concepts de base utilisés sont les mêmes que ceux du génie logiciel (systèmes de transition, concepts d'abstraction/spécialisation, etc.), mais force est de constater que les modèles sont encore très éloignés les uns des autres. Ils coopèrent mal, et sont difficilement utilisables conjointement.

L'IDM, appliquée à l'IHM, est peut-être la solution à tout cela. Il faut d'ailleurs remarquer que l'IHM a fait de l'IDM dès les années 80 : que sont les Model Based Systems, sinon des systèmes conçus à l'aide de modèle... datant d'une époque antérieure à la MDE ! Après avoir étudié les apports de l'IDM en IHM, on peut voir que ceux-ci sont encore assez modestes. En particulier, la méthode la plus utilisée est la génération. La collaboration entre modèles, même exclusivement issus de l'IHM (tâches, dialogue, architecture), est aujourd'hui limitée à sa plus simple expression.

L'objectif de cette thèse est d'apporter une contribution qui s'inscrit dans le cadre de l'IDM. Il s'agit de définir les modalités de collaboration entre modèles situés de part et d'autre de la ligne frontière mentionnée plus haut, et plus particulièrement entre modèles de tâches et modèles de dialogue (le tout étant indissociable du modèle d'architecture, côté système). Pour cela, nous allons, dans le chapitre suivant, étudier plus précisément ces modèles, et choisir ceux que nous utiliserons dans la suite de ce travail.

Chapitre 3. Étude préliminaire et positionnement du problème

Dans le cadre d'une approche dirigée par les modèles, et en nous basant sur le schéma proposé par le projet CAMELEON, nous souhaitons nous focaliser sur l'articulation entre l'étape conceptuelle et l'étape abstraite, et plus particulièrement entre les modèles de tâches et les modèles de dialogue.

Comme nous l'avons vu au chapitre précédent, les outils développés autour de ces modèles, qu'il s'agisse de simulateurs comme dans CTTE ou K-MADe, ou de générateurs comme dans TERESA, démontrent le lien naturel entre la dynamique exprimée dans les modèles de tâches et celle implémentée dans les applications, à travers leur modèle de dialogue.

Cependant, les principaux travaux exploitant ce lien utilisent une approche génératrice. Cette dernière est-elle la seule possible ? Est-elle la plus appropriée ? Depuis peu, de nouveaux outils ont fait leur apparition, et ce dans les deux domaines. Peuvent-ils permettre d'étendre les travaux précédents en considérant de nouvelles pistes ?

C'est l'objet de ce chapitre que de répondre à ces questions. Pour cela, nous avons procédé à plusieurs études, théoriques ou empiriques. À propos des modèles de tâches, nous avons étudié le pouvoir d'expression des modèles et de leurs outils associés, ce qui nous a conduit à choisir le modèle qui nous semble le plus complet à ce jour. Pour confirmer l'intérêt de ce modèle, nous avons effectué une étude empirique de son usage, et plus particulièrement des points qui le rendent original.

En ce qui concerne les modèles de dialogue, beaucoup de solutions proposées dans la littérature se contentent de décrire une vue conceptuelle, indépendante de l'implémentation. Nous avons effectué une étude comparative des modèles utilisables directement en phase de développement, soit à l'aide d'un animateur, soit par l'intermédiaire d'une boîte à outils spécialisée. Cette étude nous permettra de justifier notre choix pour le développement de notre méthode.

Enfin, dans une troisième partie, nous relatons les résultats d'une étude précise des limites de l'approche générative du dialogue à partir des modèles de tâches, qui nous permettent de définir la marche à suivre pour tester une autre approche, toujours dans le cadre de l'ingénierie dirigée par les modèles.

3.1. Choix du modèle de tâches

Dans la section 2.2.3.1, nous avons décrit le rôle des modèles de tâches dans la conception centrée-utilisateur. Nous avons également indiqué qu'un grand nombre de notations avaient été utilisées pour permettre l'expression de ces modèles. Ces notations ont été développées dans des buts différents et font intervenir des concepts variés. Pour

la suite de nos travaux, nous avons besoin de choisir une de ces notations de modélisation des tâches.

Dans ce but, nous proposons de débuter par une étude comparative de différentes notations. Étant admis que l'utilisation de modèles est facilitée par un environnement qui le supporte [Limbourg et Vanderdonckt 2003], et considérant que notre objectif consiste à aller vers de véritables outils manipulant les modèles, nous ne comparerons ici que les modèles de tâches possédant un outil associé. Parmi les outils proposés pour exprimer les modèles de tâches, certains reposent sur une description textuelle du modèle. Nos travaux s'inscrivent dans une démarche de conception centrée utilisateur dans laquelle le point de vue de l'utilisateur est représenté par l'expression de modèles de tâches. Dans ce contexte d'étude, nous considérons que l'expression de l'activité à l'aide d'une interface graphique est plus adaptée. De plus, nous avons également choisi d'affiner notre sélection en ne prenant en compte que les modèles de tâches intégrant la description d'objets dans le modèle (étant entendu que les objets font partie des éléments indispensables à la modélisation de l'activité [Dix 2008]).

Nous avons ainsi identifié six modèles de tâches différents qui respectent ces différents critères de sélection. MAD* [Gamboa et Scapin 1997, Scapin et Bastien 2001] (IMAD), CTT [Paternò, et al. 1997] (CTTE), Diane+ [Tarby et Barthet 2001] (TAMOT), GTA [van der Veer 1996] (EUTERPE), K-MAD (K-MADe) et un environnement d'analyse de tâches (AMBOSS) [AMBOSS].

Ces différents modèles sont présentés dans la section 3.1.1. La comparaison des modèles présentée dans la section 3.1.2, aboutit à la sélection d'une notation de modèle de tâches. Une évaluation de l'utilisation de ce modèle est alors présentée dans la section 3.1.3.

3.1.1. Présentation des modèles étudiés

Nous présentons dans cette section, les six notations de modèle de tâches ayant été implémentées dans des outils : MAD*, CTT, Diane+, GTA, K-MAD et AMBOSS. La description de ces modèles correspond aux modèles vus à travers l'utilisation des outils qui leur sont associés.

3.1.1.1. La notation MAD* (outil : IMAD)

La Méthode Analytique de Description (MAD) [Scapin et Pierret-Golbreich 1989] est issue de travaux à l'intersection des domaines de l'ergonomie, de l'informatique et de l'intelligence artificielle. Les travaux initiaux avaient plusieurs objectifs [Gamboa 1998] : considérer les moyens par lesquels les utilisateurs représentent les tâches ainsi que la logique de data-processing ; prendre en compte les aspects conceptuels et sémantiques et non seulement les aspects syntaxiques et

lexicaux ; obtenir une structuration des tâches de manière uniforme ; réaliser des descriptions à partir de points de vue déclaratifs (états du monde) ou procéduraux (moyens d'atteindre ces états) ; permettre le parallélisme et pas seulement la représentation séquentielle (synchronisation des tâches) ; et être évaluables.

MAD est un modèle de tâches hiérarchique composé d'un ensemble de décompositions temporellement organisées. La décomposition temporelle exprime l'organisation temporelle des tâches d'un même niveau hiérarchique. MAD propose ainsi des liens temporels tels que *SEQ*, qui organise les tâches de manière séquentielle ou *ALT,* qui indique qu'une tâche est réalisée parmi plusieurs. Afin de décrire de manière précise chaque tâche, des attributs, des pré- et des post-conditions peuvent y être associés. Une extension de ce modèle, MAD* [Gamboa et Scapin 1997] a été créée. Celle-ci enrichit le modèle MAD par l'ajout d'opérateurs temporels (l'interruption, la désactivation), de tâches multiutilisateurs, et d'une sémantique précise pour les objets manipulés. C'est ce dernier modèle qui a servi de base à l'implémentation de l'outil IMAD. Dans cet outil, deux concepts seulement sont manipulés : les tâches et les objets.

Dans IMAD, une **tâche** MAD* est définie par un *numéro* (automatiquement calculé en fonction de sa position dans l'arbre (texte)), un *nom* (texte), un *but* (texte), une *modalité* (manuel, auto, interactive), un caractère *facultatif* (booléen) et un *type* (sensori-motrice, cognitive). Le concepteur peut également y décrire des *observations* générales sur la tâche.

En plus de ces caractéristiques générales sur la tâche, des informations peuvent être précisées. Les informations temporelles sont constituées de la *durée* de la tâche (texte), de son *début* (texte) et de sa *fin* (texte). L'information sur l'*importance* de la tâche (importante, relativement importante, secondaire) est appuyée par la *fréquence* (élevée, moyenne, faible). Dans le modèle MAD*, un autre attribut de la tâche complète ces informations ; il précise les entités (objets) qui sont fréquemment utilisées par les autres tâches. Cet attribut n'est pas présent dans l'outil IMAD. Deux autres catégories d'informations sont associées aux tâches : les informations relatives aux opérateurs de la tâche et les informations relatives à l'interruption des tâches. Les *opérateurs* sont définis par le nom de l'utilisateur (texte), son statut (texte), son rôle (texte), son expérience (texte) et sa compétence (texte). Enfin, les caractéristiques liées à l'interruption dans IMAD sont le caractère d'*interruptibilité* (booléen) et la *priorité* de la tâche (entier). Dans MAD*, une pre-condition qui déclenche l'interruption (texte) est prévue pour préciser les conditions nécessaires au déclenchement de l'interruption, cette dernière n'est pas présente dans IMAD.

Deux types d'**objets** sont présents dans MAD*. Ces deux types correspondent aux concepts orientés-objets de classe (*objets abstraits*) et d'instance (*objets concrets*). Chaque objet est composé d'un *nom* (texte), d'un *numéro* (texte) et d'un ensemble d'attributs. Les attributs des objets abstraits sont caractérisés par un nom (texte) et un type (texte, booléen, entier) et les attributs des objets concrets par un nom (texte) et une valeur. Quelques caractéristiques peuvent être ajoutées à la définition des objets abstraits : une méta-classe (un lien de généralisation), une sous-classe (un lien de spécialisation) et une condition sur le nombre d'instance (restriction à une instance).

Cependant, dans IMAD les deux types d'objets ne sont pas différenciés. Le concepteur ne peut pas donner de valeur aux attributs des objets IMAD.

Enfin, IMAD prévoit l'expression de l'état du monde dans les tâches. Pour cela, l'état initial (avant l'exécution de la tâche) et l'état final (après l'exécution de la tâche) peuvent être édités. Ces deux états sont cependant exprimés sous forme de texte. Il en va de même pour l'expression de conditions à l'exécution de la tâche (pré-condtion d'entrée) et à l'arrêt de l'itération de la tâche (pré-condition d'arrêt). La définition de ces attributs sous forme textuelle empêche leur utilisation pour la simulation du modèle par exemple.

La Figure 3.1.1 représente une description possible avec la notation MAD de l'activité « Envoyer un email » via un mailer.

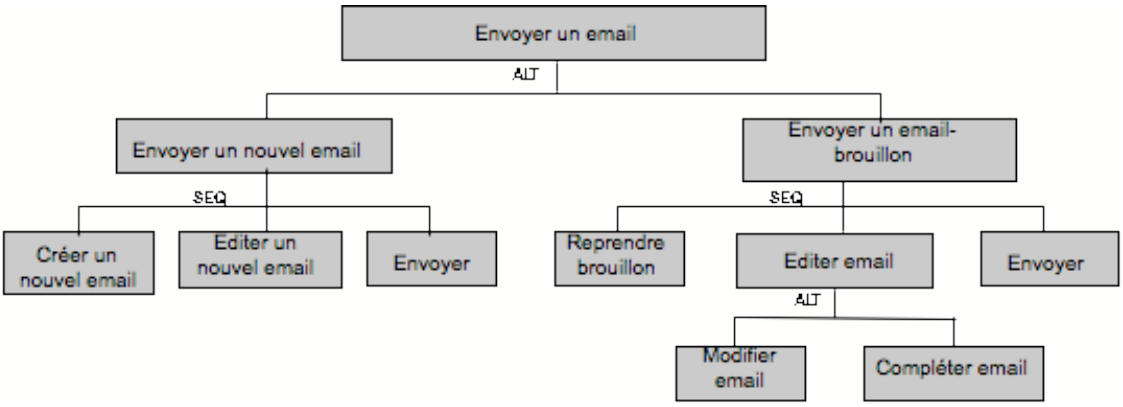


Figure 3.1.1 : *envoyer un email*, un exemple en notation MAD

Un email peut être envoyé en suivant deux procédés distincts, soit l'email est créé à partir de rien (c'est l'*envoi d'un nouvel email*), soit un brouillon de l'email à envoyer a déjà été préparé (c'est l'*envoi d'un email-brouillon*). Du fait que la tâche principale est réalisée en suivant l'un ou l'autre de ces procédés (en accomplissant l'une ou l'autre des deux tâches), *Envoyer un email* décompose les tâches *envoyer un nouvel email* et *Envoyer un email-brouillon* avec l'opérateur de décomposition *ALT*.

Dans le cas de l'envoi d'un nouvel email, la première tâche à accomplir est la création d'un nouvel email, puis, l'édition de ce nouvel email et enfin son envoi.

Si l'email à envoyer est issu d'une première production « brouillon », la tâche consiste à reprendre un des emails sauvegardé dans les brouillons, puis à l'éditer et enfin à l'envoyer. Un email ayant été sauvegardé dans les brouillons a déjà été (partiellement) édité, c'est pourquoi son contenu peut être soit modifié, soit complété.

Cette description se veut proche du point de vue adopté par les concepteurs de MAD* sur la modélisation des tâches, à savoir : permettre la description des tâches de l'utilisateur en suivant la succession de ses objectifs. Il est à noter en particulier que nous avons fait le choix, par exemple, de ne pas regrouper les tâches exécutées de la

même façon (comme *éditer email* et *éditer un nouvel email*) afin de conserver la décomposition en sous-objectifs détaillés par l'utilisateur.

3.1.1.2. La notation CTT (outil : CTTE)

CTT (*ConcurTaskTrees*) est défini par ses auteurs comme étant une notation de spécification des modèles de tâches couvrant les limitations des autres notations utilisées pour la conception des applications interactives [Paternò, et al. 1997]. Son principal but est de fournir une notation facile à utiliser, qui permet de représenter les différentes relations temporelles d'un même niveau d'abstraction et qui peut être utilisée par un novice en modélisation de tâches. CTT est une notation focalisée sur les activités.

Elle est basée sur une structure hiérarchique des tâches représentée par une structure d'arbre. Chaque tâche est graphiquement représentée dans un format de nœud ou de feuilles d'arbre. Les relations temporelles sont exprimées par les opérateurs LOTOS [Paternò et Faconti 1992, Systems 1984] qui sont capables d'exprimer les relations temporelles entre les tâches d'un même niveau d'abstraction.

CTT possède de nombreux opérateurs temporels, qui permettent une organisation hiérarchique des tâches (activation, choix, parallèle, ordre indépendant, interruption, désactivation, activation avec passage d'information, synchronisation). Au contraire du modèle MAD*, pour lequel un seul opérateur est défini pour toutes les tâches filles, les opérateurs de CTT s'appliquent entre deux tâches sœurs.

En plus des opérateurs temporels, CTT permet l'ajout de caractéristiques temporelles aux tâches. Soit T1 une tâche quelconque ;

- L'itération, représentée à l'aide d'un astérisque à côté du nom de la tâche (T1*), signifie que la tâche T1 se répète un nombre indéfini de fois. Elle cesse de se re-exécuter après avoir été désactivée par une autre tâche.
- L'itération finie, notée : T1(N), la tâche T1 est répétée un nombre fixe de fois, N fois.
- L'optionnelle, la tâche T1 peut ne pas être réalisée. Certains opérateurs ne permettent pas que les tâches sur lesquelles ils s'appliquent soient optionnelles. C'est le cas des opérateurs d'interruption, de désactivation et de choix.

Plus d'informations sur cette notation peuvent être trouvées dans [Paternò 2001].

CTTE[7] (ConcurTaskTrees Environment) est l'outil associé à CTT. Il est largement utilisé dans le monde académique. Du point de vue de CTTE, CTT manipule deux concepts : les tâches et les objets.

[7] http://giove.cnuce.cnr.it/ctte.html

Chaque **tâche** est identifiée par un *nom*, un *identifiant*, une *catégorie* et éventuellement un *type* (en fonction de cette catégorie). La catégorie peut être :

- *Utilisateur* () : tâche qui est entièrement réalisée par l'utilisateur. Les types possibles pour les tâches utilisateur sont : comparaison, planification, résolution de problème.
- *Application* () : les tâches de cette catégorie sont complètement réalisées par le système. Les types possibles pour les tâches application sont : feedback, génération d'événement, localisation, vue d'ensemble, visualisation.
- *Interactive* () : tâche réalisée par une action de l'utilisateur sur le système. Les types possibles pour les tâches interactives sont : le contrôle, l'édition, la sélection à choix multiple, la sélection à choix simple, la réponse aux alertes de surveillance.
- *Abstraite* () : tâche qui est constituée de tâches de plusieurs catégories différentes. Aucun type n'est disponible pour les tâches abstraites.

D'autres informations peuvent être ajoutées pour compléter la description de la tâche : une *description* plus détaillée de la tâche (texte), la *fréquence* (basse, moyenne, haute), son caractère *optionnel* (booléen) et *coopératif* (booléen), et la (ou les) *plateforme*(s) sur la(les)quelle(s) cette tâche peut être exécutée. Un attribut de la tâche permet d'informer sur le caractère itératif de celle-ci : *itérative* (booléen) et trois permettent de préciser la durée : le *temps d'exécution minimum* (texte), le *temps d'exécution moyen* (texte) et le *temps d'exécution maximum* (texte).

En plus d'un nom et d'une catégorie, une tâche possède un ensemble **d'objets**. Un objet est une entité utilisée pendant l'exécution de la tâche à laquelle il appartient. Il existe deux types d'objets, les objets *perçus* et les objets *application*.

Les objets *perçus* sont les objets présentant des informations ou des items à l'utilisateur et avec lesquels celui-ci peut interagir (menus, icônes,…).

Les objets *applications* sont quant à eux, des objets appartenant au domaine applicatif. Les objets *perçus* sont utilisés afin de les présenter à l'utilisateur. Un exemple d'objet *application* serait un vol, pour une application de contrôle de trafic aérien.

En plus du type, un objet est caractérisé par un *nom* (texte), une *classe* (texte, objet, description ou position), un *mode d'accès* (lecture seule ou modification), une *cardinalité* (faible, moyen, haut) et les *plateformes* où les objets sont représentés. À notre connaissance, aucune documentation ne décrit en détail les concepts de classe et de cardinalité. À partir de notre utilisation de l'outil CTTE, nous associons le besoin de cardinalité avec la génération d'interfaces [Mori, et al. 2004], basée sur les diagrammes CTT de même que le type et la catégorie des tâches.

Enfin, une pré-condition textuelle peut conditionner l'exécution de la tâche : *pré-condition* (texte).

La définition d'opérateurs hiérarchiques entre les tâches implique la possibilité qu'un même niveau hiérarchique soit temporellement organisé par des opérateurs différents. Ceci a pour conséquence la définition éventuelle d'ambiguïtés sémantiques comme l'indique Paternò dans [Paternò 2001].

La Figure 3.1.2 présente une modélisation possible de l'envoi d'un email via un *mailer* exprimée avec la notation CTT. La tâche racine de l'activité *envoyer un email via un mailer* est composé d'une tâche réalisée itérativement par *Envoyer un email* qui peut être interrompue par une tâche interactive *Quitter* qui met fin à l'activité modélisée. *Envoyer un email* est composée d'une première tâche *Identifier email* qui est interactivement exécutée soit par la création d'un nouvel email (*créer email*), soit par la reprise d'un email dans les brouillons (*Reprendre email*). Une fois l'email identifié, il est édité (*Editer*). L'édition de l'email se fait tant que l'envoi n'a pas été déclenché (*Envoyer*).

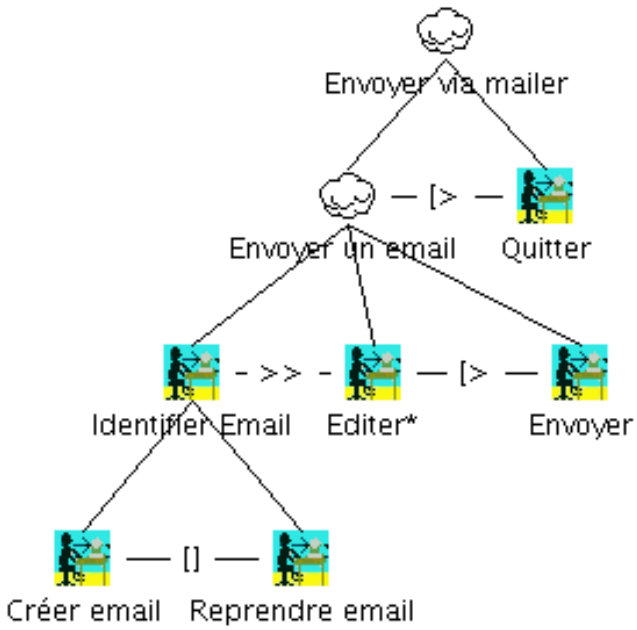


Figure 3.1.2 : *envoyer un email*, un exemple en notation CTT

Cette modélisation est différente de celle proposée sur la Figure 3.1.1 bien qu'elle représente la même activité car nous avons souhaité utiliser CTT pour réaliser une modélisation plus proche de l'activité réalisée au travers de l'utilisation du système. Ainsi, le modèle présenté que la Figure 3.1.2 représente les activités possibles associées à l'envoi du email via l'utilisation des fonctionnalités d'un *mailer* et non pas celles souhaitées pour réaliser l'envoi d'un email (comme c'est le cas sur la Figure 3.1.1).

3.1.1.3. La modélisation des tâches dans Diane+ (outil : TAMOT)

Diane+H [Tarby et Barthet 2001] est une méthodologie complète, qui vise à combler le fossé entre l'ingénierie informatique et les facteurs humains. Tout comme CTT, le modèle Diane+ est focalisé sur la conception des applications interactives. D'une part, Diane+ intègre quelques-unes de caractéristiques du domaine ergonomique et de la psychologie cognitive, et d'autre part, il intervient dans le cycle de vie du développement logiciel.

Diane+ est un modèle de tâches hiérarchique composé par un ensemble d'opérateurs. Cette représentation est différente de celle de MAD* et CTT : Diane+ utilise des fenêtres pour spécifier la décomposition des tâches. Les tâches à l'intérieur des fenêtres sont structurées à l'aide de cinq opérateurs :

- *parallèle* : toutes les tâches sont exécutées en même temps (aucun symbole ne lie les tâches entre elles)
- *séquentiel* : les tâches sont exécutées dans l'ordre (les tâches sont liées par une flèche)
- *XOR* : une seule sous-tâche est exécutée
- *OR* : au moins une tâche est exécutée
- *AND* : toutes les tâches sont exécutées

TAMOT[8] est l'outil associé à Diane+. Un seul concept du modèle Diane+ est manipulé dans cet outil : le concept de tâche.

Une **tâche** est définie par un *nom* (texte), un critère *d'obligation* (booléen), un mode correspondant à *l'exécutant* de la tâche (interactive, automatique, manuelle) et une indication sur le caractère *terminal* de la tâche (est-ce une tâche terminant normalement la tâche mère) (booléen). À ces caractéristiques peuvent être ajoutés un *commentaire* sur la tâche (lien HTML et multimédia) et une description du *feedback* de la tâche (texte). Deux caractéristiques permettent l'expression de l'itération : la *cardinalité minimum* (entier) et la *cardinalité maximum* (entier).

Le modèle de tâches Diane+ intègre des objets, dénommés *data*, et les utilise pour définir les conditions. Dans TAMOT, une seule condition peut être éditée : la pré-condition d'exécution de la tâche. De plus, tout comme pour IMAD et CTTE, cette pré-condition est exprimée sous forme textuelle (ne fait donc pas intervenir les *data*) et donc impossible à évaluer formellement (pour la simulation, la vérification du modèle…).

La Figure 3.1.3 présente l'activité d'envoi d'un nouvel email avec la notation Diane+. Les tâches *Envoyer un email* et *Identifier email* sont détaillées dans les fenêtres qui portent les mêmes noms. Les tâches élémentaires (les tâches qui ne sont pas détaillées) sont représentées dans des rectangles alors que les autres sont dans des ellipses allongées. Enfin, des flèches lient les tâches *Identifier email, Editer* et *Envoyer* puisqu'elles sont réalisées séquentiellement.

[8] http://www.ict.csiro.au/staff/Cecile.Paris/IIT-Track-Record-Past-Projects/Projects/isolde/Tamot/Index.htm

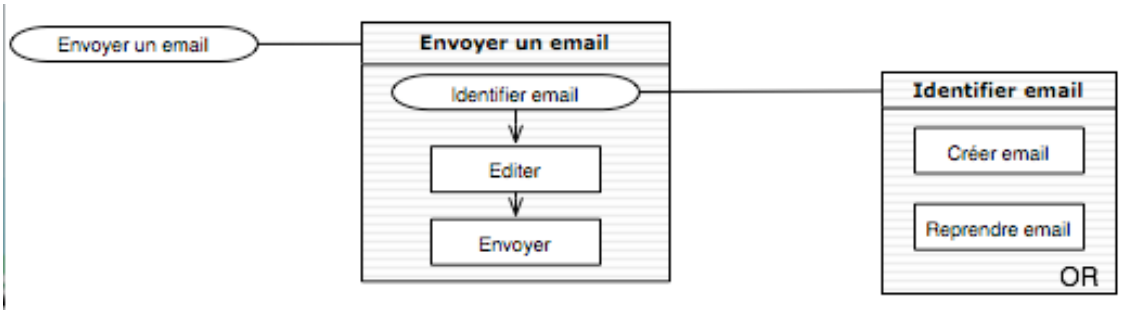


Figure 3.1.3 : *envoyer un email*, un exemple en notation Diane+

3.1.1.4. La notation GTA (outil : EUTERPE)

GTA (Groupware Task Analysis) [van der Veer 1996] est une méthode d'analyse de tâches qui a été développée pour la modélisation d'environnements complexes où différentes personnes interagissent avec les systèmes interactifs.

Deux outils ont été développés pour utiliser ce modèle : l'outil GTA (non supporté) et EUTERPE. Cet outil n'inclut aucune fonction prédéfinie d'organisation temporelle. Du point de vue d'EUTERPE, GTA manipule cinq concepts : les tâches, les rôles, les agents, les objets et les événements.

Une **tâche** dans GTA est définie par son *nom* (texte), son *but* (texte) et son *type* (complexe, unitaire (utilisateur ou système), interaction). À ces caractéristiques peuvent être ajoutées les informations sur la *pertinence* (normale, critique, dangereuse ou optionnelle), s'il s'agit d'une tâche mentale (booléen), moteur (booléen), sa durée (texte) et sa fréquence (texte). Des informations générales sur la tâche peuvent également être apportées sous forme d'un commentaire (texte) ou d'un document média (HTML, lien vers un média). Enfin, à chaque tâche est associé l'ensemble des autres concepts qu'elle manipule (agents, rôles (calculés à partir de la liste des agents), événements, objets).

Dans EUTERPE, les **objets** sont des entités à part entière. Ils sont caractérisés par un *nom* (texte), une *liste d'attributs* (chaque attribut étant composé par un nom (texte) et une valeur (texte)) et une *liste d'utilisateurs* (le nom des agents qui peuvent manipuler l'objet).

Les **agents** sont définis par un *nom* (texte), un *type* (individu, organisation ou système informatique) et un *rôle* (défini comme étant un ensemble de tâches que l'agent a le droit de réaliser).

De plus, EUTERPE permet la définition **d'événements**. Ceux-ci sont composés d'un *nom* (texte) et d'un *ensemble de tâches* liées à cet événement.

EUTERPE propose de définir pour chaque tâche : son état initial (texte), son état final (texte) et ses conditions de départ (conditions nécessaires pour pouvoir débuter

l'exécution de la tâche) (texte) et de fin (conditions nécessaires pour considérer l'exécution de la tâche comme terminée) (texte).

Toutes ces représentations de l'état du monde de la tâche sont exprimées sous forme textuelle et sont, de ce fait, inutilisables par les outils formels.

La Figure 3.1.4 présente un modèle de tâches représentant l'activité d'envoi d'un nouvel email en respectant la notation GTA (via EUTERPE). La décomposition des tâches dans le modèle GTA est spécifiée de gauche à droite. Cependant, tout comme MAD, une seule décomposition des tâches est spécifiée pour un même niveau hiérarchique. Par exemple, le « Seq » indiqué entre la tâche *Send a new email* et *Create email* vaut pour l'ensemble des sous-tâches de *Send a new email*, ce sont donc les tâches *Create email, Complete email* et *Send email* qui sont séquentiellement ordonnées.

Cependant, aucun operateur (appelé *constructeur* dans GTA) n'est défini, c'est le concepteur qui définit lui-même ces opérateurs (texte).

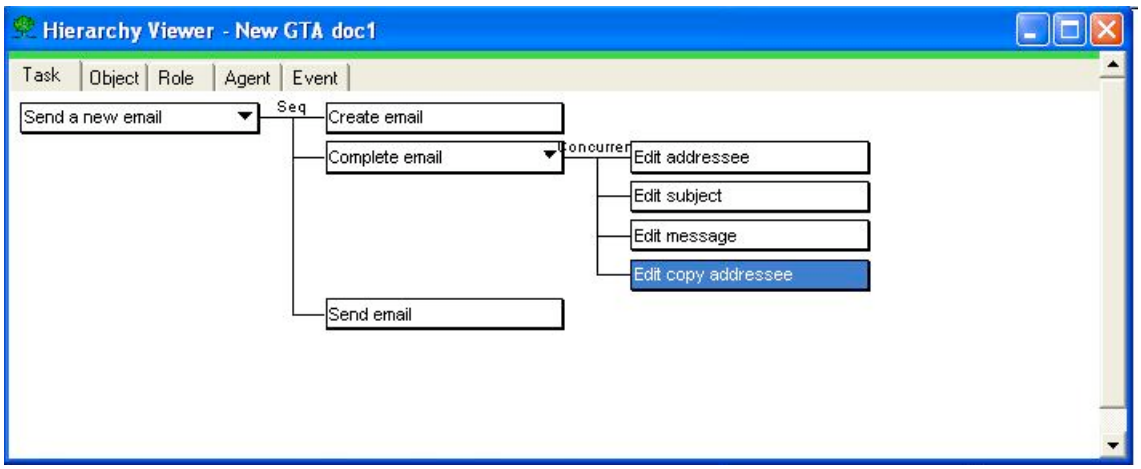


Figure 3.1.4 : *envoyer un nouvel email*, un exemple en notation GTA

3.1.1.5. Le modèle K-MAD (outil : K-MADe)

Le modèle de tâches K-MAD (Kernel of Model for Activity Description) est issu de recherches visant à proposer un noyau pour la modélisation des tâches [Lucquiaud 2005a, Lucquiaud 2005b]. Il permet d'exprimer hiérarchiquement l'activité de l'utilisateur en utilisant une sémantique formelle via les arbres de tâches.

Un outil a été développé pour supporter ce modèle : K-MADe[9]. K-MADe permet l'utilisation de tous les concepts constituant le noyau : les tâches, les objets, les utilisateurs et les événements.

[9] http://www-rocq.inria.fr/merlin/kmade/ http://kmade.sourceforge.net/download.php

Les **tâches** de K-MAD possèdent un ensemble de caractéristiques : *nom* (texte), *numéro* (texte automatiquement défini en fonction de la place de la tâche dans l'arbre), *but* (texte), *exécutant* (utilisateur, système, interactif, abstrait), *fréquence* (élevée, moyenne, faible), *importance* (très importante, importante, peu importante) *modalité* (sensori-motrice, cognitive), caractère *d'interruptibilité* (booléen) et *nécessité* de son exécution (booléen). De plus, d'autres informations peuvent être ajoutées pour compléter la description de la tâche : des *observations* générales sur la tâche (texte), les effets observables (*feedback*) (texte) et la *durée* (texte). Enfin, chaque tâche décomposée précise l'ordonnancement de ses sous-tâches (*opérateurs de décomposition*) (séquentiel, alternatif, parallèle, pas d'ordre).

En plus d'être hiérarchiquement décomposée et organisée à l'aide d'opérateurs de décomposition, chaque tâche peut être associée à des expressions formelles qui conditionnent son exécution (pré, post conditions, condition d'itération). Ces conditions sont exprimables à l'aide d'**objets** formellement définis. Tout comme dans le modèle MAD*, les objets de K-MAD peuvent être abstraits ou concrets. Les objets abstraits sont composés des caractéristiques des objets manipulés par les utilisateurs alors que les objets concrets sont des instances de ces objets abstraits. Chaque objet possède un *nom* (texte) et des *attributs*. Les attributs abstraits (attributs des objets abstraits) sont les caractéristiques et sont définis par un nom (texte) et un type de valeur (entier, texte, booléen). Les attributs concrets associent une valeur à chaque caractéristique des objets abstraits. Enfin, des *groupes* sont définis pour contenir les objets concrets.

En plus de ces objets, K-MADe permet la définition **d'événements** (défini par un *nom* (texte)) et **d'utilisateurs** (définis par un *nom* (texte) et un *rôle* (texte)) qui peuvent ensuite être associés aux tâches pour conditionner leur exécution. Un événement peut alors être identifié comme étant un événement déclencheur de la tâche ou un événement déclenché par la tâche.

La Figure 3.1.5 présente un modèle de tâches réalisé avec K-MADe et qui exprime l'activité *Envoyer un nouvel email*.

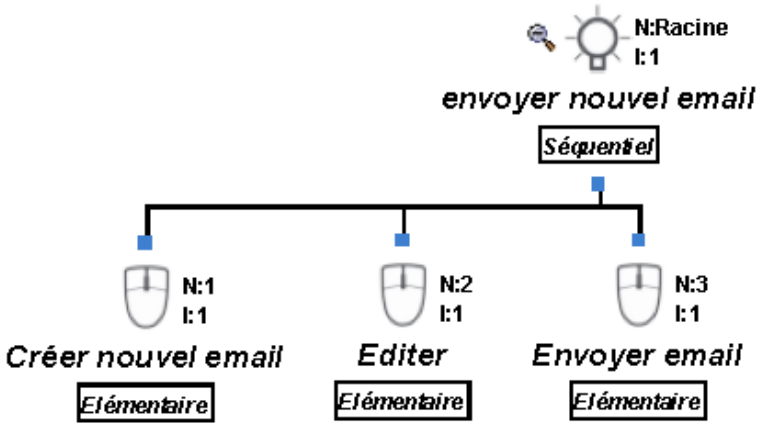


Figure 3.1.5 : *envoyer un nouvel email*, un exemple de notation K-MAD

L'état du monde défini sous forme d'objets, d'événements et d'utilisateurs dans K-MADe, est associé formellement aux tâches via des conditions formelles (pré-conditions, post-conditions, conditions d'itération) et de spécification de liens vers d'autres entités du modèle. Si la spécification des liens entre les entités est également présente dans EUTERPE, la définition de conditions formelles pouvant être évaluées n'est présente dans aucun des autres outils de modélisation. Ces expressions sont évaluées dans l'outil de simulation de K-MADe ce qui permet de prendre en compte l'état du monde lors de la simulation du modèle.

3.1.1.6. L'outil AMBOSS

Aucune publication sur le modèle de tâches sous-tendant l'outil AMBOSS [AMBOSS] n'a été trouvée. C'est pourquoi, nous avons considéré le modèle et l'outil sans distinction et nous induisons le modèle à partir de l'outil.

L'outil AMBOSS vise à assister les analystes qui spécifient leur modèle de tâches spécialement pour les systèmes critiques. Il fournit différentes vues chacune dédiée à un aspect spécifique. Les tâches sont hiérarchiquement décomposées en fonction d'opérateurs de décomposition, tout comme MAD ou K-MAD. AMBOSS manipule quatre concepts : les tâches, les objets, les utilisateurs et les barrières.

Les **tâches** sont définies pas un *nom* (texte), un *emplacement* où la tâche est exécutée (room) (texte) et un type *d'exécutant* (humain ou système). En plus de ces caractéristiques, des informations peuvent être ajoutées comme des *informations* générales (texte) ou la *durée*. Celle-ci est exprimée par trois ensembles d'entiers représentant le nombre de jours, d'heures, de minutes et de secondes. Un de ces quartets est dédié à la durée moyenne, un second à la durée minimale et un dernier à la durée maximale. Enfin, chaque tâche décomposée indique comment ses sous-tâches sont temporellement organisées à l'aide d'opérateurs de décomposition (séquence, parallèle, pas d'ordre, choix).

Les **objets** dans AMBOSS sont des objets abstraits (auxquels il n'est donc pas possible d'attribuer de valeur ni de procéder à une évaluation). Ils sont définis par un *nom* (texte) et un *type* (physique ou information). De plus, chaque objet peut lui-même contenir d'autres objets.

Les **utilisateurs** des tâches sont définis par un *nom* (texte). Enfin, les **barrières** permettent l'expression de mécanismes de protection des tâches.

La Figure 3.1.6 présente un modèle de tâches réalisé avec l'outil AMBOSS. Les icônes dans les parties supérieures des tâches permettent d'accéder aux fenêtres permettant d'éditer les différentes caractéristiques de la tâche comme l'acteur, la durée de la tâche…

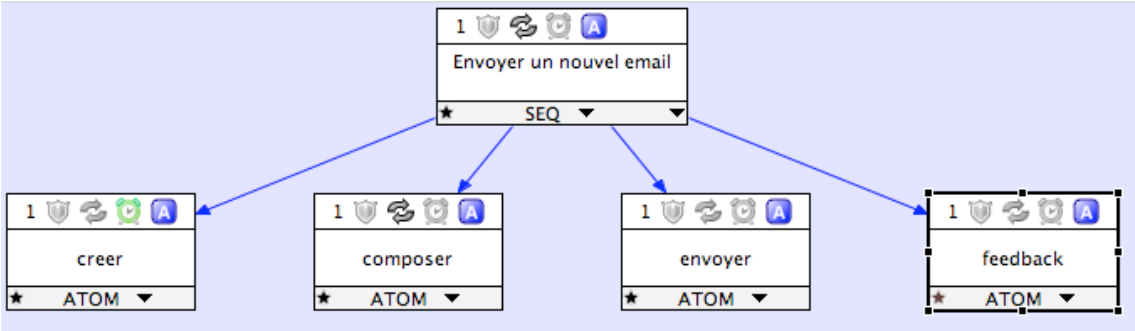


Figure 3.1.6 : envoyer un nouvel email, un exemple réalisé avec AMBOSS

3.1.1.7. Discussion sur les modèles et les outils

Avant de comparer précisément ces six modèles de tâches au travers de leur outil, trois premières remarques peuvent être faites. Alors que tous les modèles sont hiérarchiques, la décomposition des tâches ne suit pas le même processus. Les différences sont présentées dans la suite.

De plus, l'utilisation des modèles au travers les outils nous démontré que, bien que des différences de vocabulaire dans la définition des concepts existent, ces concepts sont les mêmes. Ces différences semblent être dues aux origines différentes des modèles et aux buts différents de leurs concepteurs. Même si les termes utilisés dans les outils ne sont pas les mêmes, nous considérons dans la suite ce qu'ils représentent indépendamment de leur terminologie.

Enfin, nous avons observé une certaine distance entre les modèles et leurs implémentations dans les outils. Par exemple, Diane+ prévoit dans son modèle la définition d'objets (*data*) alors que l'outil ne permet leur utilisation. Cette distance entre les concepts qui peuvent être définis via l'utilisation des outils et ceux définis par les études ayant mené à la définition des modèles rend la modélisation par les outils incomplète. Dans ce document, nous avons choisi d'étudier uniquement les modèles du point de vue de leur outil en ne considérant que les modèles implémentés.

Nous avons basé cette étude comparative sur l'étude préliminaire proposée dans [Lucquiaud 2005b] en réutilisant les critères de comparaison utilisés.

3.1.2. Comparaison des modèles de tâches

Dans cette section, nous présentons une comparaison des notations et outils présentés précédemment : CTT (CTTE), Diane+ (TAMOT), GTA (EUTERPE), MAD (IMAD), K-MAD (K-MADe) et AMBOSS. Comme nous l'avons dit précédemment, nous avons étudié ces notations par leur utilisation au travers leur outil (comment les

outils permettent d'utiliser les modèles). Nous comparons ces modèles suivant deux principaux points : le *pouvoir d'expression* pour chacun des composants du modèle et leur *pouvoir d'interrogation*.

3.1.2.1. Le pouvoir d'expression

Le pouvoir d'expression définit la capacité d'un modèle à exprimer les besoins des utilisateurs. Cette capacité est obtenue par la sémantique des composants des modèles. La première comparaison des six outils : CTTE, TAMOT, EUTERPE, IMAD, K-MADe et AMBOSS fait dégager quatre composants : les tâches, les objets, les événements et les utilisateurs. Nous comparons, dans la suite, le pouvoir d'expression de ces quatre composants pour chacun des outils.

Pouvoir d'expression des tâches

Les tâches sont définies par différents composants. Chacun d'entre eux joue un rôle particulier. Nous avons identifié trois différents rôles. Le premier est la définition des tâches (*information et caractéristiques*). Les deux autres expriment l'organisation temporelle des tâches. Cette organisation temporelle suit deux niveaux. Le premier constitue l'organisation temporelle au niveau de la tâche (*organisation temporelle locale*), il est composé des caractéristiques qui ne concernent que la tâche elle-même. Le second, l'organisation temporelle au niveau du modèle de tâche dans son entier, est composé des caractéristiques ayant des implications sur plusieurs tâches (*organisation temporelle globale*). Pour chacune des catégories, nous faisons un inventaire de toutes les caractéristiques exprimées dans les modèles de tâches et nous indiquons leur présence/absence dans chacun des modèles de tâches.

Informations et caractéristiques des tâches. Cette catégorie regroupe deux entités différentes : les attributs informatifs sur la tâche (les attributs qui ajoutent de l'information sur la tâche mais qui ne la caractérisent pas). Les attributs informatifs sur la tâche sont : le **nom**, le **numéro**, des **remarques** sur la tâche, le **but** (ou objectif), un **support médiatique** et un **lieu** (lieu où la tâche peut être exécutée). Le type des attributs, lorsqu'il est présent dans l'outil, est indiqué dans les six premières lignes du Tableau 3-1.

L'ensemble des caractéristiques des tâches est composé de : l'**exécutant**, l'**importance**, la **fréquence** et la **plateforme**. Au contraire des attributs informatifs, les caractéristiques ont une sémantique précise. Les valeurs des attributs sont souvent prédéfinies dans les outils et de ce fait, permettent l'évaluation des attributs. Les modèles identifient ces caractéristiques par l'utilisation de différents mots même s'ils représentent le même concept. Ces différences de vocabulaire peuvent être expliquées par la situation de la création des modèles initiaux [Limbourg et Vanderdonckt 2003]. La seconde partie du Tableau 3-1 détaille pour chaque outil : le nom utilisé pour identifier pour les caractéristiques, quand il diffère du nom du concept (en *italique*) ; et les valeurs prédéfinies.

Les six outils définissent comme données communes d'une tâche : son **nom**, des **remarques** et son **exécutant**. Le nom et les remarques sont des caractéristiques informelles exprimées par du texte.

Au contraire, l'**exécutant** de la tâche est défini à partir de valeurs existantes dans l'outil. Même si les six outils utilisent différents termes pour représenter les différents exécutants, nous pouvons définir quatre types d'exécutants différents. Lorsque l'exécutant est connu, la tâche peut être réalisée par un être humain (*humain*), par un système (*système*) ou être le résultat de la réaction du système par rapport à l'action d'un humain (*interactif*). Le dernier type d'exécutant est le type utilisé quand l'exécutant de la tâche n'est pas encore défini (*Composé*). Les exécutants existants sont différents pour chacun des outils (voir le Tableau 3-2).

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Nom	Texte	Texte	Texte	Texte	Texte	Texte
Numéro	-	-	-	Texte automatiquement généré	Texte automatiquement généré	-
Remarque	Texte	Texte	Texte	Texte	Texte	Texte
But	Texte	-	Texte	Texte	Texte	-
Media	-	-	Path	-	Path	-
Room	-	-	-	-	-	Texte
Exécutant	*Catégorie* utilisateur, système, interaction, abstrait	*Style :* manuel, automatique, interactif	*Type :* compléxe, unitaire (utilisateur ou système), interaction, ∅	*Modalité :* manuel, automatique, interactif	interactif, utilisateur, système, abstrait	*Rôle :* humain, système
Importance	-	-	normal, critique, dangereux, optionnel, ∅	important, relativement important, additionnel	très important, moyennement important, peu important	-
Fréquence	haut, moyen, peu, ∅	-	-	haut, moyen, peu, ∅	haut, moyen, peu, ∅	-
Plateforme	PDA, desktop, Mobile, voice	-	-	-	-	-

Tableau 3-1 : Le pouvoir d'expression des informations et des caractéristiques des tâches

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Interactif	√	√	√	√	√	-
Système	√	√	√	√	√	√
Humain	√	√	√	√	√	√
Composé	√	-	√	-	√	√

Tableau 3-2 : Les types d'exécutants dans les outils de modèle de tâches

Les tâches interactives (tâches déclenchées par l'action d'un utilisateur et entraînant une réaction du système) ne sont pas décrites dans les modèles de tâches AMBOSS. Les tâches de cet outil sont composées (tâches abstraites) ou réalisées par un humain ou un système seuls. Ces types d'exécutant impliquent un faible niveau d'abstraction de la description des activités.

Dans IMAD et K-MADe, les tâches interactives sont définies comme des tâches qui peuvent être composées de tâches réalisées par des exécutants différents (humain ou système). Ils peuvent, de ce fait, jouer le rôle de tâches décomposées.

Dans EUTERPE, une tâche peut être créée sans aucun exécutant. De plus, même s'il est possible de définir si une tâche est composée ou élémentaire, cet outil ne fait pas de distinction entre un exécutant humain ou système. Enfin, lorsqu'une tâche est interactive, l'interaction peut être précisée parmi les interactions définies par le concepteur.

Certains outils ajoutent la possibilité de fournir des commentaires supplémentaires aux tâches via l'ajout du **but** des tâches (CTTE, EUTERPE, IMAD et K-MADe), un chemin d'accès à un **support médiatique** (EUTERPE et K-MADe) ou le **lieu** où les tâches peuvent être exécutées (AMBOSS). En plus de ces informations, d'autres caractéristiques peuvent également être ajoutées : l'**importance** des tâches (EUTERPE, IMAD et K-MADe), la **fréquence** (CTTE, IMAD et K-MADe) et la **plateforme** sur laquelle la tâches est réalisée (CTTE). L'ajout de la fréquence et de la plateforme à la description des tâches dans CTTE permet l'utilisation des modèles de tâches CTT pour la génération automatique d'interfaces [Mori, et al. 2004]. Nous n'avons trouvé aucune utilisation des autres informations apportées si ce n'est leur rôle informatif.

Ces informations et caractéristiques permettent la description des tâches. Or, aucune règle n'est établie pour définir les caractéristiques et les informations des tâches pendant le processus de conception. Par exemple, pour la caractéristique de fréquence, le concepteur doit choisir le niveau de fréquence parmi un ensemble de valeur. Cependant, ce choix de valeur dépend de l'appréciation du concepteur. De plus, peu de caractéristiques sont formellement définies. Ces deux raisons limitent l'utilisation des caractéristiques des tâches par les outils.

Les modèles de tâches sont utilisés pour concevoir des applications interactives ainsi que pour aider les spécialistes en ergonomie à détecter les

dysfonctionnements [Balbo, et al. 2004]. Les modèles de tâches doivent pour cela intégrer les concepts et les notions habituellement utilisés du domaine de l'ergonomie. Cependant, les outils de modèles de tâches existant sont plus proches des besoins des spécialistes d'IHM que de ceux des spécialistes d'ergonomie [Couix 2007]. Par exemple, seuls les tâches des outils IMAD et K-MADe ont pour caractéristiques le **numéro** de la tâche, alors que celui-ci est utilisé en ergonomie pour organiser les tâches entre elles.

Organisation temporelle locale. Les attributs définissant l'organisation temporelle locale sont l'ensemble des caractéristiques des tâches qui influencent l'organisation temporelle de la tâche indépendamment de ses sous-tâches. À partir de l'utilisation des outils, nous avons identifié six caractéristiques dans cet ensemble : le **durée** de l'exécution de la tâche, le caractère **optionnel** de la tâche, sa **priorité**, son caractère d'**interruption**, l'**itération** et son caractère **coopératif**.

L'attribut **durée** de la tâche donne une information temporelle à propos de l'exécution de la tâche. Nous pouvons regrouper les informations temporelles de la tâche en deux types principaux. Le premier est constitué des indications de durée (pendant combien de temps la tâche est-elle exécutée ?) et le second, est constitué des indications sur le temps d'exécution de la tâche (quand l'exécution de la tâche débute-t-elle ? quand termine-t-elle ?).

Le caractère **optionnel** d'une tâche indique si la tâche doit être exécutée pour que l'activité soit considérée comme complète ou non (elle est optionnelle dans le cas où elle n'est pas indispensable). Une tâche peut être optionnelle pour différentes raisons (tâche non obligatoire dans le processus système, tâche nécessaire que sous certaines conditions…) et son exécution peut être dépendante des souhaits/buts/préférences de l'utilisateur (par exemple, compléter un champ optionnel d'un formulaire) ou des besoins du système (par exemple : ajouter du papier pour imprimer est nécessaire que si l'imprimante manque de papier).

La définition de l'attribut de **priorité** de la tâche n'est pas très claire. Cet attribut semble représenter la priorité de l'exécution d'une tâche par rapport aux autres tâches. Cependant, aucun outil utilise cet attribut, ni dans l'un de ses modules (outil de simulation) ni dans les outils de conception basés sur ces modèles de tâches (comme TERESA [Mori, et al. 2004], par exemple).

Définir une tâche comme **interruptible** indique que son exécution peut être stoppée ou non.

Dans le but d'indiquer que certaines tâches peuvent être exécutées plusieurs fois, une caractéristique d'**itération** peut être précisée.

Le dernier attribut utilisé pour décrire l'organisation temporelle locale que nous avons identifiée dans cette étude est l'aspect **coopératif** de la tâche. Cet attribut est une part de l'organisation temporelle locale car, l'exécution d'une tâche collaborative nécessite que tous les acteurs de cette tâche soient disponibles.

Dans le Tableau 3-3, pour chacun des outils, nous indiquons le type ou les valeurs utilisées pour exprimer ces caractéristiques.

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Durée	3 entiers (min, moyen, max)	-	-	3 entiers (début, Fin, Durée)	texte	3x4 entiers (durée, min, max) (H, D, M, S)
Optionalité	Booléen	Booléen	pas clair	Booléen	Booléen	-
Priorité	-	-	-	Entier	faible, moyen, haut	-
Interruptibilité	Operateur	-	-	Booléen	Booléen	-
Itération	Booléen	Integer (min, max)	-	-	Expression formelle	-
Collaborativité	Booléen	-	-	-	-	-

Tableau 3-3 : Le pouvoir expressif de l'organisation temporelle locale

Seul AMBOSS permet la définition de la **durée** sous forme de temps mesuré. Dans CTTE, IMAD et AMBOSS, la durée est composée de trois attributs (afin de spécifier la couverture de la durée (minimum, maximum et en moyenne) ou la place de la tâche dans l'organisation temporelle de l'activité (le départ, la fin et la durée de la tâche)). K-MADe permet la définition de la durée sous forme textuelle, comme indication informelle. Cependant, la durée d'une tâche semble être liée d'avantage à un scénario particulier qu'à la description de l'activité.

La **priorité** d'une tâche est utilisée uniquement dans IMAD et K-MADe. Cependant, cette caractéristique n'a pas de sémantique précise ni de règles. Par exemple, quelle est la signification de la priorité de tâches séquentiellement organisées. De plus, l'attribution d'une valeur à une tâche est subjectif, car c'est au concepteur du modèle de tâche d'évaluer le niveau de priorité de la tâche alors que cette caractéristique de la tâche change pour chaque scénario et en fonction des autres tâches.

CTTE, IMAD et K-MADe indique l'**interruptibilité** des tâches, en utilisant un booléen pour IMAD et K-MADe et les opérateurs pour CTTE. Dans CTTE, une tâche est interruptible si elle peut être suspendue par l'exécution d'une de ces tâches sœurs. Cependant, cet outil considère que les interruptions interviennent à des moments prédéfinis (pas comme un coup de téléphone par exemple). K-MADe et IMAD permettent d'indiquer si une tâche est interruptible ou non sans avoir à spécifier par quel élément elle est interrompue. De plus, IMAD et K-MADe spécifient uniquement un seul type d'interruption qui suspend momentanément l'exécution de la tâche.

Seuls trois outils prennent en compte l'**itération** des tâches : CTTE, TAMOT et K-MADe. Alors que CTTE permet l'addition d'une caractéristique d'itération aux

tâches, le nombre d'itération ne peut être spécifié. Dans TAMOT le nombre minimal et maximal d'exécutions des tâches peuvent être indiqués. K-MADe permet quant à lui de définir un nombre fixe d'itérations ou une expression qui conditionne l'exécution de la répétition de la tâche.

Enfin, dans CTTE, une tâche peut être définie comme étant **collaborative** (i.e. : différents acteurs peuvent exécuter la tâche). Cette caractéristique est nécessaire dans la description des tâches CTTE car le modèle CTT recommande la conception d'un modèle de tâche par acteur. Donc, l'attribut de collaboration de la tâche permet de lier les différents acteurs modélisés pour une même activité.

L'organisation temporelle globale. Cette catégorie concerne uniquement les opérateurs temporels. Avant d'analyser le pouvoir expressif des opérateurs temporels (Tableau 3-4), nous spécifions comment ils sont utilisés dans les outils. Alors que CTTE, EUTERPE, IMAD, K-MADe et AMBOSS décomposent leurs tâches en sous-tâches, en construisant des arbres de tâches, TAMOT utilise des *fenêtres*. De plus, seul CTTE utilise les opérateurs pour lier les tâches sœurs alors que les autres outils les utilisent pour spécifier la décomposition des sous-tâches.

Pour exprimer l'organisation temporelle globale, nous avons identifié quatre opérateurs et deux types d'interruption. Les interruptions peuvent être définitives ou non : les tâches interrompues peuvent continuer leur exécution une fois la tâche l'ayant interrompue terminée (*sans abandon*) ou peuvent être définitivement interrompues (*avec abandon*). Ces opérateurs sont : la *séquence* (les tâches sont exécutées par l'ordre défini par l'analyste), le *parallélisme* (les tâches sont parallèlement exécutées), *sans ordre* (les tâches sont toutes exécutées mais sans ordre prédéfini) et le *choix* (seule une tâche est exécutée). Le Tableau 3-4 résume la présence des opérateurs temporels pour chacun des outils.

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Séquence	√	√	-	√	√	√
Parallélisme	√	√	-	√	√	√
Sans ordre	√	√	-	√	√	√
Alternatif	√	√	-	√	√	√
Désactivation	√	-	-	-	-	-
Interruption	√	-	-	-	-	-

Tableau 3-4 : Pouvoir d'expression des opérateurs temporels

Dans CTTE, d'autres opérateurs permettent de spécifier lorsqu'il y a des échanges d'information. Cependant, cette indication est un élément informatif qui ne modifie pas l'organisation temporelle globale (deux tâches liées par un opérateur sans échange d'information ont le même comportement que deux tâches liées par le même opérateur avec échange d'information). C'est pourquoi, nous n'avons pas distingué les deux types d'opérateurs temporels dans notre comparaison.

Les opérateurs temporels ne sont pas prédéfinis dans EUTERPE. Les liens entre les tâches indiquent uniquement qu'une tâche est composée d'autres tâches. Les

opérateurs temporels ne peuvent être définis qu'informellement par des étiquettes sur les liens entre les tâches.

Les cinq autres outils (CTTE, TAMOT, IMAD, K-MADe et AMBOSS) permettent la spécification de tâches organisées **séquentiellement**, **parallèlement**, **sans ordre** ou par **choix**. Seul CTTE prend en compte les **interruptions** (*désactivation* et *interruption*) en utilisant les opérateurs temporels. Ces opérateurs permettent la spécification de l'interruptibilité des tâches.

Enfin, lors de notre étude nous avons noté que l'utilisation combinée de certains opérateurs temporels et caractéristiques des tâches peut créer des ambiguïtés. Par exemple, quelle est la signification d'une tâche optionnelle qui est exécutée uniquement si elle est sélectionnée (décomposition alternative) ? Dans [Paternò 2004], Paternò indique les problèmes d'ambiguïté qui peuvent survenir en utilisant les opérateurs temporels du modèle CTT.

Le pouvoir d'expression des objets, événements et utilisateurs

En plus des tâches, d'autres éléments sont considérés comme étant essentiel pour la modélisation des tâches [Dix 2008] : les **utilisateurs**, les **événements** et les **objets manipulés**, qui augmentent le pouvoir d'expression des modèles de tâches. Les utilisateurs sont définies comme étant des êtres humains qui jouent un rôle dans l'activité modélisée. Ils exécutent les tâches où sont à leur origine (comme peut l'être un superviseur). Pendant le déroulement de l'activité, certains événements peuvent intervenir. Les outils de modélisation intègrent certains de ces concepts. Le Tableau 3-5 résume la présence de ces entités dans chacun des outils.

Dans CTTE, les objets manipulés sont des caractéristiques des tâches [Paternò 2004]. Ils sont caractérisés par un **nom** (texte) ; une **classe** parmi *texte*, *numérique*, *objet*, *description* et *position* ; un type parmi *percevable* (objet présentant une information ou permettant les actions des utilisateurs) et *application* (interne au système) ; un **mode d'accès** (*lecture seule* ou *modification*) ; une **cardinalité** (*faible*, *moyenne* ou *haute*) ; et une **plateforme** sur laquelle l'objet est présenté. L'outil de simulation de CTTE ne prend pas en compte les objets. Certaines caractéristiques des objets sont cependant utilisées pour la génération des interfaces [Mori, et al. 2004] comme la cardinalité qui peut être utilisée pour définir si l'objet sera représenté par un champ texte, une liste ou une combo box…CTTE n'inclut pas les utilisateurs et les événements comme des concepts spécifiques.

Le modèle de tâche Diane+ intègre les objets manipulés, nommé *data*, et les utilise pour définir des conditions. Cependant, dans l'outil TAMOT qui permet l'édition des modèles de tâches selon la notation Diane+, il n'est pas possible de définir les *data* et les pré conditions (seules conditions exprimables) ne sont éditables que sous forme textuelle.

Dans EUTERPE, les objets manipulés sont des composants indépendants. Ils sont caractérisés pas un **nom** (texte), une **liste d'attributs** (chaque attribut est composé par un **nom** (texte) et une **valeur** (texte)) ; et une **liste d'utilisateurs** (les utilisateurs qui peuvent manipuler l'objet). Les utilisateurs sont définis comme des **agents étiquetés**.

Des relations entre les agents et les objets peuvent être définis (*propriétaire*, *créée*, *détruit*, *utilise/inspecte*, *modifie*). Un **agent** est défini par un **nom** (texte) ; un **type** (*individuel*, *organisation*, *système informatique*) ; et un **rôle** (l'*ensemble des tâches* réalisées par un agent). De plus, EUTERPE permet la définition d'**événements**. Ceux-ci sont composés d'un **nom** (texte) et de l'**ensemble des tâches** qui lui sont liées.

Deux types d'objets sont présents dans le modèle MAD*. Ils correspondent aux notions orientées objet de classe (**objets abstraits**) et d'instance (**objets concrets**). Chaque **objets** est composé d'un **nom** (texte), un **numéro** (identifier par une classe d'objet (entier)) et une **liste d'attribut**. Dans MAD*, les attributs des objets abstraits sont caractérisés par un **nom** (texte) et un **type** (*texte*, *booléen*, *entier*) et les attributs des objets concrets par un **nom** (texte) et une **valeur**. Dans IMAD, les objets abstraits (nommés classes) sont définis par un nom, un numéro et une liste d'attributs (numéro, nom et type). Les instances (les objets concrets) de chaque objet concret peuvent être définies. Les objets concrets sont définis par un nom uniquement, et les valeurs des attributs ne peuvent être saisies (les champs valeur ne sont pas éditables). Certaines caractéristiques sont ajoutées à la définition des objets abstraits : la méta-classe (lien de généralisation) ; une sous-classe (lien de spécification) ; une condition sur le nombre d'instances (restriction à une instance). Cependant, dans IMAD, les deux types d'objets ne sont pas différenciés. En plus des objets manipulés, IMAD permet la définition des utilisateurs comme une caractéristique de tâche (texte).

K-MADe dispose de tous les types d'objets disponibles dans l'ensemble des autres outils : **objets manipulés**, **événement** et **utilisateurs**. Les objets manipulés sont des deux types ayant été identifiés : abstraits et concrets. Chacun des objets (concrets et abstrait) est composé d'un ensemble d'attributs. Un attribut abstrait (attribut d'un objet abstrait) a un **nom**, un **type** (parmi *texte*, *entier*, *booléen*) alors qu'un attribut concret (attribut d'un objet concret) a un **nom** et une **valeur** (du type de l'attribut concret dont il est instance). Enfin, les objets concrets sont organisés en groupes. Chaque groupe n'étant constitué que d'objets (instances) du même objet abstrait. Les objets manipulés de K-MADe peuvent être utilisés pour définir des expressions évaluables : les préconditions, les expressions de manipulation des objets et les conditions d'itérations. Les événements et les utilisateurs sont définis par des étiquettes (texte). Ces entités peuvent alors être associées aux tâches.

L'environnement AMBOSS permet de définir les objets manipulés et les associations entre les tâches et les objets ; ainsi que de définir les utilisateurs. Les objets manipulés sont composés d'un **nom** (texte), d'une **description** (texte) ainsi que de son **type** (physique ou informatif). Chaque objet peut contenir d'autres objets (définition d'objets composés d'autres objets).

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Objets abstraits	-	-	-	√	√	√
Objets concrets	√	-	√	√	√	-
Nom d'attribut	non formel	-	√	√	√	√
Type d'attribut	non formel	-	-	√	√	non formel
Valeur d'attribut	non formel	-	√	-	√	-
Utilisateurs	propriétaire de la tâche	-	√	√	√	√
Evénements	-	-	√	-	√	-

Tableau 3-5 : Le pouvoir d'expression des objets

L'un des principaux usages des objets et l'expression de pré et post-conditions. Le Tableau 3-6 représente la définition de ces attributs dans les outils. Tous les formalismes incluent des pré-conditions associées aux tâches. Alors que CTTE, TAMOT, EUTERPE et IMAD expriment ces pré-conditions textuellement, AMBOSS exprime les pré-conditions comme étant la nécessaire présence de barrières et de messages. Cependant, ces deux éléments sont exprimés sous forme textuelle également. Enfin, K-MADe sous forme d'expression formelle évaluant les valeurs des objets définit des pré conditions. Afin de spécifier les conséquences de l'exécution des tâches sur les objets manipulés, EUTERPE et IMAD incluent des expressions de post-conditions (sous forme de chaînes de caractères également). K-MADe quant à lui propose une définition formelle et évaluable des actions des tâches sur les objets. Donc, seul K-MADe permet la définition d'expressions évaluables en permettant la prise en compte des valeurs des objets.

Outils	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Pré condition	texte	texte	texte	texte	expression formelle	barrières+messages
Post condition	-	-	texte	texte	-	-
Action	-	-	-	-	expression formelle	-

Tableau 3-6 : Le pouvoir d'expression des attributs des tâches qui manipulent les objets

Le Tableau 3-5 représente le pouvoir d'expression des objets dans les six outils de modélisation des tâches. À part TAMOT (qui ne contient pas les *data* prévus dans le modèle Diane+), tous les outils contiennent le concept d'objet. Ils peuvent être regroupés en deux catégories. La première est composée des outils considérant les objets comme étant des attributs de tâche (comme CTTE), et la seconde est composée

des outils considérant les objets comme étant des entités indépendantes (comme EUTERPE, IMAD, K-MADe et AMBOSS). La définition des objets comme attribut de la tâche implique que les objets manipulés soient transférés d'une tâche à l'autre par le biais des opérateurs.

Dans CTTE, un attribut des objets est sa cardinalité. Il est utilisé pour aider le concepteur à définir l'élément interactif qui représente cet objet. De plus, les objets perceptibles peuvent être un tableau ou une fenêtre et de ce fait, cet outil associe les objets interactifs aux tâches. L'introduction de ces éléments (cardinalité et objets perceptibles) dans le modèle de tâches illustre le lien entre les objets manipulés et la présentation des interfaces. Cette définition est proche d'un point de vue du système alors que dans tous les autres outils, les concepts d'objets se rapprochent du point de vue ergonomique. Alors que les objets de CTTE sont concrets (une valeur peut être associée à un objet dès sa création), IMAD et AMBOSS ne permettent pas de donner une valeur aux attributs des objets restant ainsi à un haut niveau d'abstraction. Ce niveau d'abstraction gèle la manipulation des objets des modèles de tâches qui nécessite des moyens de modifications dynamique des valeurs des objets. L'état du monde est donc limité à une expression statique. Seul K-MADe propose une définition des deux niveaux d'abstraction des objets.

La validation des conditions est nécessaire à l'exécution des tâches. Donc, pour permettre la validation des modèles de tâches par les utilisateurs et donc la vérification du séquencement des tâches (en utilisant les outils de simulation), ces conditions doivent pouvoir être évaluées. Afin d'évaluer les conditions, la définition des entités utilisées et les conditions doivent être formelles. Cependant, les outils CTTE, TAMOT, EUTERPE et IMAD définissent les conditions en utilisant des chaînes de caractères non-évaluables et ne manipulent pas les objets définis. La même observation peut être faite à propos des post-conditions lorsqu'elles sont présentes (dans EUTERPE et IMAD). Seul AMBOSS intègre la vérification de pré conditions avant l'exécution des tâches. Cependant, ces pré-conditions sont exprimées par le présence de *barrières* et de *messages* (définis par des chaînes de caractères).

Seuls EUTERPE et K-MADe considèrent les **événements** comme des objets. Enfin, hormis TAMOT, tous les outils prennent en compte la notion d'**utilisateurs** mais de différentes manières. Dans CTTE, un arbre de tâche est défini pour chaque acteur (par exemple, une arbre pour le vendeur, un arbre pour l'acheteur pour une activité de vente) regroupés sous forme d'un arbre global (ici, un arbre modélisant l'activité de vente). Il y a donc un arbre pour l'organisation et un arbre pour chacun des acteurs de cette organisation. Cependant, cette modélisation ne permet pas les échanges d'informations d'un acteur à l'autre et limite la description des activités de groupe. IMAD considère les utilisateurs comme étant des attributs des tâches et de ce fait, ne fait pas de distinction entre les données de l'utilisateur et celles de la tâche. Cependant, il y a deux types de données qui peuvent avoir des conséquences sur les autres. Par exemple, l'acteur d'une tâche a une fonction dans un groupe. Cette fonction peut justifier la présence d'une tâche qui modifie les valeurs des objets. Donc, les informations sur les utilisateurs et les objets manipulés sont liés mais pas dépendants les uns des autres. Au contraire, EUTERPE et K-MADe identifient deux différents concepts : le rôle et les agents pour EUTERPE et l'utilisateur et l'acteur pour K-MADe. Enfin, dans AMBOSS, différents êtres humains et systèmes peuvent être créés comme

des concepts spécifiques associés aux tâches. La définition des utilisateurs comme un concept indépendant des tâches permet l'association d'un même utilisateur à toutes les tâches exécutées par cet utilisateur.

3.1.2.2. Le pouvoir d'interrogation

Le pouvoir d'interrogation d'un modèle est sa capacité à permettre l'utilisation de sa sémantique. Nous avons identifié deux principales fonctionnalités dans les outils qui permettent cette utilisation : la **vérification du modèle** et sa **simulation**. La capacité d'interrogation des modèles garantit que le modèle est conçu en accord avec son éditeur. La sémantique des modèles permet l'utilisation des modèles de tâches par d'autres approches afin de générer les interfaces (comme dans [Mori, et al. 2003]), la migration entre différentes plateformes (comme dans [Calvary, et al. 2001]), ou l'échange de données entre modèles de tâches exprimés à l'aide de formalismes différents [Limbourg, et al. 2001b]. Nous considérons que ces différents travaux ne dépendent pas des outils d'édition des modèles de tâches. Ils dépendent d'outils indépendants qui peuvent utiliser des modèles de tâches en entrée. Par exemple, l'outil de génération TERESA utilise des modèles de tâches CTT (conçu à l'aide de l'outil CTTE) indépendamment de CTTE lui-même pour générer des interfaces utilisateur.

Vérification

Quatre outils : CTTE, EUTERPE, K-MADe et AMBOSS permettent la vérification de la grammaire des modèles édités. Cependant, aucun ne permet la vérification de toutes les entités. Tous vérifient la cohérence grammaticale de la décomposition des tâches (hiérarchique (voir les lignes 1, 2 et 4 de l'exemple avec K-MADe sur la Figure 3.1.8)), en indiquant par exemple si une tâche interactive est décomposée en tâches de types différents. Les outils CTTE, K-MADe et AMBOSS permettent une vérification grammaticale des opérateurs utilisés. Cette vérification a pour but de détecter certaines erreurs de cohérence des opérateurs et de leurs utilisations. Par exemple, les outils détectent la présence d'un opérateur temporel alors que la tâche est élémentaire (AMBOSS et K-MADe) ou qu'elle n'a pas de sœur (CTTE).

Comme nous l'avons montré dans les sections précédentes, CTTE, EUTERPE, IMAD, K-MADe et AMBOSS intègrent les objets dans la description complète des modèles de tâches.

Cependant, seul EUTERPE permet la vérification grammaticale de ces éléments. Il vérifie deux types de contraintes sur les objets : la vérification des contraintes de *cardinalité* et la vérification des *contraintes de type* (Figure 3.1.7). Les *contrainte de cardinalité* sont les contraintes sur la cohérence entre les entités définies et celles utilisées. Par exemple, EUTERPE détecte la présence d'événements qui ne sont déclenchés par aucune tâche ou la présence d'agent sans rôle. Les *contraintes de type* détectent les incohérences dans la définition des types des entités comme la composition d'un objet par lui-même.

Enfin, même si tous les outils permettent la description de conditions (comme les pré et les post conditions), seul K-MADe réalise des vérifications sur ces conditions en utilisant leur sémantique (voir les lignes 3 et 5 de la Figure 3.1.8). Les types de vérification grammaticale pour chaque outil sont résumés dans le Tableau 3-7.

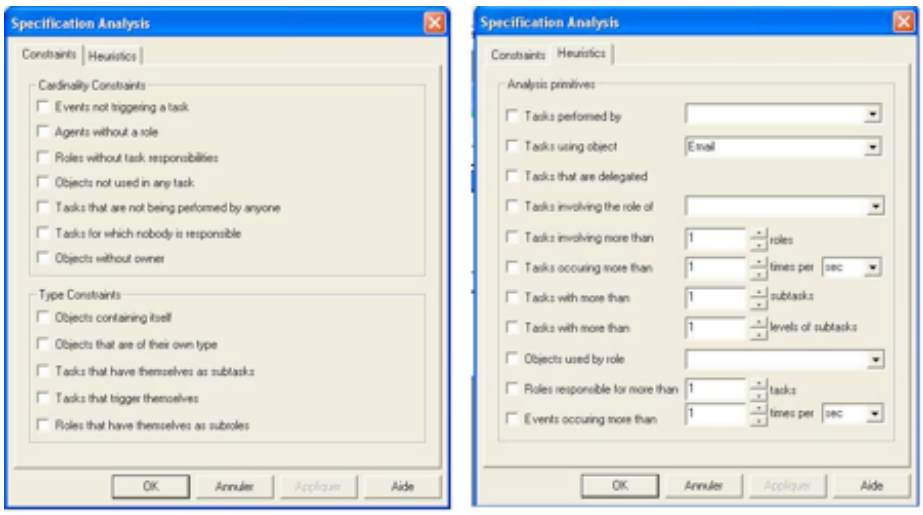


Figure 3.1.7 : Vérification des contraintes et des heuristiques de modèles EUTERPE

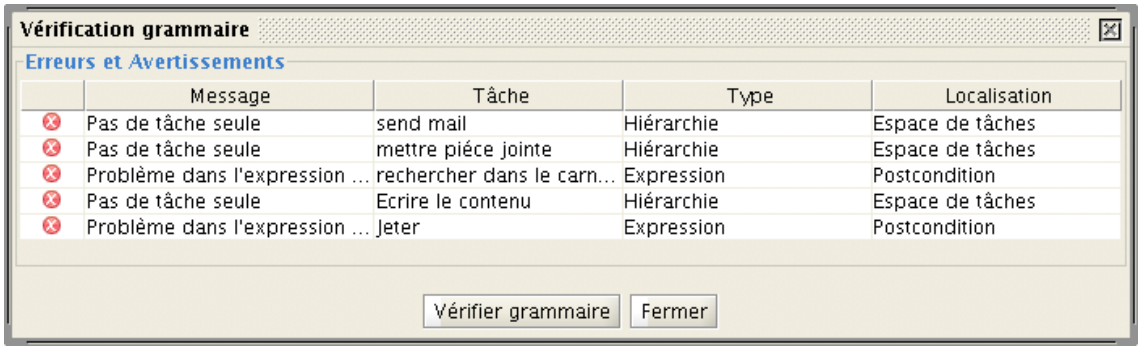


Figure 3.1.8 : Vérification de la grammaire dans K-MADe

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Hiérarchie	√	-	√	-	√	√
Opérateur	√	-	-	-	√	√
Entité	-	-	√	-	-	-
Expression	-	-	-	-	√	-

Tableau 3-7 : Les vérifications dans les outils

Processus de Simulation

La dernière capacité que nous avons étudiée est la simulation (Tableau 3-8). En raison du manque de sémantique clairement définie dans les outils, seuls trois d'entre eux, CTTE, K-MADe et AMBOSS permettent la simulation des modèles. Cette observation augmente la capacité d'utiliser les composants des modèles de tâches et leur validation. Certains formalismes de modèles de tâches sont clairement conçus pour permettre l'édition des modèles et faciliter la communication entre les différents intervenants (comme EUTERPE qui ne définit formellement aucun opérateur temporel). Mais pourquoi les autres outils rendent-ils nécessaire la définition de composants formels qui ne sont pas utilisés ?

CTTE et AMBOSS simulent les modèles en fonction de l'organisation temporelle spécifiée par les opérateurs. Les scénarios peuvent alors être enregistrés et rejoués. Un scénario est une instance particulière du modèle de tâches (une séquence spécifique des tâches). Cependant, comme les objets ne sont pas manipulés par les tâches en utilisant des références sur les entités, CTTE et AMBOSS ne simulent pas l'état des objets. Au contraire, K-MADe permet de prendre en compte la dynamique de la manipulation des objets (Figure 3.1.9). AMBOSS prend en compte les conditions d'exécution des tâches. Ces conditions sont composées de barrières et de messages devant être présents. Ces barrières et messages sont exprimés par des chaînes de caractères. Par exemple, l'entrée d'un mot de passe pour utiliser une application peut être exprimée par une barrière.

	CTTE	TAMOT	EUTERPE	IMAD	K-MADe	AMBOSS
Opérateurs	√	-	-	-	√	√
Objet	-	-	-	-	√	-
Expression	-	-	-	-	√	√
Enregistrement/Rejeu	√	-	-	-	√	√

Tableau 3-8 : Les outils de simulation dans les outils de modélisation de tâches

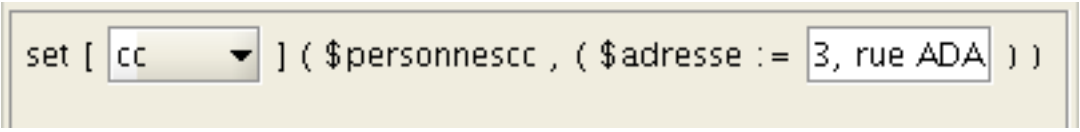


Figure 3.1.9 : Exemple d'évaluation d'une expression modifiant l'état des objets dans K-MADe

3.1.2.3. Bilan de l'étude comparative des modèles de tâches

Afin de comparer le pouvoir d'expression des modèles de tâches, nous avons étudié six outils de modèles de tâches : IMAD, CTTE, TAMOT, EUTERPE, K-MADe et AMBOSS. Nous avons comparé le pouvoir expressif de leurs composants : les tâches, les objets, les événements et les utilisateurs. Nous avons également comparé

comment ces composants sont utilisés dans les outils : lors de la vérification des modèles produits ou la simulation.

La première observation issue de cette comparaison concerne la **distance entre les modèles et les outils associés** (exception faite d'AMBOSS pour lequel le modèle n'a pas été décrit indépendamment de l'outil). Comme nous l'avons indiqué précédemment, l'exemple le plus illustratif est le cas du modèle Diane+ dans lequel les objets (*data*) sont inclus alors que l'outil TAMOT ne les prend pas en compte. Les modèles de tâches résultent d'étude sur les besoins d'expressivité. Donc, les éléments qui composent les modèles de tâches sont des éléments qui ont été définis comme essentiels pour la modélisation des tâches. C'est pourquoi l'absence d'un ou de plusieurs de ces éléments dans les outils limite le pouvoir d'expressivité des modèles réalisés à l'aide des outils.

Cependant, certains outils ont été spécifiquement implémentés pour être proche des modèles sous-jacents comme l'outil K-MADe. En effet, cette implémentation a été composée de deux étapes différentes. La première étape a eu pour but de coder le noyau EXPRESS [EXPRESS 1994] composé des éléments du modèle K-MAD [Lucquiaud 2005b]. Tous les composants du modèle décrit dans [Lucquiaud 2005b] sont donc inclus. Lors de la seconde étape, l'interface de l'outil a été développée. Elle permet d'utiliser les concepts du modèle et de l'étendre pour faciliter le processus de modélisation (comme l'ajout du type *inconnu* comme exécutant de la tâche). En dehors de ces ajouts, l'outil permet d'exprimer tous les composants des modèles K-MAD donc, l'outil K-MADe et le modèle K-MAD sont très proche.

De plus, les éléments des modèles de tâches manquent parfois de **sémantique** exprimée. Par exemple, les niveaux de priorité d'IMAD et de K-MADe ne sont pas sémantiquement décrits. Cela limite l'utilisation des concepts pour une modélisation complète (comme l'utilisateur ne sait pas pourquoi, ni comment il peut utiliser les concepts, il ne les utilise pas). Bien que la sémantique de tous les composants de K-MADe ne soit pas toujours complètement décrite (comme c'est le cas pour la priorité des tâches), un manuel d'utilisation est joint [Baron et Scapin] à l'outil K-MADe. Dans ce manuel, une définition de chacun des concepts de K-MADe est donnée. Ce document a pour but de faciliter l'utilisation de K-MADe par les concepteurs.

Si la sémantique de certains composants du modèle K-MAD reste floue (comme l'usage de l'interruption en fonction des opérateurs temporels utilisés), la sémantique des composants formels est elle, clairement décrite (comme nous l'avons dit précédemment, il existe deux types de composants, les composants formels et les composants non formels). La **définition formelle des composants** des modèles permet l'utilisation de ces composants pour la vérification des modèles réalisés (en utilisant les outils de simulation de CTTE, K-MADe et AMBOSS par exemple). Le composant le plus communément décrit formellement est l'opérateur de décomposition. Hormis EUTERPE, tous les outils les définissent formellement. En dehors des opérateurs de décomposition, d'autres composants des modèles peuvent être formellement définis dans les outils : le numéro de la tâches (IMAD et K-MADe), l'exécutant de la tâche (tous les modèles) et les caractéristiques temporelles (optionalité (CTTE, TAMOT, IMAD et K-MADe), la priorité (K-MADe), l'interruptibilité (IMAD, CTTE et K-

MADe)). Seul K-MADe permet donc la définition formelle de toutes les caractéristiques pouvant être formellement définies.

De plus, c'est le seul outil qui propose une définition formelle des objets qui peuvent être pris en compte dans la dynamique du modèle. Les objets reflètent l'état du monde. Comme nous l'avons dit précédemment, l'exécution d'une tâche peut être conditionnée par l'état des objets. Comme K-MADe permet la définition formelle des objets, ils peuvent être formellement utilisés pour exprimer les conditions (voir Figure 2.1.9). Ces **conditions formelles** sont évaluées lors de la simulation des modèles de tâches.

Une dernière limitation à l'utilisation des modèles de tâches est la **non-accessibilité aux données et à la sémantique**. Les modèles de tâches peuvent être utilisé comme des documents sur les activités permettant par exemple d'être le support de discussions entre les différents intervenants. C'est pourquoi l'accès aux données exprimées et à leur sémantique est un apport important à l'utilisation des outils de modélisation. Lors de notre étude, nous avons constaté que des disparités quant à l'accès aux données dans les outils. Si certains outils permettent un accès à certaines données sous forme de document (comme c'est le cas pour les objets dans EUTERPE), d'autres ne proposent qu'un accès informatif (comme c'est le cas de CTTE).

Les différents modèles examinés ici sont relativement proches. Cependant, le duo K-MAD/K-MADe se distingue par le fait qu'il propose une véritable implémentation de la majorité des concepts reconnus comme nécessaires à la modélisation des tâches, sous une forme calculable, et donc exploitable formellement. C'est en particulier le seul qui permet de calculer des expressions faisant intervenir des objets, point considéré comme essentiel pour atteindre la complétude de la définition de l'activité [Dix 2008, van der Veer, et al. 1996].

Ces dernières études sont cependant théoriques. Compte tenu du caractère opérationnel de l'outil K-MADe sur ce point, il était intéressant de valider empiriquement ce point. C'est l'objet de la section suivante.

3.1.3. Étude empirique de l'utilisation de K-MADe

Cette section présente une étude cherchant à évaluer les implications pédagogiques (compréhension et utilisation) dans l'emploi d'une approche formelle pour la modélisation des activités des utilisateurs. Bien que K-MADe soit prévu pour être utilisé par des utilisateurs ayant des connaissances différentes (informaticiens, experts en ergonomie…), les sujets de notre étude appartiennent à une unique catégorie d'utilisateurs : des étudiants ayant des connaissances en IHM.

Le but de notre évaluation est l'utilisation des modèles de tâches pour la conception d'applications. Afin de mieux comprendre pourquoi les entités formelles sont utilisées, nous nous intéressons particulièrement à deux aspects : l'apprentissage de

leur manipulation et leur utilisation dans le processus de modélisation des tâches. Suivant ces deux axes, nous divisons notre étude en deux évaluations.

La première vise à définir le processus d'apprentissage des aspects formels dans le processus de modélisation. Dans la seconde évaluation, nous étudions l'utilisation des entités formelles dans la conception des modèles de tâches. L'utilisation d'un outil (notamment lors de la phase d'apprentissage) a pour principal intérêt la limitation de libertés prises avec le formalisme. C'est pourquoi, tout au long de notre étude nous n'avons jamais cherché à évaluer l'utilisation du modèle K-MAD, indépendamment de l'outil K-MADe même si ce choix a entraîné l'intrusion de problèmes liés à l'ergonomie de K-MADe dans nos résultats (observations qui pourront être utilisées pour l'amélioration de K-MADe).

3.1.3.1. Les participants

Pour la première phase de notre évaluation, nous souhaitions des participants dans un contexte d'apprentissage, des étudiants d'un module d'enseignement de conception centrée-utilisateur (contenant la modélisation des tâches) étaient donc tout indiqués. De plus, la modélisation des tâches pouvant être réalisée par des profils de concepteurs différents, nous souhaitions intégrer des profils de participants différents. Nous avons eu l'occasion de réaliser notre évaluation avec deux groupes d'étudiants différents : leurs caractéristiques sont résumées sur la Figure 3.1.10. Tous les participants sont dans leur quatrième année d'étude en université française. Le premier groupe est composé de 48 étudiants en bioinformatique (31 hommes et 17 femmes), et le second groupe est composé de 20 étudiants en informatique (ils étudient l'informatique depuis leur première année d'étude) (19 hommes et 1 femme). Seul un étudiant en informatique (membre du second groupe d'étude) n'est pas un francophone. Tous les participants (du 1er et 2nd groupe) ont suivis le même cours d'IHM.

	Groupe 1	Groupe 2
Nombre	48 (31 hommes/17 femmes)	20 (dont 1 non francophone) (19 hommes/1 femme)
Niveau d'étude	4^{ième} année en université française	4^{ième} année en université française
Domaine d'étude	Bioinformatique	Informatique

Figure 3.1.10 : Les caractéristiques des deux groupes de participants

3.1.3.2. L'organisation de l'étude

L'étude sur l'apprentissage du rôle et de l'usage de K-MADe dans le processus de modélisation des tâches a été divisée en deux étapes. Pendant l'apprentissage (un mois) de la modélisation des activités en utilisant l'outil K-MADe, une première évaluation est menée ; une fois que cette phase d'apprentissage est terminée (ou

considérée comme telle), une seconde évaluation est menée pour identifier l'utilisation faite du formalisme. Cependant, le second groupe (composé des étudiants en informatique) réalise uniquement la seconde (pas la première) évaluation.

Pendant la durée de l'évaluation, des données sur l'utilisation des entités formelles dans la conception des modèles des tâches sont collectées auprès de l'ensemble des étudiants. La Figure 3.1.11 résume les différentes étapes pour chacun des groupes. Nous pouvons diviser l'étude en trois différentes parties : le cours magistral, les séances pratiques de ce cours et une session d'évaluation.

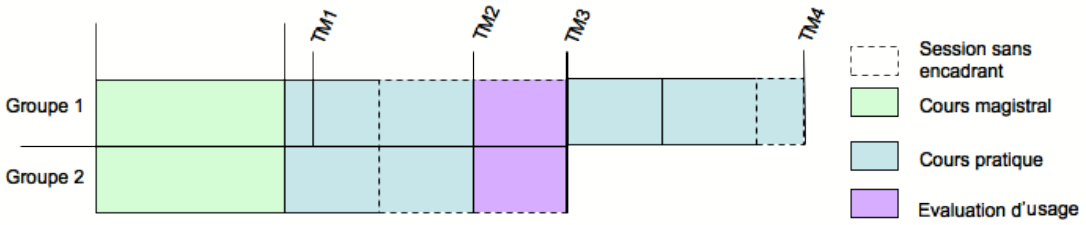


Figure 3.1.11 : Les étapes et les modèles de tâches de l'évaluation

Le cours magistral

Le cours magistral est basé sur la conception centrée-utilisateur, la modélisation des tâches en faisant partie. Le cours sur la modélisation des tâches ne vise pas à l'enseignement de plusieurs modèles de tâches, mais est uniquement focalisé sur le formalisme K-MAD. Ce formalisme est expliqué en détail pendant le cours (qui dure environ 4 heures) et les étudiants pratiquent la modélisation des tâches en utilisant K-MADe pendant les séances pratiques. De ce fait, même s'ils ne sont pas des experts en modélisation des tâches, ils sont plus entraînés sur l'utilisation des notations de ce modèle de tâches que les experts en ergonomie (d'après l'étude présentés dans [Couix 2007]).

La seconde partie du cours magistral concerne l'évaluation des interfaces (les concepts basiques de l'évaluation et des principales méthodes utilisées [Nielsen 1993]). Comme les étudiants jouent le rôle d'observateurs (pendant la session d'évaluation de la fin de l'étude), le protocole utilisé pour cette étude est utilisé comme exemple afin de faciliter leur futur travail d'évaluation. Cependant, comme cette étude est leur première évaluation pratique, leur participation est limitée à l'observation et à la prise en note de ces observations. Les notes des observateurs sont complétées par d'autres données.

Les séances pratiques

Pendant la première séance pratique, les étudiants ont eu à concevoir deux modèles de tâches différents. Le premier représente leur activité avant leur arrivée à l'université chaque matin (TM1). Peu de concepts sont utilisés pour modéliser cette activité. Ils l'ont décomposée sans définir ni objets ni conditions.

Dès la conception de leur second modèle de tâches (TM2), tous les concepts de K-MADe sont utilisés. Ce second modèle de tâches décrit l'activité d'une secrétaire d'agence de location de véhicule. Pour concevoir ce modèle de tâche, les étudiants sont placés en binôme et n'ont aucune limite de temps, une fois la séance pratique terminée (nous avons estimé ce temps à environ deux heures).

Les sessions pratiques

Pendant les dernières sessions, dites d'usage (sessions réalisées une fois que le processus d'apprentissage des étudiants est considéré comme terminé), les étudiants ont eu à modéliser l'activité d'édition d'une feuille de marquage de volley-ball (TM3). Les instructions de cette activité ont été données au début de la session. Ces instructions sont composées des instructions officielles de la Fédération Française de Volley-ball (FFVB) et de deux exemples de feuilles de marquage (une complétée et une vierge).

Enfin, pour concevoir le dernier modèle de tâche (TM4), les étudiants peuvent poser des questions aux enseignants. À partir de ce modèle de tâches, ils doivent concevoir une application pour un laboratoire d'analyse animale. Seul le premier groupe d'étudiants (les étudiants en bioinformatique) ont travaillé sur ce dernier modèle de tâche (TM4).

3.1.3.3. La méthode d'évaluation

Afin de réaliser cette étude, nous utilisons une technique d'évaluation classique [Nielsen 1993] : l'observation en temps réel de sujets en train d'utiliser l'outil. Dans le but d'évaluer la progression de l'usage des concepts formels dans l'apprentissage de la modélisation des tâches, une observation à deux niveaux a été menée. Une observation du comportement de l'ensemble du groupe (*observation globale*) et une observation de comportement de chacun des étudiants (*observation individuelle*). L'observation individuelle de tous les étudiants (du premier et du second groupe) permet l'évaluation de l'usage des composants formels de K-MADe.

L'observation globale

Après chacune des sessions du premier groupe, les enseignants notent ce qu'ils ont observé. Les difficultés globales de compréhension (de formulation ou de concepts) sont précisément notées. Les enseignants notent également les questions des étudiants. Ces notes ont pour but d'aider à l'interprétation des autres données collectées (comme les modèles de tâches collectés).

L'observation individuelle

Cette forme d'évaluation a été utilisée lors de la session d'évaluation. Tous les étudiants forment des binômes. Pendant la première moitié de la session, un étudiant joue le rôle de l'expert en évaluation (nommé *observateur*) et le second, le rôle du

concepteur de modèle de tâches (en utilisant K-MADe, nommé *utilisateur*). Leurs rôles sont échangés pour la seconde partie de la session. Chaque session dure une heure et demie, avec par quinze minutes de pause entre chaque. L'activité à modéliser est la même pour tous les étudiants et sa description est donnée en français au début des sessions. L'utilisateur doit modéliser l'activité d'édition des feuilles de marque de Volley-ball. K-MADe est utilisé pour modéliser les tâches.

Les notes des observateurs. Lors de la modélisation, les observateurs s'assurent que les utilisateurs décrivent oralement le processus de modélisation qu'ils suivent, et notent ce qu'ils observent concernant l'utilisation de l'outil par l'utilisateur (hésitations, exploration dans différentes parties de l'application sans actions…). Afin d'aider les observateurs dans leur évaluation, nous leur avons fourni des feuilles d'observation (illustrées sur le Tableau 3-9). Ces feuilles sont principalement composées d'un tableau de trois colonnes qui correspondent aux trois types d'information détaillant chacune des observations :

- Le type d'observation parmi l'ensemble des catégories définies (but de l'utilisateur (B), fonctionnalités de l'outil (F), utilisation des fonctions (U) et information (I)).
- L'observation sous forme textuelle (dans les propres mots de l'observateurs).
- Une donnée temporelle sur l'observation.

Type	Observation	Indication temporelle
U	La fenêtre principale n'est plus accessible (« simulation » est noté mais la fenêtre de simulation n'est pas accessible non plus). => K-MADe est relancé	14h32
B	recherche comment définir les objets	14h34 14h37
F	l'utilisateur ne comprend pas la signification du bouton avec un aimant	14h40

Tableau 3-9 : Un exemple de feuille d'évaluation

Comme cette étude est la première évaluation pratique des étudiants, leur participation est limitée à l'observation et à la prise en note de ces observations. Afin de compléter les notes des observateurs et les modèles de tâches conçus, nous avons utilisé deux autres types de données : les user-logs et les questionnaires.

Les User-logs. Pour compléter les observations faites lors de l'évaluation, les utilisateurs de la première évaluation utilisent une version de K-MADe qui produit des user-logs. Cette version a permis de tracer les actions des utilisateurs en utilisant des et produit un fichier texte (l'*user-log*). En particulier, ce fichier indique quand l'utilisateur entre et sort de chacun des espaces de K-MADe (espace de tâches, espace des objets abstraits…). La Figure 3.1.12 présente un exemple de ce fichier.

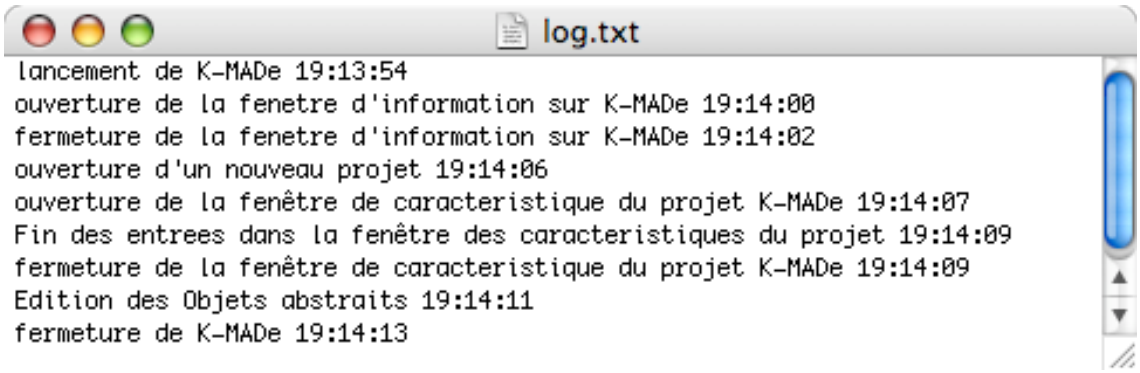


Figure 3.1.12 : Un exemple de user-log produit par K-MADe

Les questionnaires. À la fin de la session d'évaluation, les participants du second groupe ont rempli un questionnaire en français sur leur utilisation des objets. Ce questionnaire est composé de cinq questions sur la définition des objets, leurs suppressions et les conditions.

3.1.3.4. Les données

Comme chacune des sessions essaie d'atteindre différents buts, nous n'avons pas collecté les mêmes données pour chacune des phases d'évaluation. Dans cette section, nous présentons les données collectées de chacune des études. Le Tableau 3-10 résume les données collectées pour chacune des sessions et des buts d'évaluation. Certaines données n'ont pas pues être incluses lors de l'analyse (Tableau 3-11), nous justifierons notre sélection dans le seconde section.

Les données collectées

Le but de la conception du premier modèle de tâches (TM1) est d'introduire l'utilisation de K-MADe et de ce fait, nous n'avons collecté aucune donnée spécifique lors de cette session. Dès la conception du second modèle de tâche (TM2), les étudiants utilisent tous les concepts de K-MAD. Nous avons collecté les modèles de tâches réalisés lors de ces sessions pour analyser les concepts formels de K-MAD.

Le troisième modèle de tâche (TM3) vise à atteindre deux buts différents ; (1) la connaissance du processus de modélisation des tâches ; (2) la connaissance de l'usage des entités formelles et les difficultés de cet usage. Enfin, afin d'évaluer l'utilisation des entités formelles après l'apprentissage de la modélisation des tâches, les étudiants ont conçu leur dernier modèle de tâches (TM4).

La session d'évaluation du premier groupe (la conception de TM3 par le groupe 1) vise à analyser le processus de modélisation des tâches, particulièrement quand les entités de K-MADe sont définies et quand elles sont manipulées. Afin d'obtenir ces informations, nous avons utilisé les *user-logs* et les notes des observateurs.

Ces deux types d'information permettent d'avoir deux données complémentaires. Alors que les observateurs se concentrent sur l'usage des utilisateurs, quels sont leurs buts et comment ils conceptualisent leurs modèles, les *user-logs* donnent des informations sur comment les composants de K-MADe sont utilisés. En utilisant les repères temporels de chacune de ces données, nous pouvons les associer et déterminer comment les utilisateurs utilisent les composants de l'outil K-MADe.

De plus, nous avons demandé à chaque étudiant d'exploiter leurs notes et les *user-logs* pour écrire un rapport d'évaluation. Ce rapport inclut le processus de modélisation de l'utilisateur observé, son utilisation et son usage de l'outil, et une analyse du modèle obtenu. Même si les documents produits sont lisibles et organisés, les modèles, les notes des observateurs et les *user-logs* sont également collectés pour une analyse experte.

La session d'évaluation du premier groupe aide à déterminer quand les entités de K-MADe sont utilisées dans le processus global de modélisation. Nous n'avons collecté aucune information précise sur leur usage. La session du second groupe a pour but de répondre à cette question.

Comme pour le premier groupe, le comportement des utilisateurs des participants du second groupe est inclus dans les notes des observateurs et un document est écrit pour rapporter leurs observations. Cependant, les *user-logs* ne fournissent pas d'information sur les usages des entités donc, nous ne les avons pas utilisés pour cette session. Afin d'analyser l'utilisation des entités de K-MADe, nous avons considéré deux types de données : les modèles et les questionnaires. La vérification des entités dans les modèles conçus indique le degré de compréhension des concepts des objets. L'analyse des questionnaires (associés aux rapports d'analyse des étudiants) vise à nous informer sur les difficultés et les besoins dans les objets utilisés.

session	*Modèles de tâches*	*Groupe 1*	*Groupe 2*
session 1	TM1	- notes des enseignants	- notes des enseignants
	TM2	- notes des enseignants - modèles de tâches	
session 2	TM3	- user-logs - notes des observateurs - documents d'exploitation des étudiants - modèles de tâches - notes des enseignants	- notes des observateurs - documents d'exploitation des étudiants - modèles de tâches - notes des enseignants
session 3	TM4	- notes des enseignants	-

Tableau 3-10 : Les données collectées

La sélection des données

Comme les étudiants sont répartis en groupes pour réaliser le second modèle de tâches (TM2), nous avons obtenu onze modèles produits par le premier groupe. Cependant, deux modèles de tâches ne nous ont pas été retournés. De plus, un des modèles n'était pas terminé à cause d'un virus informatique c'est pourquoi nous ne l'avons pas pris en compte. Huit modèles de tâches TM2 ont donc été analysés.

Pendant les secondes sessions, nous avons collecté un dossier par utilisateur. Chaque dossier inclut les notes des observateurs, le document d'exploitation, le modèle de tâches, l'*user-log* (pour le premier groupe) et les questionnaires (pour le groupe 2). La première session d'évaluation vise à obtenir des connaissances sur le processus de modélisation. Les données utilisées pour cela sont principalement les *user-logs*. Ces fichiers ont été automatiques générés sans difficulté technique ni incident.

Cependant, nous n'avons pas voulu utiliser ces *user-log* sans prendre en compte leur contexte (illustré par les notes des observateurs et les documents d'exploitation). Deux de ces dossiers n'étant pas complets, nous ne les avons pas inclus dans le processus d'analyse. Nous avons donc considéré 46 des 48 dossiers (correspondant aux 48 étudiants du groupe 1) pour notre analyse.

Les données utilisées pour l'évaluation menée sur le second groupe de participants incluent toutes les informations contenues dans les dossiers, c'est pourquoi nous ne pouvons considérer que les dossiers complets. Dans la première analyse, les notes des observateurs, les documents d'exploitation des étudiants et les modèles de tâches étaient seulement utilisés pour nous aider en nous donnant le contexte de conception des *user-log*s mais pour la seconde, ces données sont essentielles. Pendant cette évaluation, nous avons observé que seul l'étudiant non francophone n'avait pas compris toutes les instructions (cette observation fût confirmée par le questionnaire qu'il nous a rendu). Nous n'avons donc pas considéré son dossier pour notre analyse. La seconde partie de notre évaluation est donc basée sur 19 dossiers complets.

	collecté	utilisé
TM2	11	8
dossiers TM3 du premier groupe	48	46
dossiers TM3 du second groupe	20	19

Tableau 3-11 : La sélection des données

3.1.3.5. Apprentissage de l'utilisation des aspects formels

Les données collectées pendant toutes les sessions d'entraînement nous permettent d'étudier l'usage des aspects formels dans les processus d'apprentissage : l'utilisation des opérateurs de décomposition, des entités (les objets, les événements et les utilisateurs) et des conditions.

Les **opérateurs de décomposition** organisent temporellement les sous-tâches. L'utilisation de l'outil de vérification grammaticale met en lumière les difficultés de compréhension de ces concepts par les utilisateurs. Les étudiants ont parfois mélangé la notion de décomposition et d'héritage et donc, ont défini des tâches composées d'une seule sous-tâche. Une fois détectée, lors de la première session, cette mauvaise compréhension a été expliquée et corrigée par les enseignants.

Pendant, cette première session, en concevant leur second modèle de tâches (TM2) les étudiants ont défini des objets abstraits, des objets concrets, des événements, des utilisateurs et ont eu à les utiliser pour définir des pré-conditions, des post-conditions (au sens K-MADe) et des conditions d'itération. Tous ces concepts ont été présentés pendant le cours magistral et les enseignants ont illustré l'édition d'une pré-condition. Ils ont également corrigé les erreurs des étudiants. C'est pourquoi nous n'avons pas évalué si les étudiants ont correctement utilisé les entités mais seulement s'ils les ont utilisées. La Figure 3.1.13 présente les pourcentages d'étudiants ayant défini chacune des entités pour concevoir leur second modèle de tâches (TM2). Comme le montre cette figure, le définition des **objets abstraits** et **concrets** et leur utilisation pour définir les **post-conditions** semblent être bien comprises. La définition et l'utilisation des objets requiert la manipulation de texte (pour les noms), les types d'ensemble (pour les groupes d'objets) et les types basiques en informatique (pour définir les attributs). Tous les étudiants ont une formation en informatique : l'utilisation de ces types ne conduit donc à aucune question. Alors que la nécessité de définir des entités est comprise par les étudiants, leurs rôles et leurs liens semblent être moins bien assimilés. Les objets concrets sont utilisés pour définir les conditions, cependant, 12,5% des étudiants qui ont défini des objets abstraits ne les ont pas instanciés et 12,5% n'ont pas utilisé les objets pour exprimer des conditions. Donc, 25% des étudiants ont défini les objets abstraits sans les lier à aucun autre concept.

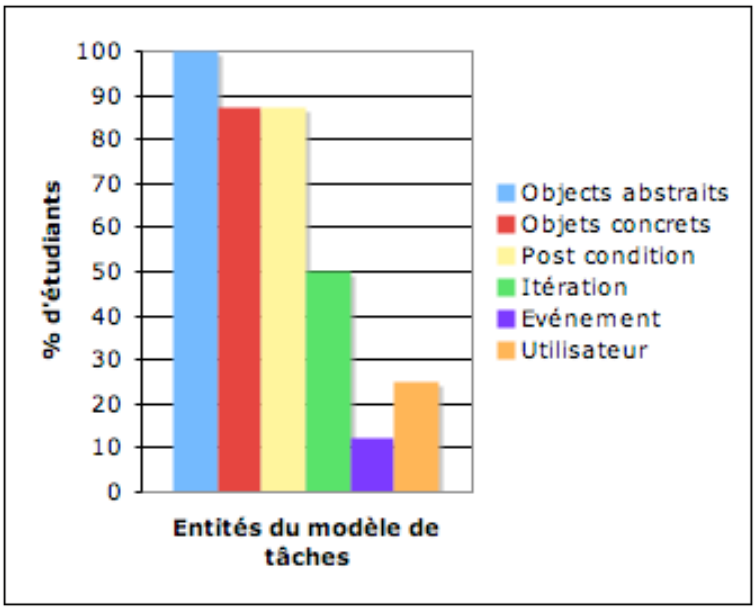


Figure 3.1.13 : Le pourcentage des étudiants ayant défini des entités formelles pendant la première séance pratique

À propos des **conditions** formelles, 87,5% des étudiants ont défini au moins une post-condition dès leur première session. Nous n'avons pas pris en compte la définition des pré-conditions dans cette étude cas les enseignants les ont utilisé pour illustrer l'utilisation des calculettes de K-MADe (voir la Figure 3.1.14). Lorsque les étudiants n'ont pas réussi à définir formellement leur conditions en utilisant les calculettes de K-MADe, ils l'ont fait textuellement (Figure 3.1.15). De ce fait, même si la définition des entités formelles et les conditions de K-MADe sont causes de difficultés d'utilisation, la nécessité de leur utilisation dans le processus de modélisation des tâches est intuitive pour les étudiants pour compléter l'organisation temporelle des tâches [Caffiau, et al. 2008a].

Figure 3.1.14 : Une calculette de K-MADe

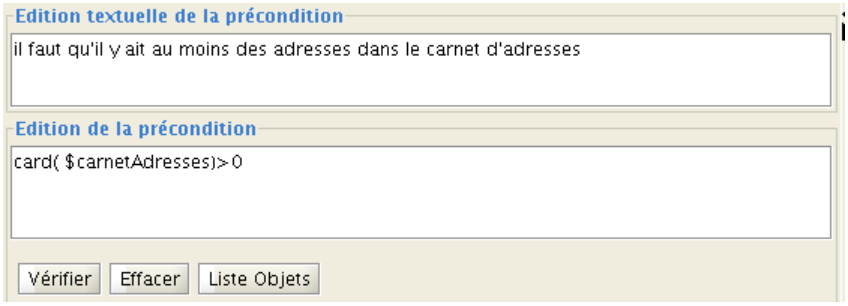


Figure 3.1.15 : Edition d'une condition pour l'entrée d'une adresse email à partir du carnet d'adresse

La Figure 3.1.13 montre que peu d'étudiants ont utilisé les **conditions d'itération**, les **événements** et les **utilisateurs**. Les itérations sont définies de la même manière que les pré et les post-conditions (en utilisant les calculettes), sans plus de difficultés techniques. Les événements et les utilisateurs sont définis par du texte. Le manque d'utilisation de ces concepts peut être dû à l'enseignement. Alors que les objets, les pré- et les post-conditions ont été présentés (et illustrés) par les enseignants,

l'itération, les utilisateurs et les événements ont été uniquement présentés dans le cours magistral (sans aucun exemple).

3.1.3.6. Processus de modélisation

Une fois que les étudiants ont découvert l'outil K-MADe et le formalisme K-MAD, nous avons étudié les processus de modélisation des tâches qu'ils ont suivis pendant la session d'évaluation et spécialement l'intervention des entités formelles dans ces processus. Avant d'identifier l'intervention des objets dans le processus de modélisation des tâches, nous avons observé que certains étudiants n'ont pas défini d'objet. De plus, 26% des utilisateurs (12/46) d'un premier groupe dans la seconde session n'ont pas essayé de définir (ou d'utiliser) d'entité formelle de K-MADe. Cependant, nous ne pouvons pas précisément identifier pourquoi. Deux raisons peuvent expliquer l'absence de ces éléments dans le processus de modélisation de tâches : la durée limitée de l'étude, ou la non-assimilation des concepts des objets. Les notes et rapports des étudiants ne nous ont pas permis d'identifier la raison principale. Six participants ont indiqué que la session n'était pas assez longue mais les autres (6/12) n'ont donné aucune information à ce sujet.

À partir des données contenues dans les 34 dossiers sélectionnés, nous avons identifié trois schémas principaux suivis par les utilisateurs pour concevoir les modèles de tâches. Le Tableau 3-12 représente la répartition des étudiants pour chacun des processus. Les plus utilisés (44,10% des schémas intégrant les entités formelles) sont divisés en deux étapes. Premièrement, les utilisateurs composent l'arbre de tâches (ils décomposent les tâches). Puis, ils définissent itérativement les entités et les associent aux tâches. Les étapes dans le second processus le plus utilisé (29,40%) sont séquentielles. Les utilisateurs réalisent la décomposition des tâches avant de définir toutes les entités et de les associer aux tâches (en éditant les conditions). De plus, un processus incomplet (suivi par 22% des étudiants) est composé des deux premières étapes du second schéma. Le dernier schéma est l'itération du second. La Figure 3.1.16 présente le premier et le second des schémas.

N'ayant défini aucune entité formelle	Schéma 1	Schéma 2	Schéma 3
12	15	10	9

Tableau 3-12 : Répartition des étudiants par processus de modélisation

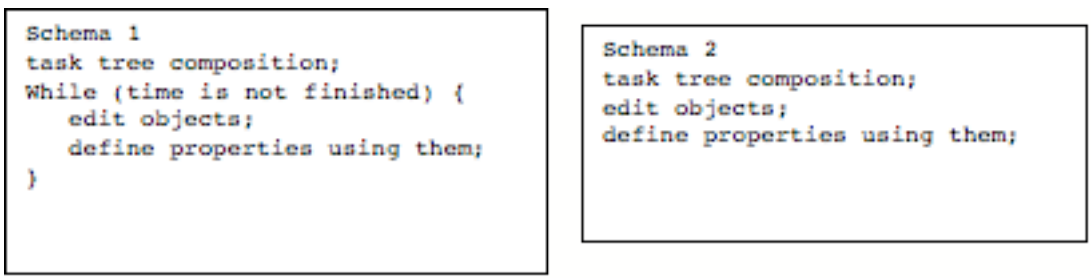


Figure 3.1.16 : Les schémas de processus de modélisation de tâches

Ces observations nous donnent quelques idées sur la place qu'occupent les objets dans le processus de modélisation des tâches. Par exemple, la définition des objets en parallèle avec la composition de l'arbre de tâches indique que les utilisateurs associent les concepts d'objets aux tâches. Au contraire, lorsque la définition et l'utilisation des objets sont séparées de la composition de l'arbre de tâches, nous pouvons conclure que les utilisateurs définissent les objets uniquement pour pouvoir les utiliser dans les expressions des conditions.

3.1.3.7. Définition des entités formelles (après apprentissage)

Le second point étudié est la définition des entités formelles pour modéliser les tâches. Nous avons cherché principalement à identifier quelles sont les entités formelles qui sont utilisées et pourquoi.

Quelles sont les entités utilisées ?

À partir des données collectées dans les deux sessions d'évaluation (du groupe 1 et 2), le Tableau 3-13 montre comment certains étudiants définissent chaque composant du modèle de tâches. En fonction de ces résultats, peu d'utilisateurs ont intégré les concepts d'**événement** (20% dans le premier groupe et 26% du second) et les **utilisateurs** (20% et 10,5%) dans le processus de modélisation de tâches. À l'inverse, la plus grande partie du premier groupe d'utilisateurs et tous les étudiants du second groupe ont défini les objets (les objets abstraits et les objets concrets) pour modéliser les activités.

	Événement	Utilisateur	Objet abstrait	Objet concret	Groupe	Pré	Post	Itération
Groupe 1	20%	20%	74%	60%	74%	40%	40%	20%
Groupe 2	26%	10.5%	100%	100%	100%	84%	89.5%	84%

Tableau 3-13 : Les utilisateurs ayant défini chacun des éléments de K-MADe

La définition des événements et des utilisateurs est textuelle. Leur association aux tâches est facilement réalisée en faisant une sélection parmi les éléments définis. Au contraire, les objets abstraits et concrets sont composés de concepts différents. Le niveau de difficulté de définition des différents concepts (composés et non-composés) ne peut donc pas expliquer la différence d'utilisation.

Pourquoi les entités formelles sont-elles utilisées ?

Cependant, le Tableau 3-13 indique également la proportion d'utilisateurs ayant utilisé des pré-conditions, des post-conditions et des conditions d'itération. Lors de la

session d'évaluation, ces trois types d'expression de manipulation d'objets ont été largement utilisés dans les processus de modélisation (pour le second groupe, 84% ont défini au moins une pré-condition, 89,5% ont défini au moins une post-condition et 84% ont défini au moins une condition d'itération). Avant d'éditer ces conditions, les utilisateurs doivent définir des objets (objets abstraits, objets concrets et groupes). Donc, les objets peuvent être définis uniquement pour permettre la définition des conditions. Alors, les utilisateurs ne voient pas les objets comme une part des tâches mais comme un moyen de définir des conditions.

À partir de ces résultats, nous observons que pour la plus grande partie des étudiants, il est naturel de définir des objets dans le but de compléter la sémantique des opérateurs temporels (opérateurs de décomposition des tâches). Par exemple, lors de leur conception du modèle de tâches d'édition de feuilles de match de Volley-ball, tous les étudiants ont naturellement défini des conditions en utilisant des objets, comme par exemple pour exprimer la fin d'un match.

L'étude de la répartition de la définition des concepts de K-MADe dans les deux sessions montre une différence entre les deux groupes. Dans le premier groupe d'évaluation, 26% des utilisateurs n'ont utilisé aucune entité. Comme tous les participants ont suivi le même cours magistral, la disparité ne peut être totalement expliquée par l'enseignement. Cependant, ces groupes n'ont pas le même *background*. Les chiffres donnés dans le Tableau 3-13 indiquent clairement que la définition des objets formels est plus facile à réaliser pour des étudiants en informatique pure.

3.1.3.8. L'utilisation des aspects formels pour valider les modèles de tâches

Les aspects formels dans la modélisation des tâches permettent de réaliser des vérifications des modèles de tâches pour détecter les incohérences, par exemple, le fait qu'une tâche soit composée d'une seule sous-tâche (Figure 3.1.8). K-MADe offre des outils pour permettre ce type de vérification avant de permettre le déclenchement de l'outil de simulation. Les dernières observations de cette étude concernent ces vérifications dans les modèles de tâches conçus. Nous présentons pourquoi et quand les étudiants utilisent les outils de correction de K-MADe dans les processus de modélisation des tâches.

Pourquoi les outils de correction sont-ils utilisés ?

Pendant l'entraînement à la modélisation des tâches, ces outils peuvent être utilisés dans le but de vérifier que le modèle est conforme au formalisme.

Lorsque cette étape est terminée (dès le TM3), 62,5% des étudiants continuent à utiliser au moins un des outils de correction, et 12,5% ont indiqué qu'ils ne les ont pas utilisés par manque de temps. L'outil de vérification de cohérence détecte au moins une erreur dans 90% des cas indiquant que les utilisateurs ont besoin de ce type d'outil pour concevoir leurs modèles. Cependant, cet outil de vérification de cohérence est l'outil le

plus utilisé (87,5% des étudiants arrêtent l'outil de simulation dès que plus aucune erreur de cohérence n'est détectée).

Quand les outils de correction sont-ils utilisés ?

L'étude des user-logs montre que les utilisateurs déclenchent des outils de correction, spécialement une fois qu'ils ont décomposé leurs tâches (entre les lignes 1 et 2 des schémas de la Figure 3.1.16) et lorsqu'ils considèrent la modélisation comme étant terminée. Chacune de ces utilisations correspond à la détection d'au moins une erreur par l'outil de vérification de cohérence.

De plus, les conditions formelles ont systématiquement été vérifiées (parfois plusieurs fois) dès qu'elles ont été écrites afin de vérifier leur syntaxe.

3.1.3.9. Bilan de l'utilisation du formalisme

La comparaison théorique précédente des différents modèles de tâches avait identifié que l'outil K-MADe était celui qui permet la définition formelle la plus complète. L'étude que nous venons de décrire dans cette section (3.1.3) avait pour but de déterminer si les aspects formels implémentés dans K-MADe ne sont pas une entrave à l'apprentissage et à l'utilisation de K-MAD.

Les résultats obtenus portent donc sur deux points essentiels : la définition des entités formelles et leur utilisation. Les entités formelles de K-MADe sont le plus souvent définies après la décomposition des tâches, pour **ajouter des informations complémentaires**. Nous avons observé que les entités formelles sont définies afin d'ajouter de la sémantique aux tâches K-MADe. **La définition des tâches n'est pas disjointe de la définition des entités qui sont manipulées pour réaliser les tâches**. De plus, la définition des tâches sans aucun objet limite la sémantique au nom de la tâche, la rendant donc dépendante de la lecture qui est faite du modèle de tâches. Au contraire, l'association des objets aux tâches (utilisés dans les conditions) décrit formellement le comportement des tâches (conditions d'exécution et effets). L'ajout des objets aux tâches facilite la compréhension et donc, la communication entre tous les acteurs de la conception de l'application (un des six points d'évaluation dans [Balbo, et al. 2004]).

En plus de ce but sémantique, les objets (et donc, les conditions) sont aussi utilisés pour compléter les opérateurs temporels. Cette contribution apparaît rapidement aux concepteurs des modèles de tâches pendant notre étude (tous les étudiants utilisent naturellement les entités formelles pour définir la fin d'un jeu de volley-ball). Donc, l'analyse des évaluations présentées montre que, pour les concepteurs, les **fonctionnalités de K-MADe sont nécessaires pour concevoir les modèles de tâches**.

Finalement, le but principal de l'utilisation des entités formelles est la capacité de les évaluer afin de **réaliser certaines validations**. Comme notre évaluation

empirique le montre, les entités les plus utilisées sont celles qui peuvent être évaluées et utilisées dans les outils de vérification de K-MADe.

K-MADe offre deux outils pour la conception des modèles de tâches afin d'aider à la conception des applications interactives ou à leur validation : l'outil de vérification grammaticale et l'outil de simulation. Pendant notre étude d'évaluation, l'outil de vérification grammaticale a été largement utilisé (seul un des étudiants a réalisé un scénario en utilisant l'outil de simulation). L'outil de vérification grammaticale vérifie que les modèles conçus sont cohérents avec le formalisme. Cette fonction joue deux rôles. D'abord, pendant l'apprentissage, elle **aide les étudiants à appliquer le formalisme** de modèle de tâches. Deuxièmement, elle permet de **détecter rapidement et automatiquement les erreurs syntaxiques** du concepteur (l'outil de vérification grammaticale est généralement déclenché à la fin des étapes de modélisation).

Ces résultats prouvent, d'une part, la nécessité de la présence d'objets dans la modélisation (appuyant les recherches théoriques menées sur le domaine) et d'autre part, que bien que l'utilisation des entités formelles dans K-MADe nécessite un temps d'apprentissage, elle est très appréciée lors de la conception des modèles. Cependant, il est à noter que nos résultats ne sont significatifs que pour les utilisateurs correspondants à nos types de participants. Il serait intéressant de compléter notre étude par la même évaluation (le même protocole d'évaluation) mais avec des participants ayant des compétences différentes (comme des étudiants en ergonomie par exemple).

3.1.4. Bilan du choix du modèle de tâches

De nombreux modèles de tâches ont été définis afin d'exprimer les activités humaines (assistées ou non de systèmes informatisés). Les concepteurs d'IHM peuvent utiliser ces modèles comme support à la conception [Balbo, et al. 2004] (communication entre les différents intervenants, validation…).

Malheureusement, parmi l'ensemble des études sur la modélisation des tâches, peu ont abouti à la réalisation d'outils permettant la conception de modèles de tâches. Nous avons trouvé six environnements support à la conception de modèles de tâches : MAD* [Gamboa et Scapin 1997, Scapin et Bastien 2001] (IMAD), CTT [Paternò, et al. 1997] (CTTE), Diane+ [Tarby et Barthet 2001] (TAMOT), GTA [van der Veer 1996] (EUTERPE), K-MAD (K-MADe) et un environnement d'analyse de tâches, AMBOSS [AMBOSS].

Malgré ces origines différentes, ils proposent l'utilisation de concepts communs (les tâches, les acteurs, …) bien que ces concepts n'aient pas nécessairement le même nom. Cependant, des différences notables sont présentes sur la décomposition des tâches ou la prise en compte de l'état du monde.

De plus, l'utilisation de ces modèles via leur outil nous a fait prendre conscience de la distance existant entre certains des modèles définis et leur implémentation dans les outils. Par exemple, Diane+ prévoit dans son modèle la définition d'objets alors que

l'outil ne permet pas leur utilisation. Cette distance entre les concepts qui peuvent être définis via l'utilisation des outils et ceux définis par les études ayant été menée à la définition des modèles rend la modélisation par les outils incomplète.

Parmi ces études, van Welie dans [van Welie, et al. 1998] a montré la nécessité des objets dans les modèles de tâches. Cette nécessité théorique a été confirmée ensuite par d'autres travaux comme [Dix 2008].

D'autre part, l'un des bénéfices apportés par l'utilisation d'un éditeur pour modéliser les tâches est de permettre la réutilisation des modèles produits par d'autres outils (comme dygimes framework [Luyten 2004] ou TERESA [Mori, et al. 2004] le font à partir d'un modèle CTT) ou par des fonctionnalités de l'outil de modélisation (comme des outils de simulation proposés [Caffiau, et al. 2008b]). Pour que les entités éditées dans les modèles soient utilisables, il est nécessaire qu'elles soient formellement définies. Parmi les outils de modélisation que nous avons étudiés, K-MADe est celui qui permet la définition formelle la plus complète (certaines caractéristiques des tâches, objets, conditions, itération…). K-MAD est le seul modèle intégrant les objets et dont l'outil (K-MADe) permet leur définition et leur utilisation formelle. Pour cela, il est un bon candidat pour servir de formalisme de modèles de tâches dans la suite de nos travaux.

Cependant, avant d'arrêter notre choix sur ce formalisme, nous avons souhaité évaluer l'implication des ajouts formels dans le processus de modélisation. L'étude proposée confirme que les objets sont, pour les concepteurs, un composant des modèles de tâches, tant comme élément descriptif de l'activité (pour compléter la sémantique des opérateurs temporels par exemple) que comme élément dynamique du modèle (après la simulation, les étudiants vont volontiers vérifier que les valeurs de leurs objets ont été modifiées par la simulation de leur modèle). De plus, cette évaluation montre qu'un temps d'apprentissage de l'utilisation de ces concepts est nécessaire pour les appréhender mais qu'une fois cette étape passée, ils sont intégrés au processus de modélisation (la plupart des étudiants (74% du groupe 1 et 100% du groupe 2) intègrent sans incitation (pour TM3) les concepts formels).

L'intégration de définitions formelles, notamment des objets, n'est donc pas une entrave au processus de modélisation, et si elle nécessite un temps d'apprentissage celui-ci ne semble pas très important au regard des bénéfices qui peuvent en être tirés pour l'utilisation des modèles dans le processus de conception des IHM. En conséquence, nous avons donc choisi d'utiliser des modèles K-MAD (édités avec l'outil K-MADe) pour exprimer les activités humaines supports à la conception d'IHM pour la suite de nos travaux.

3.2. Choix du modèle de dialogue

L'utilisation des outils de simulation proposés par les outils de modélisation de tâches (CTTE, K-MADe et AMBOSS) permet d'illustrer le lien intuitif qui existe entre le dialogue (i.e : la partie dynamique de l'application) et la dynamique des modèles de tâches (exprimée via les opérateurs temporels, les conditions… et exploitée par les outils de simulation permettant ainsi la définition de scénario d'activité). Comme nous

l'avons indiqué dans 2.2.3.3, le dialogue de l'application peut être exprimé sous forme de modèle (nommé modèle de dialogue).

Dans la présente section, nous présentons l'étude menée pour choisir le formalisme support des modèles de dialogue.

Nos travaux portant sur le domaine de la conception des applications interactives, nous avons focalisé notre étude comparative sur les modèles de dialogue pouvant être *directement* transcrits dans l'implémentation des systèmes.

Les seules études ayant poussé jusqu'à l'implémentation de la logique de modélisation du dialogue se situent dans le domaine des formalismes à états. Quatre d'entre eux sont particulièrement remarquables à ce sujet. Il s'agit des « machines à états » [Appert et Beaudouin-Lafon 2006, Jacob 1982], les Objets Coopératifs Interactifs (ICO) [Palanque et Bastide 1994], les Interacteurs Hiérarchisés [Girard, et al. 1995] et les Machines à États Hiérarchiques (HSM) [Blanch et Beaudouin-Lafon 2006].

Nous allons présenter ces quatre formalismes dans la section suivante (§ 3.2.1) avant d'en faire une comparaison (§ 3.2.2) explicitant notre choix de modèle de dialogue.

3.2.1. Présentation des modèles étudiés

Nous présentons, dans cette section les quatre formalismes de modèle de dialogue ayant été intégrés à l'implémentation d'applications interactives : les machines à états, les objets coopératifs interactifs, les interacteurs hiérarchisés et les machines à états hiérarchiques. Pour chacun d'eux, nous préciserons quels en sont les composants et dans quelle mesure l'intégration peut être faite.

3.2.1.1. Les machines à états

Une machine à états est le formalisme à base d'états qui a, le premier [Jacob 1982], servi à représenter le dialogue d'une application. Depuis, différents modèles à base d'états ont été développés pour exprimer le dialogue des applications interactives.

Les machines à états appelées aussi automates à états ou systèmes de transition étiquetés (STE) sont exprimables sous forme graphique. De plus, elles sont basées sur des modèles mathématiques ; par conséquent, elles permettent le raisonnement et donc, la validation et la vérification de nombreuses propriétés du dialogue dans les IHM telles que les propriétés de sûreté ou de vivacité.

Une machine à états permet de représenter le dialogue d'une application par un ensemble de quatre éléments :

- un ensemble d'états
- un état initial
- un ensemble de transitions
- un ensemble d'étiquettes associée à ces transitions

Dans le cas de l'IHM, les transitions permettant de passer d'un état à un autre sont activées par des événements tels qu'un mouvement de la souris ou la fin d'une horloge. À chaque transition peuvent être associées des gardes et des actions.

Une garde est une condition de réalisation d'une transition. Elle doit être évaluée à « vrai » pour que la transition soit exécutée.

L'action d'une transition est le code associé à cette transition.

À chaque état, deux actions peuvent être associées, la première nommée *action d'entrée,* est réalisée lorsque l'état devient l'état courant, et la seconde *action de sortie*, est exécutée lorsque l'état cesse d'être l'état courant.

Le fonctionnement d'une machine à état se déroule de la manière suivante. Lorsqu'un événement se produit, s'il existe une transition correspondant au traitement de cet événement et partant de l'état courant avec sa garde à vrai, alors les actions : *de sortie* de l'état courant, *de la transition, d'entrée* du nouvel état courant sont exécutées. Sinon, l'événement est ignoré.

L'état courant au début de l'exécution de la machine à état est l'*état initial*.

De plus, un automate peut être représenté graphiquement, chaque état étant représenté par une ellipse et chaque transition par une flèche. L'état initial est, quant à lui, spécifié par une petite flèche sur le côté gauche de l'ellipse. La Figure 3.2.1 est la représentation graphique de la machine à état exprimant le dialogue de l'envoi d'un email sauvegardé dans les brouillons.

Au début de l'exécution de la machine à états, l'état courant est donc l'état nommé *init* (état spécifié par la flèche à gauche). Lorsqu'un email en brouillon est sélectionné, les gardes des transitions déclenchables par cet événement à partir de l'état initial sont évaluées (ici les gardes de *selectBrouillonSans* et *selectBrouillonAvec*). Si l'email sélectionné a une adresse de destinataire (valide ou non) alors la garde de la transition *selectBrouillonAvec* est évaluée à vrai et l'état courant devient *avecDest* sinon, c'est la transition *selectBrouillonSans* qui est exécutée et l'état courant devient *sansDest*. À partir de celui-ci deux transitions sont exécutables *modifMessage* (qui ne change pas l'état courant) et *mettreDest* qui met le système dans l'état *avecDest*. De cet état, quatre transitions sont déclenchables : *modifierMessage* et *modifierDest* (qui ne modifient pas l'état courant) et *supprimerDest* et *envoyer*. La transition *supprimerDest* est déclenchée lorsque l'adresse du destinataire est supprimée, l'état courant redevient alors *sansDest*. Enfin, *envoyer* est déclenchée lorsque l'envoi de l'email est déclenché. Le système revient alors dans son état initial (avant qu'un email n'ait été sélectionné).

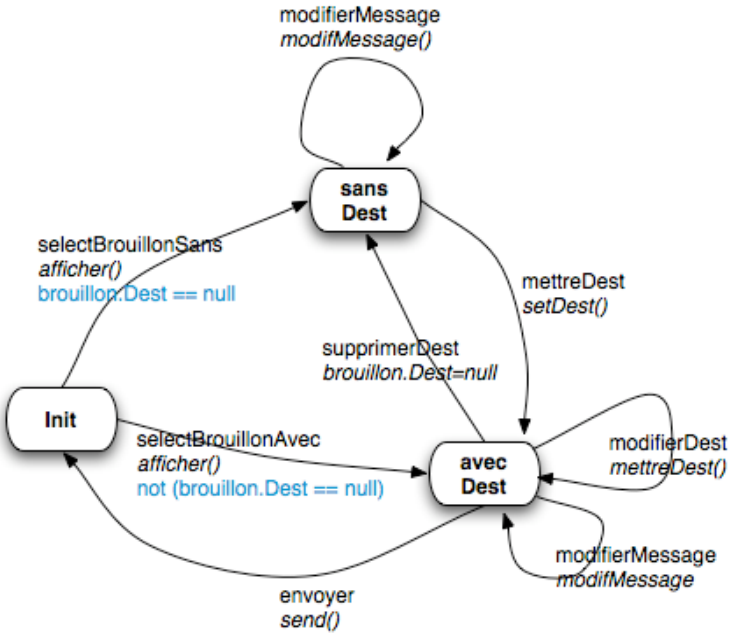


Figure 3.2.1 : Machine à état du dialogue de l'envoi d'un brouillon avec un mailer

Le formalisme des machines à états permet une représentation graphique du dialogue d'une application. Cependant, lorsque le dialogue à formaliser est dense, la machine à états produite est d'une taille importante ce qui la rend illisible.

Dans les grammaires, les non-terminaux ont la même utilité que les sous-dialogues dans les machines à états (voir le paragraphe sur les grammaires, § 2.2.3.3), alors que les terminaux sont utilisés dans l'ordre dans lequel ils apparaissent et sont traités directement.

De plus, depuis le milieu des années 2000, les machines à états peuvent être directement implémentées dans l'application grâce à une bibliothèque développée : SwingStates [Appert et Beaudouin-Lafon 2006, Appert et Beaudouin-Lafon 2006]. Cette bibliothèque repose sur une implémentation en JAVA/Swing.

3.2.1.2. Les interacteurs hiérarchisés

Les interacteurs hiérarchisés [Texier 2000] sont des réifications d'éléments du dialogue issus du modèle H^4 [Guittet 1995]. Ce sont des éléments ajoutés aux boîtes à outils qui permettent la conception du dialogue des applications de Conception Assistée par Ordinateur (CAO). Le but des interacteurs hiérarchisés est de permettre à un concepteur de créer un contrôleur de dialogue supportant le dialogue structuré.

Les études menées sur la définition de ces interacteurs hiérarchisés dans le contrôleur de dialogue de H^4 repose sur l'utilisation d'automates bien que ce ne soit pas un pré-requis pour l'utilisation de H^4.

Le contrôleur de dialogue conçu avec la boîte à outils du dialogue doit appeler les actions de l'application (les questionnaires) en fonctions des données (commandes et paramètres) fournies par l'utilisateur.

Le contrôleur de dialogue de H^4 est composé de plusieurs éléments :

- les **jetons** qui représentent les unités d'informations. Il existe deux types de jetons :
  - les jetons « commande » qui contiennent le nom de la tâche à réaliser. Ils sont représentés dans l'interface par un bouton ou une case de menu.
  - les jetons « paramètre » qui sont définis par le type de données et contiennent une valeur de ce type.
- les **questionnaires** qui représentent une action (tâche dans les travaux proposés). Ce sont les signatures des fonctions qui sont appelées par le contrôleur de dialogue pour réaliser une tâche.
- les **interacteurs de contrôle** qui regroupent les questionnaires représentant les tâches de même niveau d'abstraction. Ils contiennent un réseau de transition qui conserve les jetons reçus et appelle les questionnaires en fournissant les jetons nécessaires.
- le **moniteur** qui récupère les jetons produits dans la couche entrée-sortie et les transmet aux interacteurs en suivant la hiérarchie des tâches. Celle-ci est réalisée à l'aide de la hiérarchie d'interacteurs (de contrôle).

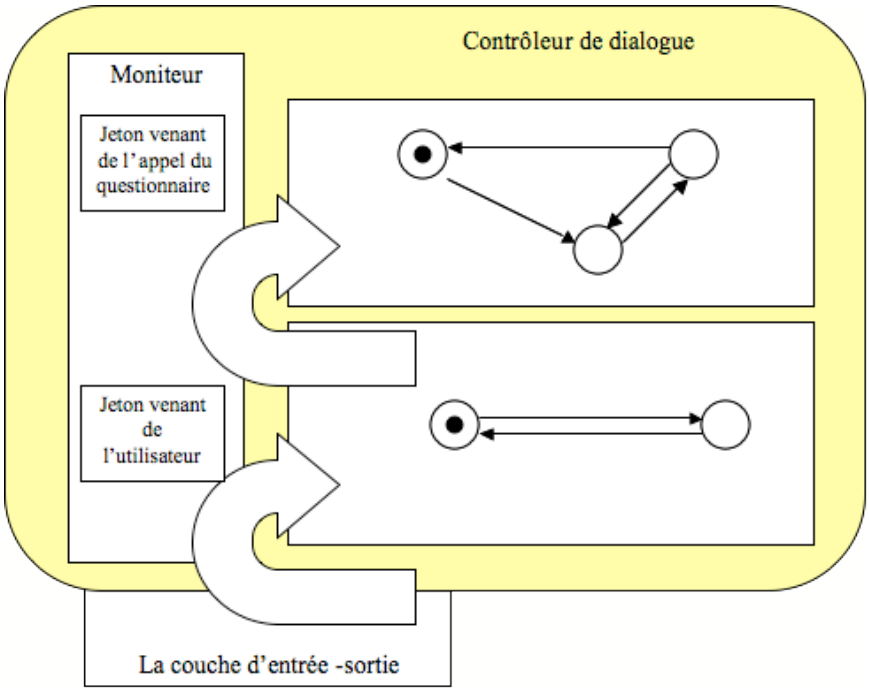


Figure 3.2.2 : Contrôle de l'appel des fonctions et circulation des jetons dans le contrôleur de dialogue de H^4.

Le réseau de transition d'un interacteur de contrôle est utilisé afin de contrôler que les jetons reçus sont bien ceux qui manquaient pour l'appel d'un questionnaire. En effet, puisqu'un questionnaire est spécifié en termes de commande d'activation et de

nature de jetons de paramètres, son appel est strictement déterminé par la suite des transitions qui produisent ces paramètres.

La Figure 3.2.2 présente l'organisation du contrôleur de dialogue pour l'appel des actions. Ce contrôle est réalisé par la distribution des différents jetons par le moniteur.

3.2.1.3. Les Objets Coopératifs Interactifs (ICO)

Le formalisme des Objets Coopératifs Interactifs (ICO) a spécifiquement été développé pour exprimer la dynamique des applications interactives [Palanque et Bastide 1994]. Ce formalisme combine les Objets Coopératifs [Bastide 1992] et des réseaux de Petri. L'outil PetShop [Bastide 2000] a été développé afin d'animer le dialogue d'applications interactives exprimé avec ce formalisme.

Les réseaux de Petri (RdP) ont été créés pour modéliser des processus communicants. Tout comme les machines à états, ils permettent la vérification de propriétés [Olsen 1992] et une représentation graphique du dialogue, cependant, ils ont une plus grande puissance d'expression car ils permettent de compter les répétitions.

Les réseaux de Petri sont constitués de trois types d'éléments, les *places* repère 1 de la Figure 3.2.3), les *transitions* (repère 2 de la Figure 3.2.3) et les *arcs*. Les *places* représentent les états, les *transitions*, les événements et les arcs, les conditions et effets.

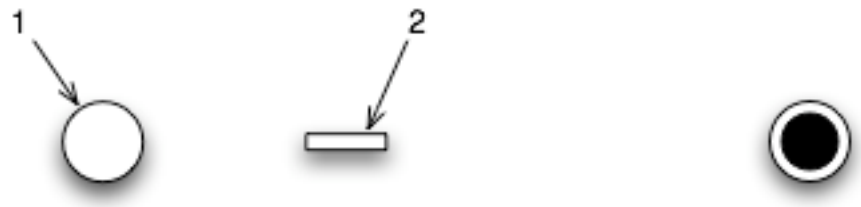
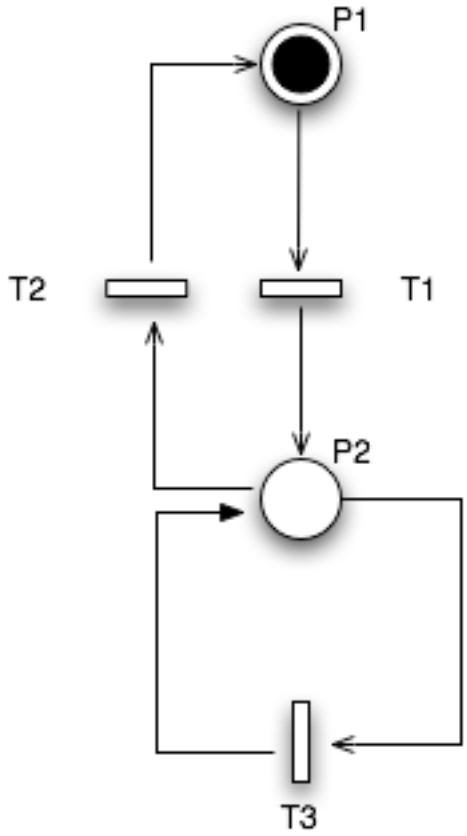


Figure 3.2.3 : les différents éléments d'un réseau de Petri

Pour représenter l'état du réseau, des marques sont utilisées sur les différentes places (repère 3 Figure 3.2.3).

La Figure (Figure 3.2.4) présente le réseau de Petri pour l'envoi d'un email préalablement enregistré dans les brouillons d'un mailer. Nous avons fait l'hypothèse qu'un email enregistré dans les brouillons dispose obligatoirement d'au moins un destinataire (le champ « À » est rempli). Les noms des places et transitions du réseau de Petri sont précisés sous la Figure 3.2.4.



P1 : place initiale
P2 : place de présentation de l'email en cours
T1 : selectEmail
T2 : envoyer
T3 : modifierEmail

Figure 3.2.4 : Utilisation d'un réseau de Petri pour l'envoi d'un email

Initialement, la place marquée avec un jeton est P1, le système est dans l'état correspondant. À partir de cette place, la seule transition exécutable est T1 (activée par l'événement *click*). Une fois cette transition réalisée, le jeton se trouve dans la place P2. Deux transitions sont alors possibles : T3 qui ramène le jeton à la place P2 et T2 qui met le jeton à la place P1.

Les Interactive Cooperative Objects (ICO) permettent la description formelle du comportement dynamique des systèmes interactifs. Ils sont une extension des formalismes des Objets Coopératifs (OC), proposés dans [Bastide 1992], et des réseaux de Petri. Les réseaux de Petri décrivent le comportement des objets et leurs protocoles de communication.

Un objet est composé de deux éléments :

- un **objet coopératif** avec des services utilisateurs
- une **partie présentation**

La liaison entre ces deux parties est assurée par deux fonctions : une fonction d'activation et une fonction de rendu.

Un *objet coopératif* (CO) établit comment l'objet réagit à des stimuli extérieurs en fonction de son état. Ils proposent deux types de services, ceux offerts aux autres objets de l'environnement et ceux offerts aux utilisateurs. La partie *présentation* est l'apparence externe de l'objet. Deux fonctions réalisent la liaison entre ces deux parties. La fonction *d'activation* fait correspondre à une action de l'utilisateur sur un *widget* au service utilisateur à déclencher. Quant à la fonction *de rendu*, elle permet de maintenir la cohérence entre l'état interne du système et son apparence externe.

3.2.1.4. Les machines à états hiérarchiques (HSM)

Les machines à états hiérarchiques [Blanch 2005] constituent un formalisme qui étend le langage de programmation. Elles font partie de la boîte à outils HsmTk [Blanch 2005], dont l'un des objectifs est de faire en sorte que les interactions deviennent des objets à part entière du vocabulaire du programmeur. Pour atteindre cet objectif, elle utilise un formalisme adapté à la description et à la spécification de l'interaction : les machines à états hiérarchiques.

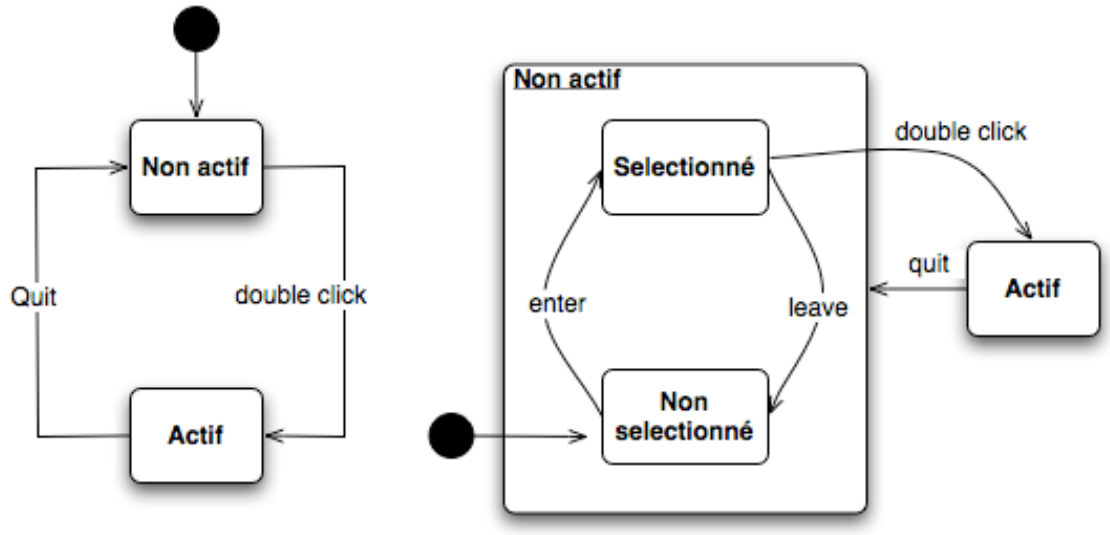
Les machines à états hiérarchique reposent sur le principe des Statecharts [Harel 1987], permettant de ce fait la définition de machines à états dans les états. C'est de cette structure qu'est issu l'organisation hiérarchique des machines à états entre elles. Elle a donc les composants des machines à états : les **états** et les **transitions**.

Le fait d'utiliser le formalisme des *Hierarchical State Machine* permet de raffiner le comportement d'un composant. Par exemple pour deux versions d'un composant email :

- la première se contente de gérer l'activation ou non d'un email (l'email est-il ou non éditable ?) (Figure 3.2.5a).
- la seconde qui, en plus, prend en compte la sélection ou non de l'email dans une liste d'email. L'email est alors « présélectionné » (et donc afficher) mais pas sélectionné, ce qui le rend impossible à éditer (Figure 3.2.5b).

La seconde version de l'activation d'un email est donc un raffinement de la première.

La Figure 3.2.5 présente les machine à états hiérarchiques des comportements des deux versions du comportement de l'email. La hiérarchisation des états permet de conserver au premier niveau de la machine à états raffinée, les états de la machine initiale. Dans cet exemple, les états *Non actif* et *Actif*.



a) b)

Figure 3.2.5 : Modélisation simple et raffinée de la sélection d'un email parmi une liste

3.2.2. Comparaison théorique

Afin de déterminer quel formalisme nous allons utiliser pour exprimer les modèles de dialogue dans nos travaux, nous avons comparé les quatre formalismes que nous avons présentés précédemment : les interacteurs hiérarchisés, les HSM, les ICO et les machines à états. Les points de comparaison que nous avons utilisés répondent aux besoins d'expressivité que nous avons identifiés.

Si l'on adapte la définition de Minsky [Minsky 1988] (voir introduction de la section Chapitre 2) à la définition d'un modèle de dialogue, les questions auxquelles celui-ci doit pouvoir répondre peuvent être classées en trois catégories :

- les questions qui peuvent être posées sur tous les modèles (la représentation de l'objet A est-elle bien conforme à cet objet ? Est-il possible d'utiliser A* pour raisonner à propos de A ?...) C'est ce que nous avons regroupé sous le terme de *pouvoir expressif général*,
- les questions spécifiques relatives à l'usage de ces modèles pour la représentation de la dynamique de l'interaction homme-machine. Nous verrons que la nature particulière de la relation entre l'homme et la machine induit des spécificités importantes dans la représentation de la dynamique. Les modèles permettent-ils de les prendre en compte facilement ? Nous avons regroupé ces points sous le terme de *pouvoir expressif spécifique*,
- les questions relatives à l'utilisation de la modélisation dans l'activité de conception. L'utilisation des modèles de dialogue permet de vérifier des propriétés sur le dialogue des applications interactives, comme les propriétés de vivacité. Là encore, certaines spécificités du domaine de l'Interaction

Homme-Machine doivent être prises en considération. Elles relèvent de la catégorie *intégration dans le processus de conception*.

Notre but dans cette étude est de chercher à identifier le(s) meilleur(s) candidat(s) pour une modélisation du dialogue utilisable en conception d'application interactive. Nous présentons les conséquences de la prise en compte de ces questions dans un formalisme de modèle de dialogue. Une fois tous les besoins d'expressivité mis en lumière pour pouvoir répondre aux questions relatives au dialogue lors de sa conception, nous analysons les apports des quatre formalismes utilisés en conception des systèmes interactifs.

3.2.2.1. Pouvoir expressif général

En ingénierie informatique, la représentation des informations est couramment utilisée afin d'en permettre l'analyse. C'est pourquoi des notations ont été développées pour soutenir les différentes étapes de conception (comme UML [Object Management Group 1997] , par exemple). L'utilisation de ces représentations comme support à l'analyse implique une sémantique non ambiguë et une représentation la plus exhaustive possible. Nous allons essayer de définir l'ensemble des composants nécessaires pour les modèles représentant le dialogue des applications interactives.

Il existe différentes définitions du dialogue des applications interactives. Tarby dans [Tarby 1993] en donne une définition générale : « Le dialogue est l'ensemble des échanges entre un utilisateur et une machine (ou un ensemble de logiciels) ». Cette définition regroupe le dialogue utilisant le langage naturel pour travailler sur un ordinateur et le dialogue défini comme étant une succession d'*actions* de l'utilisateur qui implique une *réaction* du système. C'est cette succession que les modèles de dialogue ont cherché à formaliser. Green [Green 1986] les définit comme des modèles abstraits utilisés pour décrire la *structure* du dialogue entre un utilisateur et un système interactif. La structure du dialogue revient à permettre de prédire, en fonction des actions précédentes de l'utilisateur, et de l'état des variables du système, l'ensemble des actions possibles sur le système ainsi que leurs conséquences.

Même si, comme le dit Green, les trois types de formalismes précités (grammaires, formalismes à états et formalismes à événements) permettent de représenter cette information, les formalismes à base d'états sont la manière la plus *naturelle* de représenter ces échanges. Comme le décrit Thimbleby [Thimbleby 2007], l'interaction peut être réduite à une phrase : l'utilisateur réalise des actions, qui changent l'état du dispositif, qui en retour contrôle des indicateurs[10]. Les grammaires utilisent un niveau d'abstraction élevé, alors que les systèmes à événements, centrés sur la notion de déclencheur d'action, rendent peu facile l'identification des états. De plus, seuls les formalismes à état sont représentés sous forme graphique – facilement compréhensibles même pour un novice.

Les machines à états (ou automates) [Jacob 1982] sont la base des formalismes à états (voir 3.2.1.1). Tous les formalismes pris en compte dans cette étude sont des

[10] "the user performs actions, which change the device state, which in return controls indicators"

formalismes à base d'états. Dans ces formalismes, si les concepts nécessaires au dialogue (représentation des états de l'interface, des actions de l'utilisateur et des réactions du système) sont bien présents, qu'en est-il des questions que l'on peut se poser sur le modèle ? Fondamentalement, l'objectif du modèle de dialogue (Green [Green 1986]) est de permettre de répondre à la question : dans un état donné, quelles sont les différentes actions possibles et leur conséquences ? Les formalismes à état permettent de répondre à cette question à condition d'être déterministes. Le modèle devra donc contrôler cette propriété. Mais d'autres questions pourront être posées quant à la validité du dialogue. En effet, les propriétés de vivacité ou d'atteignabilité par exemple, permettront une analyse plus fine des interfaces.

3.2.2.2. Pouvoir expressif spécifique

De nombreux systèmes temps-réel peuvent être aisément modélisés par des systèmes à état. Le nombre de transitions est relativement modeste, ce qui permet une utilisation optimale des outils associés à ces formalismes. Le domaine de l'interaction homme-machine présente un certain nombre de spécificités qui rendent les choses beaucoup plus compliquées. D'un point de vue général, le nombre de transitions devient rapidement très important. Dans cette section, nous étudions les raisons de cette particularité, et nous décrivons les solutions globales qui peuvent être proposées pour pallier ce problème.

Prise en compte de la diversité des utilisateurs

L'utilisateur n'est pas unique ni uniforme. Son degré d'expertise, tant sur le système en général que dans sa tâche en particulier, peut varier considérablement. Les recommandations ergonomiques admises aujourd'hui suggèrent de prendre en compte le critère de flexibilité (différentes manières d'atteindre un même résultat) par la mise en œuvre de niveaux d'utilisation. Au-delà de ce point, l'existence de publics spécifiques (handicapés, enfants, personnes âgées,…) impose bien souvent des modalités spécifiques d'interaction. Dans tous les cas, le nombre de transitions, voire même d'états à prendre en compte, peut augmenter sensiblement, comme le montre l'exemple de la Figure 3.2.6.

Afin de limiter le nombre d'états (qui peut rapidement devenir important lorsque la description explicite de tous les états est obligatoire), des formalismes ont été développés pour permettre la factorisation des états. C'est le cas des interacteurs hiérarchisés ou bien des machines à états hiérarchiques qui reposent sur le principe des Statecharts.

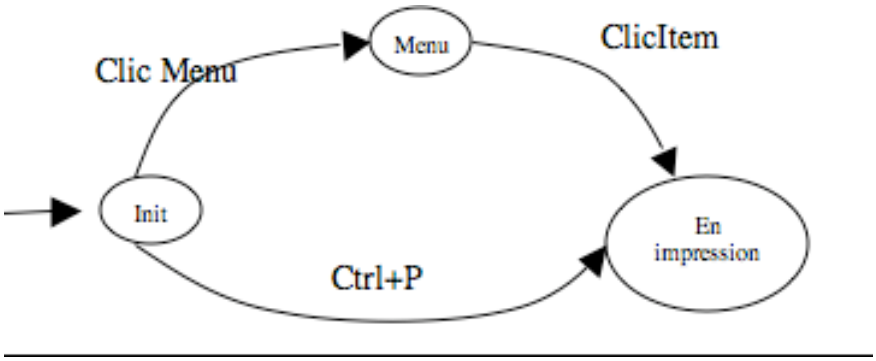


Figure 3.2.6 : "Imprimer" pour deux niveaux d'expertise

Prise en compte des dialogues multi-fils

Les dialogues multi-fils (*multi-threads* en anglais) correspondent à la prise en compte d'un autre volet du critère de flexibilité, en permettant à l'utilisateur d'effectuer des actions en parallèle sur une ou plusieurs applications. Ils permettent de corriger des enchaînements de tâches incorrectement maîtrisés (par exemple, la création d'un client avant l'enregistrement d'une facture), ou de répondre à des interruptions inopinées (réponse urgente au téléphone nécessitant une interrogation d'application). Représenter la réalisation concurrente de plusieurs tâches peut se faire de la même manière que la prise en compte de différents niveaux d'utilisateurs. Un réel parallélisme est cependant rarement présent dans les machines à états. À l'inverse, les réseaux de Petri sont classiquement utilisés pour prendre en compte cette spécificité. C'est le cas en particulier des ICO.

L'erreur humaine

Le propre de l'humain est de commettre des erreurs. Le contrôleur de dialogue étant la partie qui permet d'exprimer la liaison entre l'humain et le système, il est nécessaire qu'il prenne en compte cette spécificité afin de protéger le système (critère d'intégrité [Scapin 1986] et de robustesse [Jambon 1997]) et de permettre à l'utilisateur de se corriger (critère de recouvrement [Scapin 1986]).

La protection du système signifie que le contrôleur de dialogue contrôle l'accès et la modification des données et, de manière générale, contrôle les actions autorisées (prévention des erreurs [Lewis et Norman 1986]). Les formalismes à état contiennent toute l'information nécessaire à la prévention des actions inopportunes. Si les transitions décrivent les actions autorisées, l'absence de transitions dans un état donné décrit la prévention possible des erreurs : l'utilisateur ne doit pas avoir la possibilité de réaliser les actions correspondant aux transitions non autorisées. Les mécanismes de gardes sur les transitions (pour les formalismes basés sur les machines à états) ou de contrôle par jetons (pour les formalismes basés sur les réseaux de Petri) permettent une expression plus précise de ce contrôle. Une garde est, dans un formalisme dérivant des machines à états, une condition booléenne sur une transition. Il faut alors que la condition (garde) soit évaluée à vrai pour que la transition puisse être exécutée. Les formalismes issus des réseaux de Petri expriment le dialogue sous forme de places et de transitions. Chaque place contient des jetons qui conditionnent le franchissement des transitions. Le franchissement

de transition se traduit alors par le *passage* des jetons vers une autre place. Donc, les jetons représentent les conditions sur l'état du système. Le formalisme des ICO ajoute à cette vérification celle du type de jeton et de sa valeur. De plus, les formalismes des ICO et des interacteurs hiérarchisés prévoient le cas d'erreurs survenues lors de l'exécution du code associé à la transition. Ce sont les seuls formalismes de contrôleur de dialogue qui prennent en compte l'expression de ces erreurs d'exécution. Ce mécanisme permet de faire de la prévention du système vers l'utilisateur.

Un deuxième type d'erreur peut survenir : les erreurs de logique de l'utilisateur, qui n'aboutit pas au résultat qu'il avait escompté. L'utilisation est syntaxiquement correcte, mais sémantiquement fausse. Afin de corriger ce type d'erreur, le système doit fournir à l'utilisateur des mécanismes de *recouvrement*. Les mécanismes les plus communément utilisés sont les possibilités d'abandon, ou les fonctions *Undo/Redo*. Là encore, le problème de ces mécanismes est qu'il entraîne de nouvelles transitions, parfois nombreuses, dans le dialogue modélisé sous forme de diagrammes à états. Ces fonctions sont le plus souvent gérées à la main, cependant un formalisme de modélisation du dialogue permet de gérer automatiquement (sans matérialisation de transitions supplémentaires) l'abandon et le « Undo » : le formalisme des interacteurs hiérarchisés. Ce formalisme a été développé pour exprimer le dialogue de systèmes CAO (Conception Assistée par Ordinateur) [Texier 2000]. Le dialogue est alors exprimé sous forme de machines à états hiérarchiquement organisées. Chaque action de l'utilisateur produit un jeton qui est consommé par le contrôleur, ce qui modifie l'état du système. Lorsque tous les jetons nécessaires à la réalisation d'une instruction sont consommés, l'exécution de l'instruction peut produire un jeton qui lui-même est consommé par une machine à états de niveau supérieur. La séquence des actions de l'utilisateur et réactions du système (représentée par une série de jetons) est stockée. Lorsque la commande « Undo » est lancée, les jetons sont dépilés et les machines à états sont remises dans leur état initial.

La gestion des erreurs au niveau du contrôleur de dialogue doit être liée avec la présentation afin d'assurer sa fonction de visualisation de l'état du système. Les formalismes des interacteurs hiérarchisés et des ICO prennent en charge ce lien en assurant l'activation des commandes autorisées (à l'aide de la méthode de connaissance des entrées attendues des interacteurs hiérarchisés et de la fonction d'activation des ICO).

3.2.2.3. Intégration dans le processus de conception

Les points que nous avons étudiés dans les sections précédentes concernent le pouvoir d'expression final des formalismes. Une autre dimension, tout aussi importante, concerne l'aptitude d'un outil à supporter la phase dynamique de conception. La conception des applications interactives est universellement considérée comme une processus itératif. Le formalisme s'adapte-t-il bien à cette particularité ? L'intégration du dialogue dans l'architecture de l'application est indispensable. Le modèle le permet-il aisément ? Enfin, l'existence d'une base formelle est-elle utilisée pour mettre en œuvre des outils de vérification ou d'aide à la conception ?

Conception itérative

Du fait de son aspect itératif, la conception d'une application interactive nécessite l'utilisation de modèles pouvant être itérativement élaborés. À chaque itération, des compléments sont apportés pouvant engendrer des modifications de l'application conçue (quelle que soit l'étape courante du processus de conception). Afin de faciliter la prise en compte de ces modifications par le(s) concepteur(s) (*maintenance*), l'une des approches abordées dans le domaine informatique est la modularisation. Les premières modularisations d'applications ont été permises par l'application des modèles d'architecture (comme [Pfaff 1985], [Coutaz 1987] ou [Depaulis, et al. 2006]). Chacun de ces modèles permet de diviser l'application en plusieurs parties plus ou moins indépendantes. À chacune de ces parties est associé un rôle (comme celui de contrôleur de dialogue par exemple). De ce fait, la modification d'une partie (d'un module) reste indépendante des autres. Le même procédé peut être appliqué à la représentation du dialogue afin d'en faciliter la modification tout au long du processus itératif de conception. Le plupart des modèles de dialogue permettent cette modularisation : les modèles hiérarchiques, de par leur nature, et les modèles qui communiquent entre eux (comme les ICO et les machines à états).

Intégration dans l'architecture de l'application

Dans les modèles d'architecture utilisés dans le domaine des IHM, le dialogue fait l'objet d'un (Seeheim [Pfaff 1985],…) ou plusieurs modules spécifiques (PAC [Coutaz 1990],…). Cette structuration permet de modulariser l'application et d'isoler chaque composant en fonction de son rôle dans l'application. Par exemple, dans la définition de référence du modèle de Seeheim [Pfaff 1985], le dialogue est décrit dans [Dix, et al. 1993] comme étant une double liaison :

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

Le *raccord* du dialogue avec les autres parties n'est pris en compte que pour le formalisme des interacteurs hiérarchisés qui sont des composants du contrôleur de dialogue dans l'architecture H^4 [Depaulis, et al. 2006] (voir Figure 2.2.8 dans 2.2.3.2). Comme ARCH [Bass, et al. 1988], ce modèle est composé de cinq composants, le sommet de l'arche étant le contrôleur de dialogue. Dans H^4, le rôle, la composition et le fonctionnement du contrôleur de dialogue sont précisément définis et intégrés dans le modèle d'architecture. Les interacteurs hiérarchisés ont pour rôle essentiel l'appel des primitives du système en fonction des entrées de l'utilisateur. Leur nom est dû à la prise en compte d'une partie de l'interaction (*interacteur*) et à l'organisation hiérarchique (réalisée en fonction du niveau d'abstraction) qui les caractérise.

De récents travaux [Appert, et al. 2009] proposent de séparer l'interaction du dialogue exprimé sous forme d'automates exprimés à l'aide de SwingStates [Appert et Beaudouin-Lafon 2006] en déléguant la gestion des événements de bas niveaux à la boîte à outils ICON [Dragicevic et Fekete 2004]. Avec ces travaux, le formalisme des interacteurs hiérarchisés est le seul qui permet l'isolement du contrôleur de dialogue de l'interaction. Du fait que les transitions sont déclenchées par des actions de l'utilisateur (donc

sémantiquement très liées à l'interaction utilisée), les autres formalismes de dialogue associent les événements produits par les actions de l'utilisateur (interaction) avec les transitions du contrôleur de dialogue. Les transitions des interacteurs hiérarchisés sont déclenchées par la réception de jetons, sans porter attention à la provenance de ces jetons. Ainsi ils peuvent provenir de l'interaction (dans la présentation) ou résulter d'une production interne au contrôleur de dialogue.

Aide au développement et/ou à l'implémentation

Afin d'inclure les modèles de dialogue dans le processus de conception des applications interactives (et ainsi tirer parti des vérifications pouvant être faites sur les aspects formels), deux principales approches sont suivies. La première intègre les modèles de dialogue en les associant à différents autres modèles (modèles de tâches, modèles d'interaction…) pour produire des interfaces. L'association des différents modèles a pour but d'exprimer l'ensemble des différents points de vue (point de vue de l'utilisateur, point de vue du système…). Cependant, ces approches basées sur modèles sont difficiles à utiliser et restent peu utilisées, notamment dans le domaine industriel. La seconde vise à *transcrire* les modèles dans l'implémentation des applications. L'intérêt de cette approche est de permettre aux programmeurs de tirer parti des vérifications pouvant être faites sur les formalismes utilisés sans impliquer un coût d'utilisation (et d'apprentissage) trop important.

Cependant, comme nous l'avons exposé en introduction de cette session, peu de modèles ont fait l'objet d'études pour intégrer le dialogue dans le processus de conception des applications. Le premier de ces formalismes est celui des interacteurs hiérarchisés pour lequel a été développée une boîte à outils de dialogue [Texier 2000]. Celle-ci propose toutes les fonctionnalités pour concevoir un dialogue respectant strictement le formalisme. L'outil PetShop a été développé afin d'animer le dialogue d'applications interactives exprimé avec le formalisme des ICO. Des bibliothèques ont été développées afin *d'intégrer* les autres formalismes (automates et machines à états hiérarchiques) dans le code de l'application. Les dialogues exprimés sous forme de machines à états peuvent ainsi être facilement implémentés à l'aide de la bibliothèque JAVA *SwingStates* [Appert et Beaudouin-Lafon 2006]. Enfin, les machines à états hiérarchiques (HSM) permettent la définition de machine à états dans les états d'autres machines à états (suivant le principe du formalisme des StateCharts). Tout comme pour le formalisme des machines à états, une bibliothèque a été développée pour permettre l'implémentation des HSM : HsmTK [Blanch et Beaudouin-Lafon 2006].

Exploitation des aspects formels

Les aspects formels peuvent être exploités à deux niveaux pour la conception des applications interactives. Un premier niveau, que nous qualifierons d'interne et un second niveau, dit externe.

Vérifications internes. Les vérifications internes sont les vérifications inhérentes à l'utilisation d'un formalisme. L'utilisation d'un formalisme à base d'états permet par exemple de vérifier que tous les états sont atteignables (atteignabilité) ou qu'il est possible de *sortir* de n'importe lequel des états (non blocage) [Aït-Ameur, et al. 2005]. En plus de

ces vérifications communes à l'ensemble des formalismes à base d'états, l'utilisation de PetShop, permet au concepteur d'animer un dialogue défini par des ICO et donc d'avoir une « visualisation » du comportement de sa future application.

Vérifications externes. Les vérifications externes sont les vérifications qui peuvent être faites sur le dialogue en fonction d'éléments extérieurs. Des études ont notamment été proposées pour vérifier la cohérence du dialogue en fonction de spécifications exprimées sous forme de modèles de tâches. C'est le cas des travaux menés pour vérifier la cohérence du dialogue exprimé sous forme d'ICO et de modèle de tâches. Ces travaux proposent d'utiliser les scénarios [Navarre, et al. 2001].

3.2.2.4. Bilan de la comparaison théorique

L'étude des formalismes utilisés pour exprimer les modèles de dialogue en fonction de leur utilisation en conception, nous a permis d'identifier onze besoins différents. Nous allons les résumer ici et pour chacun d'eux, préciser dans quelle mesure les formalismes étudiés y répondent (Tableau 3-14). Ces formalismes étudiés sont les automates implémentés à l'aide de la bibliothèque SwingStates (SS), les interacteurs hiérarchisés (IH), les machines à états hiérarchiques (HSM) et les Objets Coopératifs Interactifs (ICO).

Tous ces formalismes ont une *représentation graphique* ; ils sont à base d'états, et donc bénéficient tous des apports de l'expressivité états/transitions : ils permettent *l'expression des différents états et les changements d'état*, l'*expression de différents moyens d'arriver au même état* et possèdent une *sémantique formelle* bien définie. Pour ce dernier point, il nous faut cependant noter que l'utilisation des techniques formelles, même si elle est rendue possible par le formalisme, est rarement intégrée directement dans l'usage des outils associés. L'animation proposée par PetShop peut être vue comme une certaine forme de vérification en temps-réel, comme, d'une manière générale, l'utilisation directe des formalismes pour implémenter les systèmes. Néanmoins, si tous ces formalismes permettent une vérification « interne » théorique, aucun ne propose d'outil vraiment évolué de vérification directe de propriétés, et seuls les formalismes hiérarchiques et plus particulièrement les ICO ont fait l'objet d'études sur les vérifications « externes ».

Tous les formalismes prennent également en compte, mais à des degrés divers, l'expression de la *concurrence* (réalisation concurrente d'actions différentes) et la *prévention* (prévoir et protéger contre les erreurs). Comme nous l'avons indiqué précédemment, la concurrence peut être exprimée en multipliant les états (comme le font les machines à états), cependant cela limite la modélisation à des applications de petite taille (un grand nombre d'actions possibles entraîne une explosion du nombre d'états et donc un manque de lisibilité du modèle). C'est pour répondre à ce besoin que des formalismes qui permettent la modularisation du modèle ont été développés (comme les Interacteurs Hiérarchisés, les HSM et les ICO, ces derniers s'appuyant en plus sur les réseaux de Petri, naturellement adaptés à la modélisation concurrente). Notons cependant qu'une réelle concurrence nécessite de s'appuyer sur une implémentation elle-même concurrente, ce qui n'est pas le cas de la plupart des boîtes à outils d'interaction.

Les interacteurs hiérarchisés et les HSM imposent la *structuration* des différentes fractions du dialogue en fonction du niveau d'abstraction. Cette structuration donne un rôle sémantique non ambigu à chacun des modules et de ce fait, peut contribuer à diriger le concepteur lors de sa modélisation. De plus, il permet de concevoir le dialogue en raffinant étape par étape les transitions et/ou les états et de ce fait ; il s'adapte « naturellement » à une conception itérative.

	SS	IH	HSM	ICO
Expression des états et des changements	√	√	√	√
Représentation graphique	√	√	√	√
Expression de différents moyens d'arriver à un même état	√	√	√	√
Concurrence	+	++	++	+++
Prévention	+	++	+	+++
Recouvrement		√		
Activation		√		√
Modularisation		√	√	√
Structuration		√	√	
Intégration dans l'architecture		√		
Utilisation du formel	+	++	+	++

Tableau 3-14 : Comparaison des modèles de dialogue

Tous les formalismes ont des gardes et/ou des jetons devant être vérifiés et/ou présents pour que les transitions soient déclenchables, donc tous permettent l'expression de la gestion de la *prévention* par le contrôleur. Le formalisme des ICO est le plus expressif sur ce point puisque, d'une part, il combine plusieurs mécanismes de prévention (présence de jetons + arrivée d'un événement + vérification du type de jeton), et d'autre part, il prévoit les cas d'erreurs du système afin de les gérer. Ce besoin de haut niveau de prévention peut être expliqué pour l'utilisation des ICO pour un haut niveau de sécurité [Palanque, et al. 1998], au contraire des autres formalismes développés pour lesquels l'aspect « sécurité » n'est pas prioritaire.

Si la liaison entre le contrôleur de dialogue et le système est assurée par tous les formalismes par l'appel de fonctions (du noyau fonctionnel), seuls les ICO et les interacteurs hiérarchisés prévoient des mécanismes permettant de lier le contrôleur de dialogue avec la présentation. Ce lien consiste à *activer* les « commandes» de la présentation en fonction des transitions attendues. Le lien avec la présentation est renforcé avec le mécanisme de *recouvrement* mis en place dans les interacteurs hiérarchisés. Celui-ci prévoit le traitement de jetons particuliers (jetons *Undo*) qui permettent le rejeu de l'ensemble des actions (jetons) sauf le dernier, ce qui a pour effet de ramener le système dans l'état précédent. Notons que ce mécanisme de gestion « automatique » du *Undo* n'est applicable que parce que les applications concernées sont des applications CAO. Cette gestion du *Undo* ne peut s'appliquer à tout type d'application (une application faisant appel à de l'aléatoire comme la distribution aléatoirement de cartes par exemple). La gestion du *Undo* dans toutes les applications interactives requiert que les fonctions correspondantes soient prévues au sein du noyau sémantique [Fekete 1996].

Enfin, bien que le contrôleur de dialogue soit *intégré dans tous les modèles d'architecture*, seul le formalisme des interacteurs hiérarchisés définit formellement ses fonctions et ses liens avec les autres modules.

3.2.3. Bilan du choix de formalisme de dialogue

En conclusion, d'après cette étude, et en fonction des points de comparaison que nous avons définis, nous pouvons considérer deux niveaux d'abstraction d'expressivité des formalismes : (1) les formalismes de faible niveau d'abstraction (très proche de l'implémentation) (c'est le cas des automates de Swingstates) ; et (2) les formalismes permettant l'expression de niveaux d'abstraction différents (les trois autres formalismes font partie de cet ensemble).

De part leur définition, le formalisme des HSM est inclus dans le formalisme des Interacteurs Hiérarchisés (IH). Les formalismes des IH et des ICO apparaissent donc, en fonction de nos critères, les plus adaptés à notre étude.

Les nombreuses études menées autour des ICO, la présence de l'outil d'animation PetShop, ainsi que la base formelle reposant sur les réseaux de Petri, font de ce formalisme un bon candidat pour le développement d'outils d'aide à la conception, incluant des outils opérationnels de vérification. Les interacteurs hiérarchisés, quant à eux, possèdent certains de ces outils, et proposent des solutions dans la démarche de conception elle-même. Cependant, ce formalisme ayant été développé pour les dialogues des systèmes CAO, il demande à être étendu pour pouvoir exprimer l'ensemble des dialogues des applications interactives tout en conservant ses apports.

La principale différence entre ces deux formalismes réside dans la structuration des différents niveaux d'abstraction exprimés, celle-ci étant imposée dans le formalisme des IH. En effet, si les ICO et les IH permettent tous les deux d'exprimer le dialogue des applications interactives sous forme de plusieurs modules communiquant entre eux, les IH structurent cette décomposition. De ce fait, chaque interacteur a une sémantique induite, liée au niveau d'abstraction auquel il appartient.

De plus, l'inclusion de la conception des IH dans le processus de conception des interfaces nous semble plus intéressant pour la suite de nos travaux. C'est pourquoi, bien que les ICO et les IH sont tout deux des formalismes pouvant être utilisés pour nos besoins, nous avons choisi d'utiliser le formalisme des IH.

3.3. Étude préliminaire : dérivation du modèle de dialogue à partir du modèle de tâches

Nous avons évoqué, dans la section 2.3.2, quelques travaux menés dans le domaine de l'IDM, appliqués aux IHM. Il se trouve que la majorité de ces travaux utilisent une approche de génération afin de dériver tout ou partie d'une application interactive du

modèle de tâches. L'IDM ne se résume pas à cette approche. D'une part, les transformations de modèles ne sont pas forcément unidirectionnelles, et une coopération de modèles est possible. D'autre part, la génération s'accommode mal d'un cycle de développement itératif, avec de nombreux ajustements et modifications des solutions précédemment retenues. Pourtant, cette conception itérative est reconnue comme indissociable de la conception centrée-utilisateur.

Dans la présente section, nous mettrons en lumière les différentes limites de l'approche exclusivement générative à partir des modèles de tâches.

3.3.1. Les approches de génération

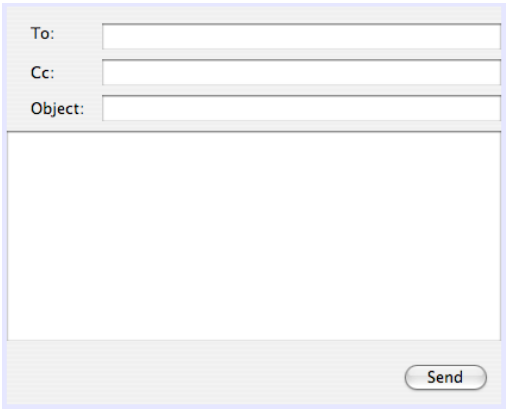
Le développement de multiple plateformes et dispositifs interactifs nécessite de produire des interfaces utilisateurs (UI) capables de s'adapter au contexte d'utilisation [Calvary, et al. 2002]. Pour concevoir ce type d'interfaces, une des stratégies proposées est la dérivation des différentes UI adaptées à chacune des différentes plateformes à partir du même modèle de tâches contenant les informations communes. Cette approche a été suivie par ARTStudio [Thévenin 2001], TERESA (Transformation Environment for inteRactivE Systems representAtions) [Mori, et al. 2003] et le Dygimes framework (Dynamically Generating User Interfaces for Mobile Computing Devices and Embedded Systems) [Luyten 2004]. Certaines de ces approches ont été présentées dans la section 2.3.2.

Après différentes étapes, ces outils produisent les interfaces finales spécialisées pour chaque plateforme. Pendant ces étapes, le modèle de tâches est complété par d'autres modèles (comme les modèles d'interaction et de domaine) ou est exploité pour produire d'autres modèles (comme le modèle de dialogue). Des informations sont ajoutées afin de souligner les contraintes des plateformes (spécifiées dans le modèle de tâches) ou pour insérer les informations des autres modèles (par exemple l'ajout des widgets dans Dygimes). Ces ajouts peuvent modifier le modèle de tâches initial.

Une autre méthode pour obtenir des parties de l'interface est l'exploitation des éléments du modèle de tâches. Dans TERESA et Dygimes, le modèle de dialogue est obtenu comme suit : les opérateurs dans le modèle de tâches induisent directement une chaîne de fenêtres, ce qui produit la séquence de contrôle. Des optimisations peuvent être faites, comme dans TERESA, en appliquant des heuristiques. De plus, l'utilisateur peut aussi intervenir durant le processus de génération. Ces deux points peuvent conduire à la modification du modèle de tâches initial.

3.3.2. Tâches et présentation des interfaces

Dans une approche basée sur les tâches, la présentation de l'interface doit respecter le modèle de tâches. Cette consistance entre le modèle de tâche et la présentation peut être exploitée pour valider l'interface. Afin de générer les interfaces qui respectent les usages utilisateur les modèles de tâches ont besoin de plus d'informations. Les modèles de tâches décomposent les tâches utilisateur dans le but de spécifier les activités des utilisateurs et les besoins des utilisateurs pour réaliser les tâches. Cependant, ceci est fait indépendamment de la présentation de l'interface alors qu'il y a des recommandations à respecter [Bastien et Scapin 1993]. Par exemple, pour compléter les champs d'un formulaire, l'utilisateur peut utiliser un ordre usuel. Ainsi, pour compléter (éditer) un nouvel email, il est communément admis d'utiliser une présentation d'interface avec les champs organisés comme sur la Figure 3.3.1a : adresses du destinataire et des autres destinataires (destinataires en copie conforme ou non), sujet et message.

a) 

b) 

Figure 3.3.1 : Deux présentations différentes. a) Présentation avec l'ordre usuel et b) Présentation sans l'ordre usuel

C'est un ordre de présentation communément admis mais qui n'est pas obligatoirement suivi (l'utilisateur peut commencer l'édition d'un email par l'écriture du message sans que cela n'empêche l'exécution de la tâche d'édition d'un email). Il n'est donc pas possible de l'imposer dans le modèle de tâche sans réduire la flexibilité de l'interface conçue. Donc, la complétion des champs doit pouvoir être réalisée par l'activation de toutes les tâches en même temps. La connaissance de l'ordre d'exécution des tâches dépend du modèle de tâches (données représentées dans les modèles de tâches par les opérateurs d'ordonnancement) mais comment représenter les widgets qui permettent l'exécution des tâches dont l'ordre dépend des choix du concepteur. La Figure 3.3.1b montre une présentation d'interface qui permet l'exécution du même ensemble des tâches sans respecter l'ordre usuel.

Un moyen d'arranger les widgets est de suivre l'ordre dans lequel les tâches sont décrites dans l'arbre de tâches (TERESA suit ce procédé). Cette approche utilise la sémantique d'ordre de la description du modèle de tâche. Cette sémantique complète la sémantique des opérateurs. Les tâches liées par l'opérateur de *concurrence* (*Pas d'ordre*

dans le formalisme K-MAD) doivent pouvoir être réalisées de manière concurrente, mais l'ordre usuel est l'ordre de description.

De plus, la présentation nécessite d'autres informations. Chaque plateforme a ses propres contraintes d'espace, c'est pourquoi la présentation des applications doit être adaptée à chaque plateforme. Déduire du modèle de tâche les contraintes d'espace est impossible. Toutefois, Dygimes produit la présentation de différents appareils en se focalisant sur les contraintes de taille des écrans. Cette adaptation de taille est réalisée par le biais d'un algorithme basé sur la taille des différents widgets ajoutés. ARTStudio propose l'adaptation à la taille des écrans grâce à l'ajout d'un modèle de présentation. Ces modèles définissent la position de tous les éléments dans les fenêtres.

Enfin, une des fonctions de la présentation est de présenter les informations dont l'utilisateur a besoin pour exécuter une tâche. Généralement, les tâches cognitives sont ajoutées avant les tâches interactives afin de modéliser les choix stratégiques de l'utilisateur. Fréquemment, l'exécution de ces tâches cognitives requiert des données devant être affichées. Par exemple, le choix d'un nom dans une liste requiert que la liste soit affichée. L'utilisation d'objets pourrait exprimer ce besoin, mais, à notre connaissance, aucun travail de recherche n'a encore exploré cette voie.

3.3.3. Tâches et contrôle de dialogue

Dans la majorité des modèles d'architecture d'IHM, le contrôleur de dialogue joue un rôle central, composé de deux responsabilités : il doit associer les actions des utilisateurs avec les procédures et les fonctions du noyau fonctionnel ; et il doit contrôler le dialogue (les échanges entre les utilisateurs et l'application). Ce dernier point a pour but d'ordonner les actions possibles en fonction du noyau fonctionnel et des états de l'interfaces, nous avons nommé cette responsabilité le contrôle temporel.

3.3.3.1. Le contrôle temporel

Les modèles de tâches utilisent des opérateurs temporels qui permettent de représenter l'organisation temporelle des tâches. Cela semble très proche du contrôle des actions disponibles. De ce fait, certains éditeurs de modèle de tâches, comme K-MADe ou CTTE, fournissent des outils de simulation et l'opportunité de générer des scénarios de tâches. En utilisant ces outils, le concepteur exécute son modèle de tâches en sélectionnant les tâches qu'il souhaite exécuter à partir de l'ensemble des tâches actives. C'est pourquoi il organise hiérarchiquement les tâches en utilisant les opérateurs temporels.

L'évaluation des ensembles d'actions exécutables doit être réalisée dans le contrôle de dialogue des applications interactives. Les ensembles de tâches actives [Paternò 2001] et d'actions exécutables intuitivement apparaissent partager des similarités, et sont souvent considérés comme identiques [Luyten 2004, Mori, et al. 2003].

Cependant, nous pensons que ces deux ensembles représentent deux points de vue différents de l'application. Une tâche du modèle de tâches ne correspond pas toujours à une action, par exemple une tâche utilisateur est entièrement exécutée par un utilisateur. Bien que cette tâche puisse jouer un rôle clé dans l'exécution de l'application, cette tâche ne sera pas représentée dans l'interface comme une partie de l'ensemble des actions exécutables.

De plus, le passage d'un ensemble de tâches actives à un autre est réalisé via l'exécution de tâches spécifiques (parmi l'ensemble des tâches actives). Il est cependant, difficile de détecter quand l'exécution des tâches est terminée. Dans l'exemple illustré sur la Figure 3.3.2, la tâche système *Envoyer email* peut seulement être réalisé une fois que le destinataire a été entré (exécution de la tâche *Editer dest* terminée). Cependant, ces deux tâches appartiennent au même ensemble de tâches actives. Le statut d'activation de la tâche *Envoyer email* est défini en fonction du résultat de l'exécution de la tâche *Editer dest*.

3.3.3.2. Lien entre les tâches et le noyau fonctionnel

Les interfaces ont besoin d'être liées au noyau fonctionnel afin d'activer les fonctions et les procédures devant être exécutées. À travers ce lien, les différentes actions exécutables sont définies en fonction des actions préalablement exécutées. Certaines informations concernant les tâches concernées sont nécessaires pour ce processus, et certaines variables sont manipulées.

Les tâches liées

Les liens des modèles de tâches sont utilisés pour représenter les décompositions des tâches ainsi que l'organisation temporelle entre les tâches. Les liens peuvent être exprimés entre les tâches sœurs ou entre une mère et ses filles. Cela est exprimé dans le modèle de tâches. Cependant, l'exécution d'une tâche peut être liée à une autre sans qu'elles soient sœurs ni en ayant une relation mère-fille. Par exemple, sur la Figure 3.3.2, la tâche *Envoyer email* peut être exécutée uniquement une fois qu'une adresse a été éditée. De plus, lorsque cette adresse est complètement effacée, il doit être impossible d'activer la tâche *Envoyer email*. Donc, les exécutions des tâches *Envoyer email* et *Saisir dest* sont liées.

La représentation des relations entre les tâches dans le modèle de tâches est parfois un défi. Une première approche repose sur l'utilisation d'un opérateur de désactivation. Dans notre exemple, il peut être utilisé pour séparer les ensembles des tâches réalisées avant et après l'entrée d'une adresse de destinataire (Figure 3.3.2). Les tâches *Ecrire message*, *Saisir sujet* et *Entrer dest CC* sont itérativement réalisées. Le premier ensemble de tâches regroupe la première tâche abstraite *Completer email* et la tâche *Saisir dest*, alors que la seconde tâche *Completer email* appartient à l'autre ensemble de tâches (dans [Paternò 2001] un algorithme pour calculer les ensembles des tâches actives est présenté) bien qu'ils manipulent le même objet email. Cette conception ne répond pas à la problématique de la suppression. Il est en effet possible d'envoyer un email même lorsque l'utilisateur a supprimé une adresse.

Les opérateurs temporels ne permettent pas une description précise du contrôle mais K-MAD propose d'utiliser les objets et les conditions pour augmenter cette description. K-MADe permet la définition d'objets, de conditions (pré- et post-). Lorsqu'une adresse est entrée, l'objet email en cours de modification passe de l'état *non_envoyable* à l'état *envoyable* (modélisé pas deux groupes différents d'emails), il s'agit toujours du même objet *email* qui est complété et déplacé. La tâche *Envoyer email* peut être exécutée uniquement si au moins un email se trouve dans le groupe *envoyable*. Cette conception permet de décrire les tâches *Ecrire email*, *Saisir sujet* et *Saisir dest CC* une seule fois. Cependant, K-MAD ne fournit pas de solution à propos du problème de la suppression.

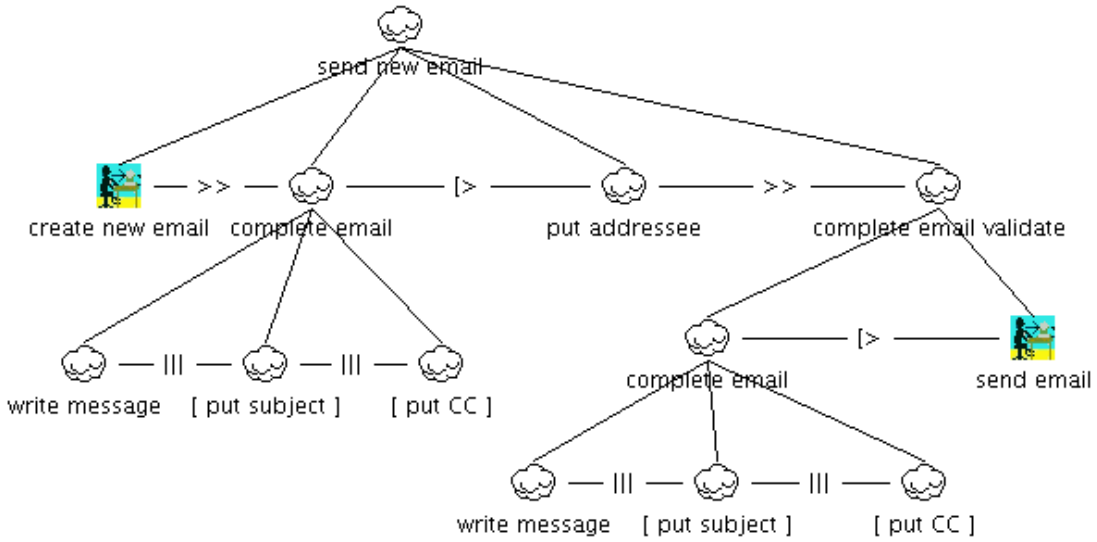


Figure 3.3.2 : Diagramme CTT d'envoi d'un email

Les objets

Les interfaces peuvent avoir besoin de manipuler des variables afin d'appeler les fonctions et les procédures. Il y a deux types de variables. Le premier correspond aux *objets du monde* (comme, par exemple un email) : ce sont les objets de l'application. Les seconds sont seulement créés pour les besoins de l'interface mais n'existent pas dans le monde réel (comme un booléen). Les modèles de tâches représentent le point de vue de l'utilisateur et donc, ne décrivent que les variables qui sont manipulées par les utilisateurs (correspondant aux *objets du monde*). Cependant, pendant la conception de l'application, le point de vue du système peut imposer l'ajout de variables qui ne correspondent pas à l'état du monde et donc ne sont pas représentés initialement dans le modèle de tâches.

3.3.4. Prise en compte des erreurs de l'utilisateur

Une des particularités des utilisateurs est qu'ils commettent des erreurs et parfois changent d'avis. Les modèles de tâches ne sont pas un bon support pour décrire les erreurs utilisateur et exprimer le moyen de les corriger. Même s'ils peuvent être utilisés pour exprimer les erreurs, nous allons voir dans cette section qu'ils ne sont pas le meilleur moyen d'exprimer la correction des erreurs.

3.3.4.1. Les erreurs entre l'intention et l'exécution

Comme Norman l'a présenté dans ses travaux, l'intention et l'exécution sont deux niveaux différents des activités [Norman 1984]. Alors que les modèles de tâches indiquent l'intention de l'utilisateur –the « mental characterization of the design goal »[11]- les interfaces sont conçues pour l'exécution des tâches. Pendant la transcription de l'action mentale en une action concrète, il faut faire attention afin de prendre en compte les erreurs de l'utilisateur.

Les modèles de tâches décrivent habituellement les activités de l'utilisateur sans les erreurs. L'inclusion des erreurs dans les modèles implique des représentations difficiles à lire. Par exemple, lorsqu'un utilisateur entre une adresse de destinataire pour compléter le nouvel email, la tâche interactive est *Saisir dest* décrit le but souhaité de l'utilisateur, cependant il peut entrer une adresse invalide. Afin de compléter le modèle K-MAD de la Figure 3.1.5 pour prendre en compte l'entrée d'une adresse invalide, deux tâches doivent être ajoutées. La tâche *Saisir dest* est composée de deux tâches *Saisir adresse valide* et *Saisir adresse invalide*. La tâche *Saisir dest* est alors une tâche itérative réalisée tant que l'email courant n'est pas prêt à être envoyé. Donc, cette tâche est exécutée tant que l'adresse du destinataire n'est pas valide.

Cependant, les modèles de tâches n'expriment pas comment le choix entre l'exécution des tâches *Saisir adresse valide* et *Saisir adresse invalide* alors que cette information est nécessaire pour la protection des erreurs [Bastien et Scapin 1993]. Les interfaces doivent permettre la correction des erreurs par l'utilisateur. L'arbre de tâche peut être complété afin de prévoir cette correction (Figure 3.3.3). Cependant, le modèle de tâche résultant est imposant et difficilement lisible.

[11] la caractérisation mentale du but

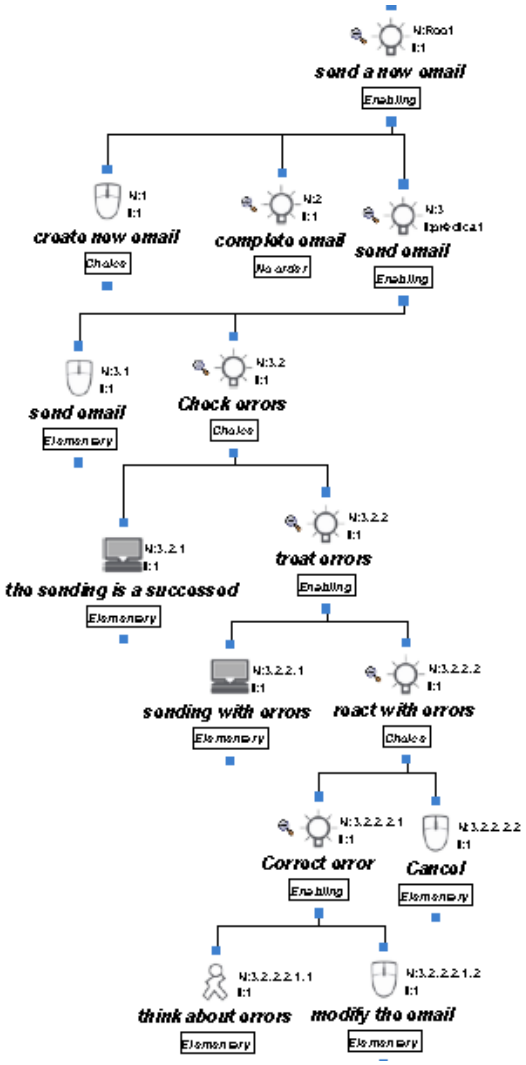


Figure 3.3.3 : Le modèle de tâche de l'envoi d'un email avec la correction des erreurs utilisateur

3.3.4.2. Annulation (Undoing)

Un utilisateur peut changer de but tout en utilisant l'application. Donc, l'interface doit permettre l'annulation d'une action utilisateur (retour à l'état précédant la dernière action). Cependant, les modèles de tâches ne prennent pas en compte les tâches d'*annulation*. Un moyen consiste à ajouter une tâche d'annulation pour chaque action. Mais, cette conception pose deux questions : (1) Est ce que toutes les tâches peuvent être annulées ? (2) Est-ce qu'une même tâche d'*annulation* gère plusieurs actions utilisateur ?

Des tâches non-annulables

Annuler une action utilisateur est une fonction clé qui permet la correction des erreurs utilisateur. Mais toutes les tâches peuvent-elles être annulées ? Par exemple, lorsqu'un utilisateur envoie un email (en cliquant sur le bouton *Envoyer*), le système réalise l'action associée. Cette action ne peut pas être annulée car il est impossible de faire revenir un email une fois que celui-ci a été envoyé. De plus, le concepteur peut vouloir ne pas autoriser le retour en arrière après l'exécution de certaines tâches. Par exemple, lorsqu'un mot de passe est mémorisé par le système, si l'utilisateur le modifie, alors il peut être prudent de ne pas autoriser son annulation. Donc, certaines tâches ne peuvent pas être annulées parce que les actions qui leur sont associées sont des actions finales ou parce que le concepteur ne souhaite pas l'autoriser.

Regroupement de tâches

Quelle est la signification d'une tâche d'*annulation* ? Lorsqu'un utilisateur réalise un retour en arrière (action d'*undo*), cela concerne la dernière action qui est supprimée. Les actions correspondent aux tâches atomiques (tâches feuilles). Donc, il n'est possible de supprimer que les tâches atomiques (la suppression de toutes les tâches filles entraîne la suppression de l'action de la tâche mère). Mais il n'est pas toujours facile de savoir ce qui doit être supprimé. Par exemple, supprimer une tâche d'écriture signifie-t-il que tout ce qui est écrit est supprimé ou que seul le dernier mot est supprimé ?

3.3.5. Tâches et interaction

Les modèles de tâches ne sont pas conçus pour spécifier quel type d'interaction est utilisée pour exécuter les tâches. Cependant, pour faire une interface complète, il est nécessaire de connaître quelle interaction est utilisée. Afin d'ajouter cette information, certaines approches comme le Dygimes framework [Luyten 2004] associent les tâches atomiques avec les widgets qui permettent leur exécution. Avec ce type d'approche, l'interaction est uniquement composée d'interactions disponibles avec les widgets. De plus, elle impose un lien bijectif entre les tâches atomiques et les widgets, ce qui implique des limitations importantes de l'interaction.

3.3.5.1. Ajouter l'interaction

Les modèles de tâches décrivent quelles sont les tâches que l'utilisateur doit réaliser pour atteindre leurs buts. Cependant, la conception d'application interactive nécessite de faire des choix à propos de l'interaction. Bien que le modèle de tâches initial soit le même, les tâches peuvent être exécutées différemment en fonction des instruments d'interaction utilisés.

Pour permettre la génération à partir d'un modèle de tâches, le *Dygimes Framework* propose de lier les tâches atomiques avec des widgets. Ceci est très restrictif pour les utilisateurs. L'ajout d'un autre modèle d'interaction (comme la manipulation directe par exemple) est alors impossible. De plus, lorsque les applications utilisent le *drag and drop*, compléter ou modifier les modèles de tâches pour y inclure l'interaction devient impossible. Par exemple, pour ajouter un document en pièce jointe, l'utilisateur peut utiliser le *drag and drop* pour sélectionner, déplacer et positionner le document. Même si la tâche *ajouter un document* est une tâche atomique, elle est exécutée avec une succession d'actions interactives. Donc, les tâches atomiques sont composées de *prendre un fichier*, *déplacer un fichier* et *déposer un fichier*. SUIDT [Baron et Girard 2002] réalise cette étape pour obtenir des modèles de tâches concrets. Alors que cette approche permet de spécifier le dispositif utilisé pour réaliser les tâches (clavier, souris…) et les actions faites sur ces dispositifs (click…), elle n'est pas appropriée pour ajouter où les actions sont faites (sur un fichier, sur un champ du message…). Cette information peut être importante. Par exemple, quand un fichier sélectionné est déposé dans l'email, le document est ajouté alors que s'il est déposé n'importe où ailleurs, la tâche *ajouter document* est abandonnée. Ajouter cette information dans les modèles de tâches signifie insérer des variables et des données qui ne sont pas du niveau d'abstraction des modèles de tâches.

Aujourd'hui, de plus en plus d'applications utilisent de nouvelles techniques d'interaction, généralement regroupées sous l'appellation d'interaction *post-WIMP* afin de tirer parti des principes de la manipulation directe [Shneiderman 1983]. Suivre ces principes consiste dans le fait que les actions de l'utilisateur et les réponses du système sont très proches et en la suppression des intermédiaires comme les fenêtres de dialogue. L'ordre d'exécution des tâches peut être modifié en fonction des choix d'interaction et de dispositifs. Donc, certaines de tâches peuvent être ajoutées ou supprimées. Dans [Beaudouin-Lafon 2000], l'exemple montre une application pour éditer du texte. Pour remplacer un mot, un utilisateur indique quel est le mot qu'il souhaite modifier et avec quel autre mot, sans utiliser ni menu ni commande externe. Les mots sélectionnés sont surlignés et lorsqu'un mot a été modifié, la couleur change. Enfin, l'utilisateur peut facilement revenir en cliquant sur le mot modifié. En termes de modélisation de tâches, cela s'exprime comme suit. Une tâche de *retour en arrière* doit être ajoutée dans le modèle de tâche alors que les tâches qui décrivent les actions liées à l'utilisation de menu doivent être supprimées.

De plus, l'organisation temporelle des tâches est très proche du dialogue de l'application. Cependant, ajouter l'interaction peut modifier le dialogue lui-même. Par exemple, alors qu'un document peut être ajouté à l'email, l'utilisateur peut le supprimer ou le déplacer à une autre place dans l'email (Figure 3.3.4a). Ces deux tâches sont complètement différentes du point de vue du modèle de tâches. Cependant, un utilisant le *drag and drop* pour les exécuter (Figure 3.3.4b) le début de leur exécution est commun. L'utilisateur débute en cliquant sur un fichier et en le déplaçant. Après ces deux premières actions, il n'est pas possible de savoir quel est le but de l'utilisateur : supprimer ou déplacer le document. Le but apparaît lorsque le document est déposé : s'il est déposé à une autre place dans l'email, alors c'est le déplacement du document qui est réalisé, s'il est déposé hors de la fenêtre c'est la suppression. L'équivalence entre le modèle de tâches et le modèle de dialogue est alors cassée.

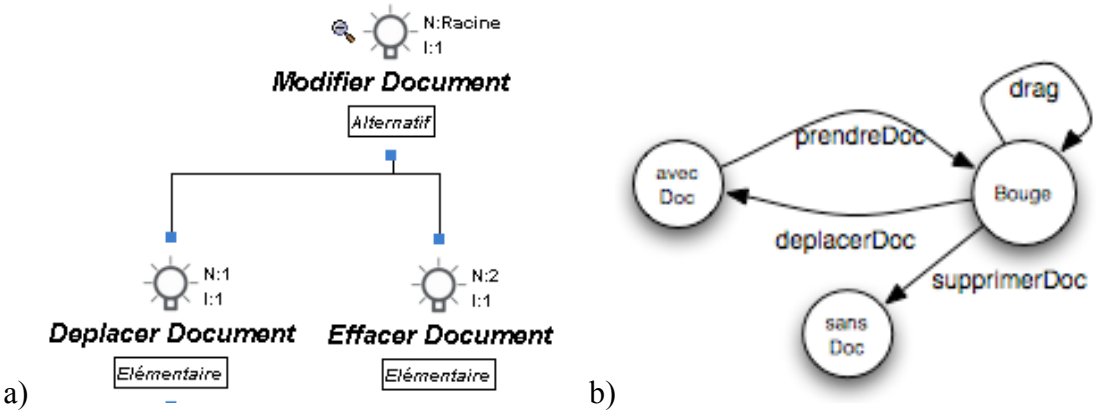


Figure 3.3.4 : Les différences entre a) la description des tâches et b) celle du dialogue pour le mouvement d'un document ajouté

3.3.5.2. Différents moyens de réaliser une tâche

Même si toutes les interactions pour une tâche donnée peuvent être représentées dans les modèles de tâches, comment peut-on indiquer quand une tâche peut être réalisée par deux différentes interactions ? Dans SUIDT, une tâche concrète est créée pour chaque interaction. Par exemple, si la tâche *Envoyer email* peut être réalisée soit en cliquant sur un bouton soit en utilisant un raccourci clavier (*Ctrl+S*), alors la tâche *Envoyer email* est raffinée en deux tâches concrètes liées par l'opérateur *alternatif*. Cette conception augmente le nombre de tâches concrètes. De plus, comme nous l'avons dit précédemment, l'interaction choisie peut modifier ou complètement supprimer une tâche. Enfin, les modifications d'un modèle de tâche peuvent être faites à un haut niveau et ne sont pas limitées à l'ajout de tâches concrètes.

3.3.6. Conclusion sur la génération

Générer les applications interactives à partir des modèles de tâches est un défi difficile. Certains travaux essaient d'y parvenir par dérivation, complète ou partielle, nécessitant des interventions humaines ou non. D'autres essaient d'ajouter aux modèles de tâches les données qui dépendent d'autres modèles (comme la présentation ou le dialogue). Dans cette section, nous avons étudié les différentes parties de la conception des applications interactives qui peuvent ou non être obtenues à partir des modèles de tâches. Nous avons étudié quatre points qui pourront constituer des axes de recherches futurs.

La partie **présentation** semblerait être la partie la plus rapide à obtenir à partir du modèle de tâches. Ajouter des items aux modèles de tâches pour définir la présentation ne semble pas être une bonne technique. Cependant, les modèles de tâches peuvent également être utilisés pour vérifier la correspondance entre les besoins utilisateur et la présentation.

Les aspects cognitifs de l'activité doivent être pris en compte. Le premier défi relevé par notre étude est donc : identifier comment relier la présentation et les aspects cognitifs.

À propos de la **gestion des erreurs de l'utilisateur**, nous avons illustré que les modèles de tâches peuvent participer à la prévention des erreurs, mais ne sont pas adaptés à la représentation de leur correction. Le deuxième défi consiste à étudier la possibilité de prendre en compte la gestion des erreurs, en collaboration avec les modèles de tâches.

La génération du **contrôle de dialogue** à partir du modèle de tâches semble être la partie la plus simple à obtenir par la génération. Les outils de simulation qui animent les modèles de tâches montrent que cela est possible. Cependant, notre étude montre que cela n'est pas si facile. Les modèles de tâches, même complexes, ne permettent pas de dériver le modèle de dialogue. Des transformations doivent être définies. Être capable de conserver le lien entre ces deux parties pendant les transformations constitue le troisième défi que nous avons identifié.

Enfin pour conclure, l'étude des **modèles d'interaction riches**, qui contiennent les interfaces post-WIMP, ouvre de nouvelles perspectives pour la transformation entre les modèles. De plus, l'ajout d'éléments d'interaction dans le modèle de tâches impose que le modèle de tâches soit composé de niveaux différents (les niveaux *tâches* et les niveaux d'*interaction*). Nous considérons alors que deux niveaux de modèles de tâches existent : le niveau dédié à l'expression de l'activité seule (spécification uniquement des tâches sans l'interaction) et un niveau dans lequel l'interaction est ajoutée (c'est ce dernier qui peut être utilisé comme modèle intermédiaire dans des travaux menés sur l'interaction lors de la conception comme ceux de Charfi [Charfi 2009]).

Bien qu'il semble impossible de générer complètement l'interface à partir des modèles de tâches, il est clair que certaines parties de l'interface sont liées aux données contenues dans le modèle de tâches. L'une d'entre elles, le contrôleur de dialogue, semble être très proche du modèle de tâches, comme le montrent les simulations des modèles de tâches. Même si la génération de tout le contrôle de dialogue à partir du modèle de tâches apparaît impossible, ce lien doit être exploité. C'est ce point que nous avons précisément choisi comme but de notre travail.

3.4. Conclusion des études préliminaires

Dans cette section, nous avons présenté les premières études de nos travaux sur la conception du dialogue des applications interactives en fonction des données exprimées sous forme de modèles de tâches. Ces premières études concernent trois pôles : les différentes notations de modèle de tâches, les formalismes de modèles de dialogue et la génération automatique des applications (incluant de facto le dialogue) à partir des modèles de tâches.

Cette dernière étude préliminaire montre que la génération automatique des applications à partir des modèles de tâches ne peut être complète bien que certains composants des applications soient directement issus des modèles de tâches. Les approches de génération s'appuient sur ces points communs en négligeant (ou comblant par des

valeurs arbitrairement fixées) les difficultés liées à l'absence de données non dépendantes des modèles de tâches.

De plus, il est à noter que la dérivation des applications interactives à partir des modèles de tâches est une génération qui ne permet pas de retour à l'étape précédente. Cette méthode de conception en *cascade* n'est pas adaptée à la conception itérative d'IHM.

Nous proposons une approche alternative pour concevoir et valider la partie dédiée au dialogue en fonction du modèle de tâches. Le reste de nos travaux se focalise sur la conception du dialogue des applications interactives, ne prenant pas en compte les difficultés liées à la conception de la partie présentation, au contraire des trois autres limitations que nous avons identifié : la gestion des erreurs humaines, le contrôle de dialogue et la possibilité d'intégrer les techniques d'interaction riches (voir 3.3.6).

Pour cela, nous avons identifié les modèles de tâches (§ 3.1) et de dialogue (§ 3.2) que nous utiliserons dans la suite. Notre choix de modèle de tâches s'est porté sur le modèle de tâches exploitant le plus de données calculables, dans des expressions formelles : K-MAD. Pour le formalisme utilisé pour exprimer le dialogue, nous avons choisi d'utiliser le formalisme des interacteurs hiérarchisés qui nous semble être un formalisme adapté à nos besoins.

L'approche que nous privilégions consiste à travailler sur le méta-modèle des deux formalismes choisis (Chapitres 4 et 5) afin d'établir des règles de correspondance et de transformation entre ces modèles (Chapitre 6).

Chapitre 4. Méta-modélisation de K-MAD (v2)

L'étude préliminaire sur les modèles de tâches que nous avons exposée dans la section 3.1, a mené à choisir K-MAD comme formalisme pour exprimer les modèles de tâches dans notre étude. Nous avons montré dans l'évaluation de l'utilisation de K-MADe que l'utilisation des objets pour compléter l'ordonnancement exprimé par les opérateurs est naturelle pour les concepteurs. L'ordonnancement exprimé dans les modèles de tâches est le principal point exploité pour la conception du dialogue. Cependant, les évaluations ont également mis en lumière que la définition et l'utilisation des objets via K-MADe n'est pas intuitif pour les concepteurs.

Tout au long de nos travaux (études théoriques et évaluations empiriques), nous avons utilisé K-MAD à l'aide de l'outil K-MADe pour réaliser des modélisations de différentes activités. Ces modélisations (que nous présentons dans la section 4.1) nous ont amené à identifier les limitations nécessitant d'apporter des modifications au formalisme K-MAD (section 4.2) pour accroître le pouvoir d'expression de l'ordonnancement des modèles de tâches (caractéristiques exploitées pour la conception et la validation du dialogue). La méta-modélisation des composants de cette nouvelle version du noyau de K-MAD, que nous nommerons dans la suite K-MAD(v2), seront présentés tout au long de la section 4.2. EXPRESS est un langage de modélisation intégrant l'expression de conditions formelles remplissant le même rôle que le langage UML associé au langage de contraintes OCL. Cependant, au contraire d'UML/OCL, EXPRESS est un langage destiné à l'expression uniquement des données. Il correspond donc à nos besoins en matière d'expressivité pour la suite de nos travaux. Les méta-modèles de K-MAD(v2), tout comme toutes les autres méta-modélisations réalisées dans cette thèse sont exprimées à l'aide du langage EXPRESS (voir Annexe 3).

Enfin, dans la dernière section de ce chapitre (section 4.3) nous décrirons les transformations qui sont nécessaires pour *retranscrire* des modèles de K-MAD(v1) en modèles K-MADe(v2).

Dans l'ensemble de ces travaux, chaque fois que nous nous rapporterons au modèle K-MAD (ou K-MAD(v1)), nous ferons référence au modèle implémenté dans l'outil K-MADe à partir du modèle exprimé dans la thèse de Lucquiaud [Lucquiaud 2005b].

4.1. Études de cas

Tout au long de nos travaux, nous avons modélisé différents cas. Ces études de cas nous ont permis d'appréhender l'utilisation du formalisme K-MAD, de détecter les éléments nécessaires au modèle et de déterminer les modifications à apporter. Ces cas d'étude seront tout d'abord succinctement décrits (§ 4.1.1) puis nous détaillerons les leçons apprises de leur modélisation (§ 4.1.2). Nous les avons détaillées en annexe (Chapitre 9).

4.1.1. Présentation des études de cas

Nous avons utilisé K-MADe pour valider une application (*ParAdmin*) et pour en concevoir quatre autres. Le choix de ces études de cas nous a permis de concevoir des modèles de tâches dans des contextes différents. Les trois premiers (le *mailer*, *la tenue d'une feuille de match de Volley-ball* et le *jeu de Mastermind*) ont été conçus dans un but d'entraînement à la modélisation K-MAD. Le quatrième, le cas de l'application *ParAdmin*, a été conçu dans un contexte de recherche, avec des évolutions fréquentes. Enfin, le cas de *Genindexe* a été conçu pour produire une application opérationnelle (actuellement utilisée par *Genindexe*[12]). Les trois premières lignes du Tableau 4-1 résument les caractéristiques de chacune de ces études de cas.

De plus, les modèles de tâches de ces études de cas ont été conçus par des concepteurs différents de par leur niveau d'expertise dans la modélisation des tâches, leur connaissance du domaine et leur nombre. Dans un second temps, nous préciserons pour chacune ses caractéristiques de conception. Elles sont résumées dans les deux dernières lignes du Tableau 4-1.

	Mailer	ParAdmin	Feuille de match de Volley-ball	Mastermind	Genindexe
Nombre d'utilisateurs	1	1	1	1 ou 2	Tous les employés
Type de plateforme	ordinateur	ordinateur	tablette	ordinateur	ordinateur
Nombre de plateformes	1	1	1	1	4
But de l'utilisation de K-MAD	Conception	Validation	Conception	Conception	Conception
Nombre de concepteurs	2	1 (avec le développeur)	68	2	48

Tableau 4-1 : Les caractéristiques des études de cas

Enfin, nous ferons un bilan de la conception spécifique de chacun des cas présenté.

Description

Mailer. La première application est une application classique de traitement d'emails. Sa conception est basée sur un modèle de tâches exprimant toutes les activités permises par la communication par email (gestion d'un carnet d'adresses, production de courriers électroniques, gestion des emails reçus…). C'est un des deux cas d'étude qui sont (partiellement) utilisés pour illustrer nos propos dans la suite de cette thèse.

[12] http://www.genindexe.com/

ParAdmin. Afin d'étudier le rôle des modèles de tâches K-MAD lors de la validation d'une application, nous avons modélisé la gestion d'un entrepôt de données par un administrateur. Un outil, nommé *ParAdmin* [Boukhalfa, et al. 2008], a été développé pour réaliser cette tâche. Dans un premier temps, son développement n'a pas été réalisé à partir d'une analyse de tâches. Puis, dans un second temps, afin de faciliter l'utilisation de cet outil par de nombreux administrateurs d'entrepôt de données, les modèles de tâches de l'activité ont été utilisés pour proposer des modifications à l'application.

La tenue de la feuille de match de Volley-ball. Cette activité étudiée qui ne fait intervenir qu'un seul utilisateur est la tenue d'une feuille de match de volley-ball (pendant le jeu). Cette activité est traditionnellement réalisée en utilisant une feuille papier (voir annexe Chapitre 9). Le but de la conception du modèle de cette activité est de migrer sur un support constitué d'une tablette informatisée et également de permettre la réalisation automatique de certains calculs et de certaines vérifications (par exemple, à partir des données entrées évaluées si la partie est terminée ou non).

Mastermind. La conception du jeu de Mastermind montre les besoins d'une application dans laquelle plusieurs utilisateurs interagissent simultanément. Ce jeu a été développé pour être utilisé sur un même ordinateur par un ou deux joueurs. Ce cas d'étude est le second utilisé pour illustrer nos propos dans la suite de cette thèse.

Genindexe. La dernière application permet la gestion des activités d'un laboratoire d'analyse animal[11]. Un modèle de tâches a été conçu pour créer une application adaptée aux activités de tous les employés tout en respectant les besoins de sécurité et de qualité du laboratoire.

Conception

Mailer. Les modèles de tâches conçus pour l'étude de cas du *mailer* ont été utilisés pour servir de support à la conception d'une application de gestion d'emails. Le processus de conception pour cette étude de cas a été réalisé par deux fois. Deux modèles de tâches supportant l'activité de gestion d'emails ont donc été conçus. Ces conceptions ont été réalisées par deux concepteurs différents, de niveau d'expertise en modélisation des tâches différents (l'un de niveau moyen, l'autre expert).

ParAdmin. La conception de modèles de tâches modélisant l'activité d'administration d'un entrepôt de données avait pour but de modifier l'interface d'une application existante (*ParAdmin(v1)*) pour faciliter son utilisation par des administrateurs n'étant pas intervenus dans sa conception. Ces modèles ont été conçus par un unique concepteur (expert en IHM) à partir d'interviews régulières du concepteur/développeur de *ParAdmin(v1)*.

La tenue de la feuille de match de Volley-ball. À partir de la description de l'activité de tenue de feuille de marquage par un *marqueur* de Volley-ball (sous forme de règles écrites et d'exemples de feuilles vierges et complétées (voir annexe Chapitre 9), 68 étudiants ont eu à concevoir le modèle de tâches de cette activité réalisée sur un support informatisé (une tablette interactive). Le but fut de proposer une modélisation qui permette de réaliser l'activité décrite et pouvant servir de support à la conception d'une application facilitant son déroulement (en ne proposant par exemple à l'édition que les champs

pouvant être complétés à chaque instant). Cette modélisation a été utilisée pour notre évaluation de l'apprentissage et de l'utilisation de K-MADe. Les étudiants ayant produit les 68 modèles de tâches étaient tous à la fin de leur apprentissage en modélisation des tâches (dispensé dans le cours d'IHM). Les caractéristiques de ces sujets sont plus longuement détaillées dans la section 3.1.3.1.

Mastermind. Deux modèles de tâches décrivant l'activité *jouer au Mastermind* ont été réalisés par deux concepteurs différents (les deux mêmes concepteurs que pour l'étude de cas du *mailer*). L'un des deux modèles conçus a été utilisé pour servir de support à la conception de plusieurs interfaces permettant le déroulement de ce jeu (nous avons choisi la modélisation réalisée par le concepteur de niveau d'expertise le plus élevé). Les différentes versions d'interface de ce jeu ont été développées à partir du même modèle de tâches avec des techniques d'interaction différentes, ou une présentation différente. Par exemple, la Figure 4.1.1 présente différentes interfaces du jeu de *Mastermind* réalisées par des développeurs différents (le concepteur du modèle de tâches et des développeurs à qui le modèle a été fourni) à partir du même modèle de tâches.

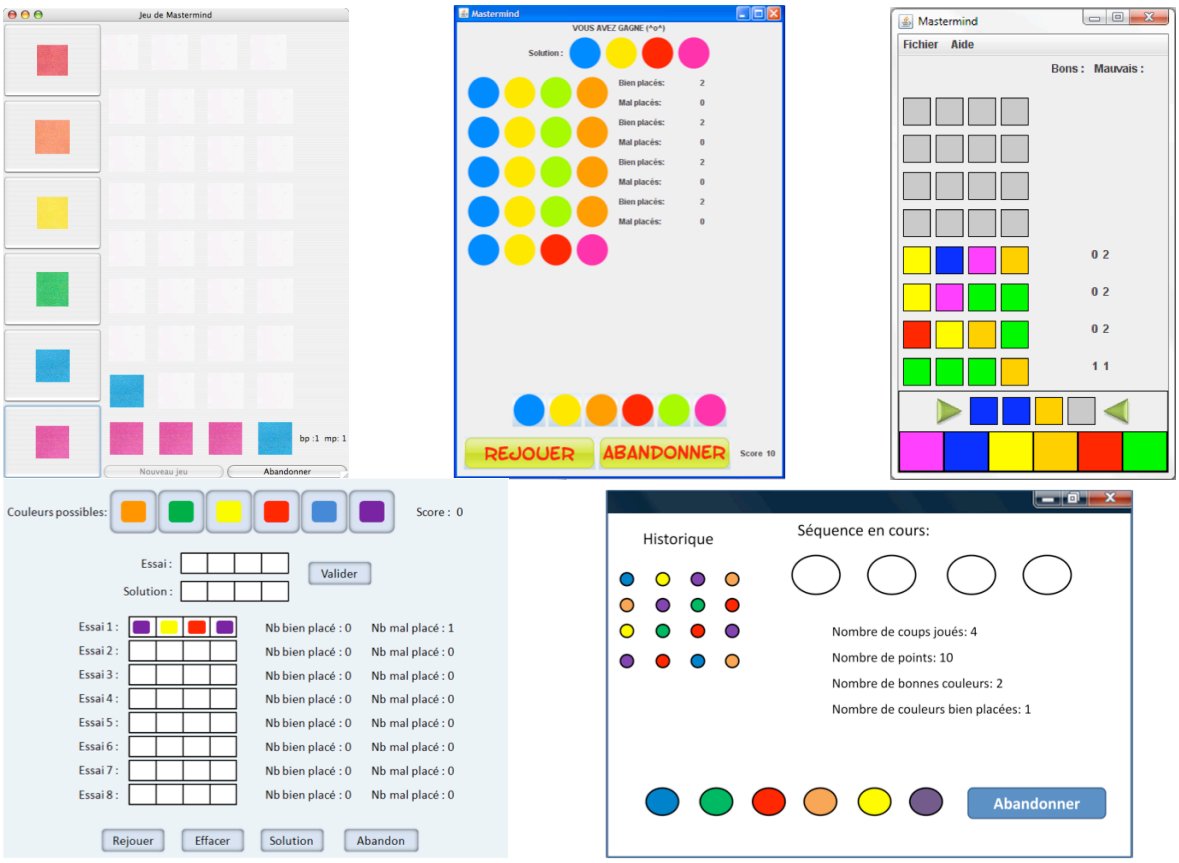


Figure 4.1.1 : Différentes présentations du jeu de Mastermind

Genindexe. Les modèles décrivant l'ensemble des activités menées dans un laboratoire d'analyses animales (*Genindexe*) ont été réalisés par 48 étudiants ayant terminé leur apprentissage sur la modélisation des tâches. Après vérification par des experts, ces

modèles ont servi de support au développement d'applications pouvant être utilisées dans le laboratoire (l'une d'elles l'est actuellement).

Bilan des conceptions

La conception des études de cas nous a permis d'observer la modélisation des tâches. Celle-ci fait ressortir des besoins spécifiques. Nous allons ici faire un bilan de la conception de chacune des études de cas. Certains points ont été relevés dans plusieurs des cinq applications, nous ne les décrirons que pour la conception de leur première itération. Le Tableau 4-2 indique pour chaque application les besoins rencontrés.

Mailer. Les modèles de tâches de gestion d'emails ont été conçus en plusieurs modules. Chaque module représente une activité de gestion d'emails (traitement des emails reçus, envoi d'emails…). Ces différents modules ont été définis séparément et leur conception a été espacée de plusieurs semaines (parfois plusieurs mois), en fonction de nos besoins. Ce mode de conception (dû à l'aspect itératif de la modélisation) permet d'illustrer les besoins liés à l'intégration de modules différents d'une même activité. Ces modules peuvent être modélisés dans des dossiers différents (stockés dans des espaces différents), leur liaison doit alors être faite par le concepteur (qui, le plus souvent doit recopier le modèle entier). L'intégration de différents modèles de tâches comme un seul nécessite la définition d'un modèle *intégrateur* constitué d'une tâche d'un niveau d'abstraction supérieur décomposée des modèles-modules définis [Sinning, et al. 2007]. Le cas de l'intégration des modules dédiés à la gestion de l'envoi d'emails et à celle de la gestion des emails conçus peut être représenté comme sur la Figure 4.1.2.

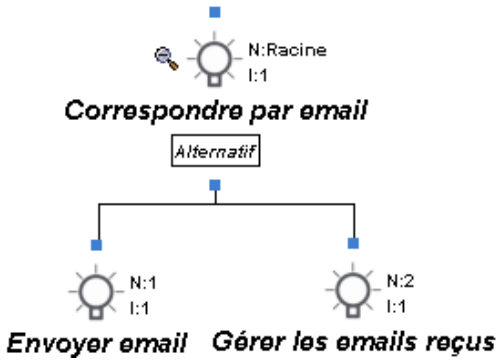


Figure 4.1.2 : Modèle intégrant deux modèles de tâches, le modèle *Envoyer email* et le modèle *Gérer les emails reçus*

Le concepteur (le même pour tous les modules) doit alors reprendre un modèle précédemment réalisé. Pour faciliter la prise en main d'un modèle conçu, les concepteurs cherchent à le comprendre le plus rapidement possible. Pour cela, les caractéristiques des tâches doivent être rapidement comprises ; par exemple, le nom de la tâche doit être explicite.

Dans le cas de la conception du *mailer*, les modèles de tâches furent repris par les concepteurs eux-mêmes, la compréhension du modèle conçu en fût de ce fait grandement

facilité. Malgré cela, dans K-MAD, des difficultés ont été relevées sur la compréhension des conditions décrites uniquement de manière formelle. Les expressions peuvent être formellement et textuellement décrites (Figure 3.1.15). Cependant, si les concepteurs décrivent formellement les conditions pour pouvoir rendre leurs modèles dynamiques, une fois qu'ils savent éditer les expressions formelles, ils négligent leur expression textuelle. De plus, l'utilisation de la vue dynamique (par l'outil de simulation) ne comble pas ce besoin car, lors de la simulation du modèle, l'utilisateur doit saisir des valeurs pour l'évaluation des expressions formelles sans qu'aucune information ne lui soit transmise sur la sémantique de l'expression (voir exemple de la post-condition sur la Figure 4.1.3).

ParAdmin. Les modèles de tâches utilisés pour modifier l'interface de *ParAdmin* ont été réalisés conjointement par un spécialiste de l'IHM (le concepteur du modèle) et un spécialiste du domaine d'application (le concepteur et développeur de *ParAdmin*). La proximité entre ces deux collaborateurs a permis la mise à jour régulière du modèle de tâche pour y intégrer les modifications dues à l'avancée des travaux de recherche (sur la gestion des entrepôts de données). Pour la compréhension de l'activité du domaine d'application par le spécialiste d'IHM, la sémantique ajoutée par les objets de K-MAD a joué un rôle important. Cette compréhension est d'autant plus importante qu'elle permet la détection d'erreurs de conception. Par exemple, dans le cas de *ParAdmin*, l'entrepôt de données peut être trié par trois algorithmes différents. Cependant, la tâche de choix du type d'algorithme n'était pas nécessaire à la réalisation de l'activité (tâche optionnelle). En utilisant l'outil de simulation de K-MADe (qui manipule dynamiquement les objets), nous ne pouvions pas définir de scénario utilisant cette donnée lorsqu'elle n'était pas entrée. La détection de cette erreur de conception a donc été facilitée.

Enfin, la compréhension du modèle réalisé par l'expert du domaine d'application a été facilitée par l'utilisation de l'outil de simulation de K-MADe. L'exploitation des entités formelles (et des expressions) lors de la simulation des modèles de tâches fournit une **représentation complémentaire** de l'arbre de tâches. À partir des études de cas *Genindexe* et *ParAdmin*, nous avons observé que cette représentation de l'état du monde est très proche de la représentation qu'a l'utilisateur de l'application finale. Cet outil est donc un outil indispensable dans une conception centrée-utilisateur pour laquelle l'utilisateur est intégré dans la conception [Caffiau, et al. 2008b]. Cependant la représentation de l'état du monde, dans l'outil de simulation (partie encadrée sur la Figure 4.1.3), est difficile à comprendre pour les utilisateurs de l'outil qui ne sont pas les concepteurs.

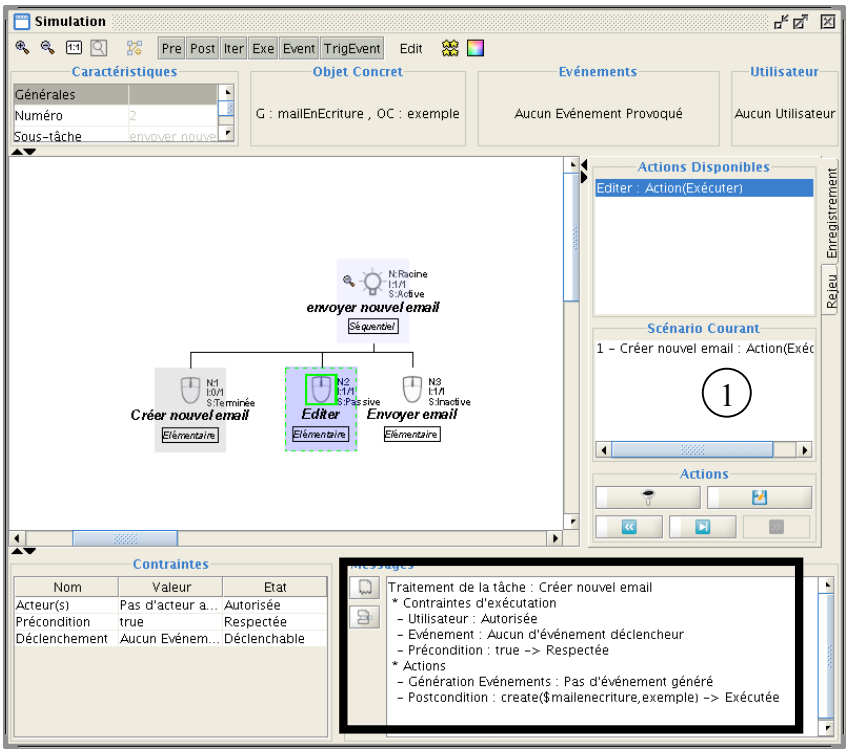


Figure 4.1.3 : L'outil de simulation de K-MADe

La tenue de la feuille de match de Volley-ball. Les modèles de tâches réalisés pour cette étude de cas ont permis d'utiliser K-MAD pour détecter les tâches pouvant être automatisées. Par exemple, la tâche de calcul pour déterminer la fin d'une partie peut être automatiquement réalisée par le système. Ce type de tâche a été défini dans K-MADe par des expressions formelles (dans les conditions) utilisant les objets définis.

Mastermind. La modélisation de l'activité *jouer au Mastermind* nous a permis d'étudier l'utilisation d'un même modèle de tâche comme support à la conception de plusieurs interfaces différentes. À partir de cette modélisation, nous avons déterminé un niveau de décomposition des tâches indépendant de l'interaction utilisée pour son accomplissement. C'est à ce niveau que nous avons choisi d'arrêter la concrétisation des modèles de tâches (pour toutes nos activités). Par exemple, pour modéliser le déplacement d'un pion dans une combinaison, le moyen utilisé ne sera pas décrit, les décompositions présentées au plus bas niveau sur la Figure 4.1.4 ne seront pas précisées dans le modèle, seule la tâche *déplacer Pion* sera indiquée. Précisons que d'autres travaux ont choisi d'utiliser un niveau de détail différent comme Charfi dans [Charfi 2009] qui va plus bas dans la décomposition des tâches pour relier un modèle K-MAD avec un modèle d'interaction (AZUR [Gauffre 2009]).

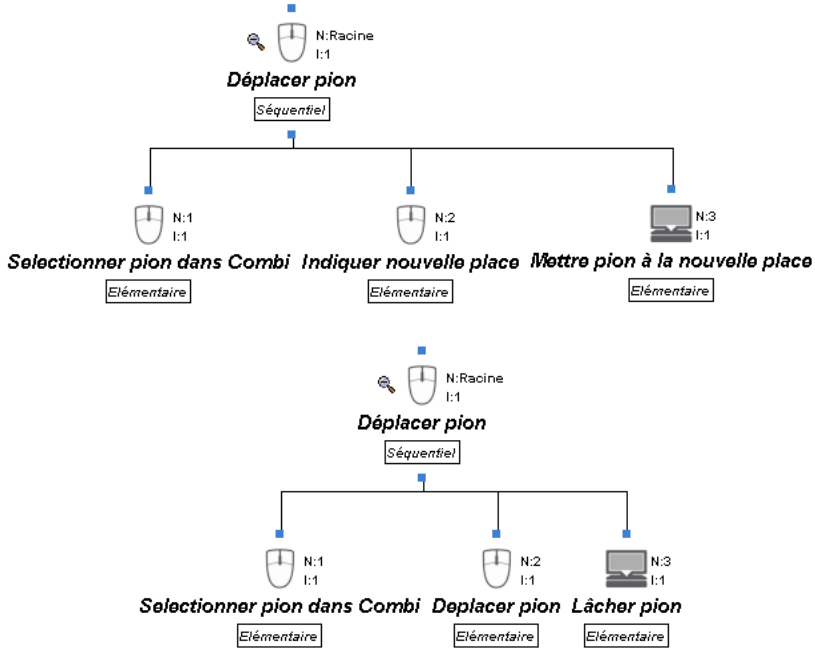


Figure 4.1.4 : Deux modélisations différentes de *déplacer Pion*

Genindexe. Les modèles de l'activité de *Genindexe* ont deux caractéristiques spécifiques : cette étude de cas est la plus importante réalisée, et est celle dont les experts du domaine d'application ont des connaissances les plus éloignées de l'IHM (l'expert du domaine d'application de *ParAdmin* ayant des connaissances en informatique et le concepteur du modèle de tâches, des connaissances sur la gestion des données). Elle nous a donc permis de mettre à l'épreuve K-MAD pour la communication entre experts du domaine différents (qui est l'un des rôles des modèles de tâches [Balbo, et al. 2004]).

L'activité du laboratoire fait intervenir différents individus ayant des compétences et des accréditations différentes. Les composants *Utilisateurs* du modèle K-MAD ont donc été utilisés afin de gérer les droits d'accomplir les tâches pour lesquelles ils étaient définis comme acteur. Cette association entre tâches *utilisateur* permet d'identifier l'expert du domaine concerné par l'activité lors de la validation avec l'outil de simulation. Tout comme pour le cas de *ParAdmin*, le besoin d'une vue « non concepteur » du modèle simulé s'est fait ressentir.

En plus de ces vues pour la simulation, nous avons constaté qu'une même activité peut être représentée en suivant plusieurs points de vue. Ainsi, l'activité peut être faite en suivant le parcours d'un échantillon dans le laboratoire, le rôle dans le laboratoire ou le déroulement d'une journée. Si lors de la décomposition de l'activité des concepteurs ont commencé à suivre une modélisation du parcours d'un échantillon, dès qu'ils ont intégré les objets dans le modèle, ils ont adopté le point de vue de l'activité de chacun des rôles en définissant les échantillons comme des objets manipulés. Cette modélisation montre comment l'intégration des objets dans les modèles de tâches facilite l'expression

d'activités dans lesquelles un élément est au centre des attentions tout en conservant une modélisation centrée sur l'activité humaine.

	Mailer	ParAdmin	Feuille de match de Volley-ball	Mastermind	Genindexe
gestion de modèles composés	√	√	-	-	√
prise en main rapide d'un modèle conçu	√	√	-	√	
utilisation des objets pour la compréhension du domaine d'application	-	√	-	-	√
simulation avec vue pour non IHM	-	√	-	-	√
détection de tâches pouvant être automatisées	-	-	√	√	√
modélisation indépendante des techniques d'interaction	-	-	-	√	-
plusieurs vues du modèle	-	-	-	-	√

Tableau 4-2 : Synthèse des points observés dans les études de cas

4.1.2. Leçons apprises de la réalisation des études de cas

La modélisation des études de cas et la conception des applications à partir des modèles de tâches réalisés (seule la modélisation de la tenue de la feuille de match de Volley-Ball n'a pas abouti au développement d'une application) nous a permis d'étudier les apports et les limitations de l'expression des modèles de tâches pour la conception avec K-MADe. Nous allons, dans cette section, faire un bilan de ces constats. Parmi les points concernés, certains sont liés au modèle K-MAD et d'autre à l'utilisation de l'outil K-MADe. Considérant le modèle utilisé via l'outil nous les présenterons tous dans la section 4.1.2.1.

En complément de nos travaux sur l'expressivité des modèles K-MAD pour la conception, une autre étude [Sinning, et al. 2007] a proposé des extensions pour palier les limitations à l'utilisation de modèles CTT comme support à la conception d'applications interactives. Lors de la réalisation des modélisations, nous avons particulièrement observé

comment K-MADe répond à ces limitations. Nous présenterons le résultat de cette étude spécifique dans la section 4.1.2.2.

4.1.2.1. Principaux résultats

Les modélisations réalisées expriment les activités de différents utilisateurs, différents systèmes et sont choisies dans le but de concevoir ou de valider différentes applications interactives. La conception de cette large variété d'applications montre les bénéfices de l'utilisation du modèle K-MAD par l'outil K-MADe par rapport à l'utilisation d'autres modèles. Ces bénéfices sont apportés par des éléments de K-MAD qu'il est donc important de conserver dans de nouvelles versions de ce modèle. Parallèlement à ces avantages, des limitations à l'utilisation de K-MAD/K-MADe ont été relevées. Celles-ci seront l'objet d'étude supplémentaire pour être supprimées (étude présentée dans 4.2).

Le premier bénéfice identifié est commun à toutes les modélisations réalisées et a déjà été illustré lors de nos évaluations menées sur l'utilisation des composants formels de K-MADe (§ 3.1.3), c'est l'utilisation de ces composants pour **compléter la spécification temporelle des tâches** exprimée par les opérateurs de décomposition. Dans la majorité des outils de modélisation, la spécification temporelle des tâches n'est exprimée que par les opérateurs de décomposition (§ 3.1.2.1). Malheureusement, ces opérateurs ne sont pas suffisants pour exprimer toutes les activités.

Par exemple, la conception du modèle de tâches relatif au remplissage d'une feuille de match de volley-ball nécessite l'utilisation des objets pour exprimer la fin d'un jeu. Un jeu de volley-ball se termine lorsqu'une des deux équipes a gagné trois sets, quel que soit le nombre de sets gagnés par l'autre équipe. Donc, le nombre de sets joués n'est pas déterminé par avance mais dépend du déroulement de la partie. Dans CTT (qui est une notation largement utilisée pour exprimer les modèles de tâches pour la conception), cette condition de terminaison n'est pas exprimable. Le déroulement d'un set (et d'une partie) est exprimé comme étend itératif jusqu'à ce que l'arbitre déclare la fin du set (ou de la partie). Dans le modèle CTT (représenté sur la Figure 4.1.5), rien ne permet à un individu non expert du domaine d'application de savoir que la fin d'une partie dépend des scores et non pas du bon vouloir de l'arbitre ou de la durée de la partie (comme cela serait le cas pour un match de football par exemple).

Au contraire, dans K-MAD, pour exprimer cette condition d'interaction de jeu nous avons utilisé des expressions formelles d'itération manipulant des objets définis formellement. Ces expressions précisent la sémantique de l'itération (nul doute sur la condition de fin d'un match) et peuvent être utilisés dans un but de conception pour détecter par exemple, les tâches pouvant être automatiquement réalisées (par le système conçu).

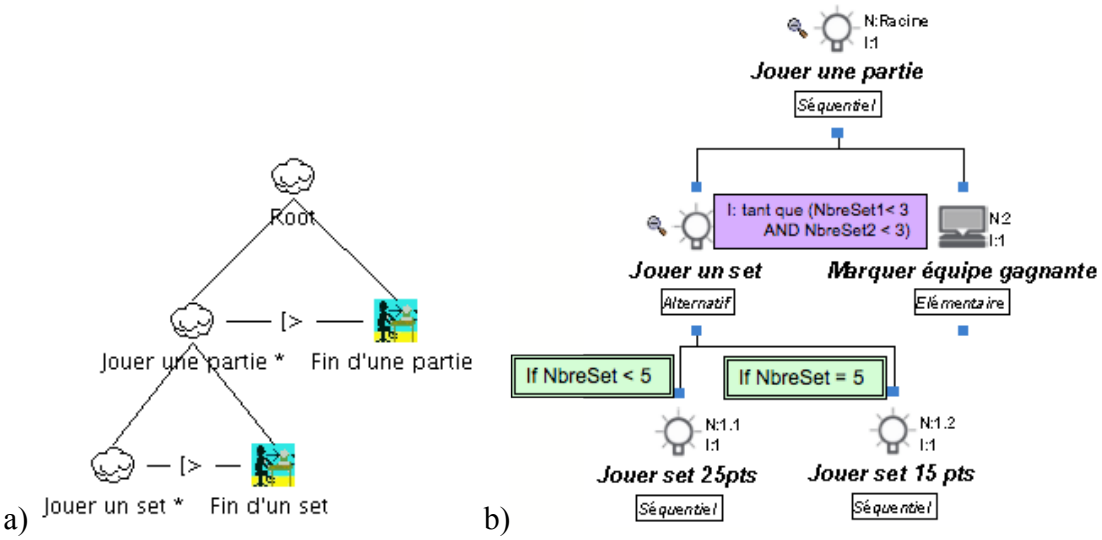


Figure 4.1.5 : Modélisation de la fin d'un set avec a) CTT et b) K-MAD

En plus de ce rôle temporel, les entités formelles évaluées peuvent **aider à la validation** du modèle de tâches. Notre étude sur l'utilisation de K-MADe (étude présentée dans 3.1.3) montre que ces aspects formels sont particulièrement exploités dans deux des outils de K-MADe.

L'outil de vérification grammaticale (Figure 3.1.8) permet de rapidement détecter les erreurs de grammaire (comme une erreur de syntaxe lors de l'édition d'une pré-condition) et l'outil de simulation permet de réaliser des scénarios en intégrant les entités formelles représentant les entités formelles (comme la réalisation d'un scénario en fonction de l'acteur) (marque 1 sur Figure 4.1.3).

L'utilisation du premier outil (la vérification grammaticale) a pour but principal la validation syntaxique du modèle afin de permettre son évaluation dans le second outil (l'outil de simulation). La simulation dynamique du modèle s'est montrée très utile pour les experts du domaine (particulièrement lorsqu'ils ne sont pas des experts en modélisation des tâches) notamment grâce à la modification dynamique des valeurs de objets. En effet, les changements de valeur des objets permettent à l'expert du domaine d'application de concrétiser le résultat du déroulement d'un scénario.

Cependant, lors de la modélisation de nos cas d'étude, nous avons relevé des besoins en expressivité du modèle K-MAD. Ces limitations d'expressivité portent sur certains attributs des tâches, sur la définition et l'utilisation des objets, sur les utilisateurs et les machines intervenant sans l'activité. Ainsi lors de la caractérisation des tâches des modèles K-MAD, la sémantique des attributs d'interruptibilité et la post-condition ont posé des difficultés aux concepteurs.

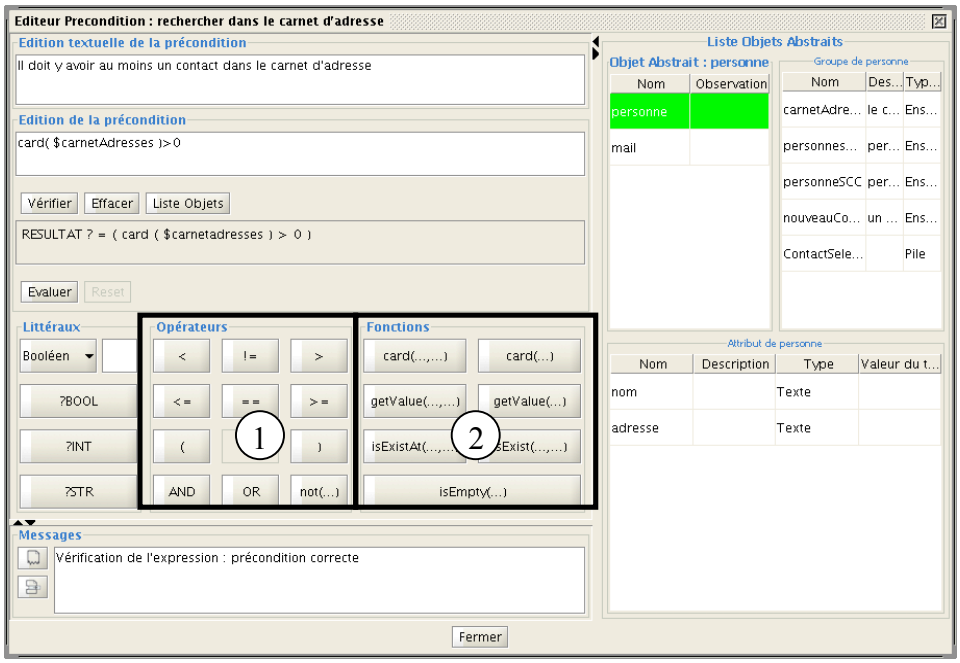
L'**interruptibilité** dans K-MAD est spécifiée comme étant une caractéristique de la tâche : la tâche est interruptible ou non. Cette attribution à la tâche ne permet pas d'indiquer par quoi la tâche peut être interrompue (est-ce par une autre tâche, comme dans CTT ? est-ce par un événement ?). De plus, elle n'indique pas si cette interruption est

définitive (annulation) ou s'il s'agit d'une suspension temporaire. Parmi l'ensemble des outils de modélisation étudiés, seul CTTE permet clairement l'expression de l'abandon des tâches en utilisant l'opérateur de désactivation (comme sur la Figure 4.1.5 pour signifier la fin d'une partie). Mais, quelles sont les conséquences de l'utilisation de cette action sur l'état du monde ? Est-ce que l'abandon d'une tâche entraîne le retour de l'état du monde à l'état précédant le déclenchement de la tâche ou reste-t-il dans son état courant ? CTTE ne prend pas en compte l'état du monde lorsque cet opérateur est utilisé et donc, ne fournit pas de réponses à ces questions. Au contraire, avant de permettre l'abandon des tâches dans K-MADe, il est impératif de déterminer leur conséquence sur l'état du monde.

La seconde difficulté sémantique survenue lors de la définition des tâches concerne la définition de la **post-condition** de la tâche. Dans K-MADe, la post-condition d'une tâche est l'action sur les objets. Or, cette définition n'est pas la même que celle utilisée en génie logiciel qui est une *condition booléenne devant être vérifiée après exécution de la tâche*. C'est cette définition de post-condition que nos concepteurs utilisent habituellement. Cette différence terminologique a rendu difficile la définition des premières expressions de post-conditions par de nouveaux utilisateurs de K-MADe.

Les expressions de conditions (pre, post et conditions d'itération) sont définies grâce à l'utilisation de calculettes (Figure 4.1.6). Ces calculettes sont composées de plusieurs parties. Deux de ces parties composent la définition des opérations : la partie constituée des *opérateurs* (marque 1 sur la Figure 4.1.6) et les parties constituées des *fonctions* (marque 2 sur la Figure 4.1.6). Ces parties représentent toutes les opérations disponibles sur les objets.

a)



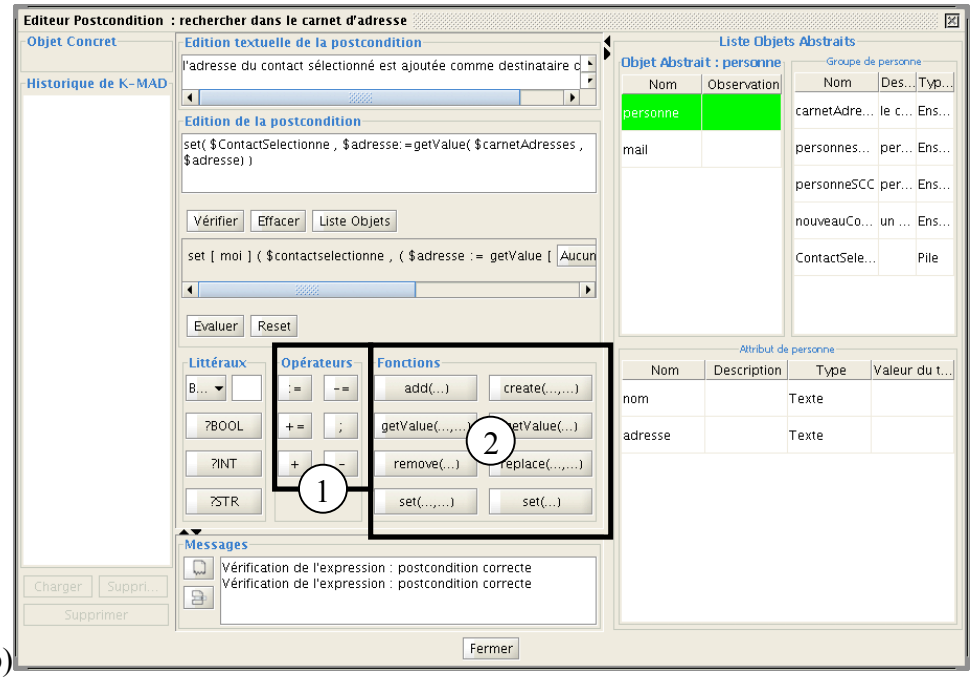
b) 

Figure 4.1.6 : Les calculettes de a) pré et b) post-conditions

Cependant, lors de l'édition des expressions, les concepteurs ont parfois eu des difficultés car les **opérateurs disponibles** ne leur suffisent pas. Par exemple, pour exprimer la condition d'itération un point de set, le concepteur n'a pas à disposition les opérateurs arithmétiques (Figure 4.1.7), il ne peut donc pas exprimer dans la condition d'itération que la différence du nombre de points marqués par les deux équipes pendant un set doit être supérieur à 2 une fois le nombre de points minimum atteint. Dans le cas du jeu du premier set, la condition d'itération s'exprimer formellement comme ceci : *tant que ((score1 <25 OR score1-score2 <2) AND (score2 <25 OR score2-score1 <2))*.

Figure 4.1.7 : Les opérateurs disponibles pour l'édition des expressions d'itération

Les opérateurs arithmétiques étant disponibles pour éditer les post-conditions, ils expriment une tâche système dans le but de calculer si le set est terminé et c'est cette valeur qui sera testée dans la condition d'itération du jeu des points (comme le montre la Figure 4.1.8 qui complète la Figure 4.1.5b).

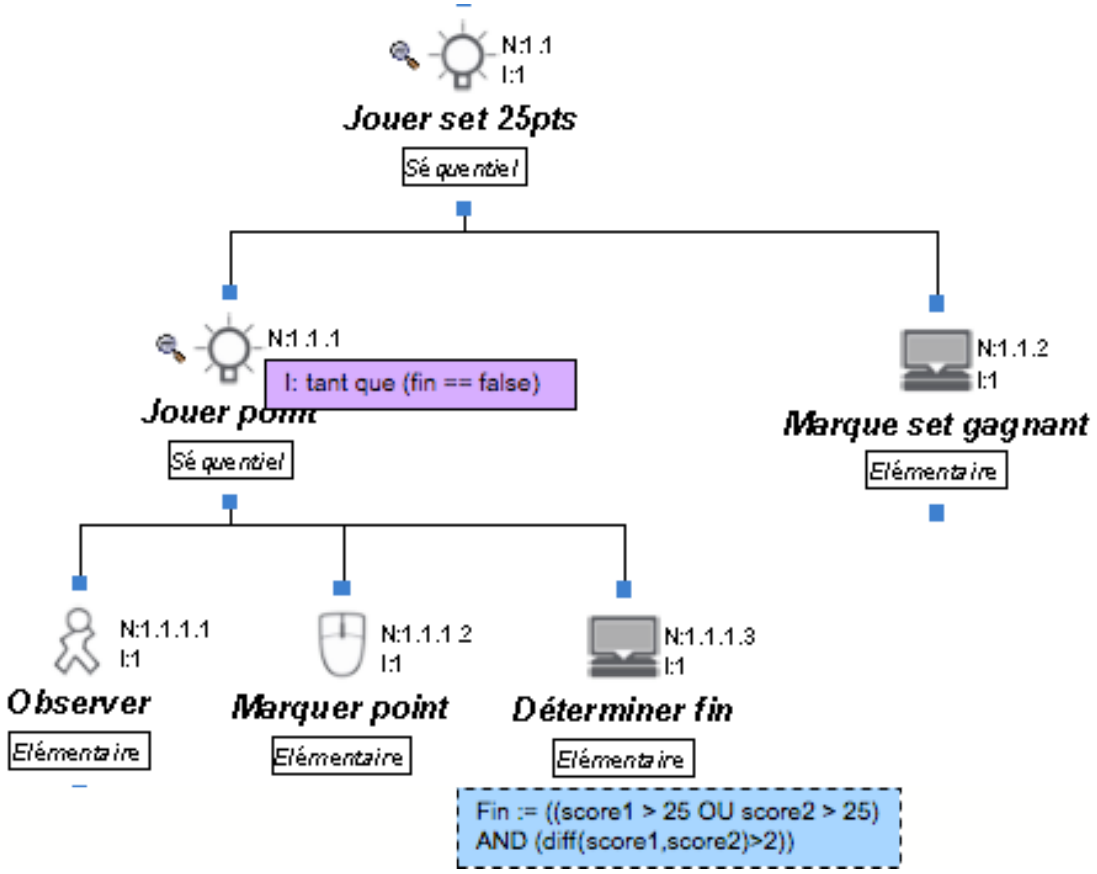


Figure 4.1.8 : Modèle partiel d'une partie de Volley-ball

Dans K-MADe, les **conditions d'itération** exprimées sont évaluées en fonction des valeurs des objets, elles ne peuvent donc pas être **définies comme étant conditionnées par le souhait d'un acteur**. Par exemple, l'itération de la tâche de positionnement d'un pion dans une combinaison ne peut pas être exprimée comme étant de la volonté du joueur.

Comme nous l'avons montré précédemment, les objets formellement définis sont bénéfiques pour la modélisation des tâches dans le cadre de la conception centrée-utilisateur. Afin de faciliter leur définition (par des utilisateurs de domaine de compétences différentes), des choix ont été faits sur les types d'attributs disponibles et la gestion des entités (la définition des groupes). Ces choix ont limité le pouvoir expressif des objets pour les concepteurs de nos études (rappelons que tous nos concepteurs ont des compétences en informatique). Par exemple, la **représentation de tous les attributs numériques par des entiers** n'a pas permis d'exprimer les valeurs de certaines analyses comportant des décimales, bien que ces données aient une sémantique pour les employés du laboratoire.

De même, la définition des attributs comme étant de type prédéfini ne permet pas de modéliser par le concepteur par exemple une adresse postale comme étant un attribut d'un objet *FichePersonnel*, chacun des composants de l'adresse postale devant être indépendamment intégré à la définition d'une fiche du personnel (Figure 4.1.9).

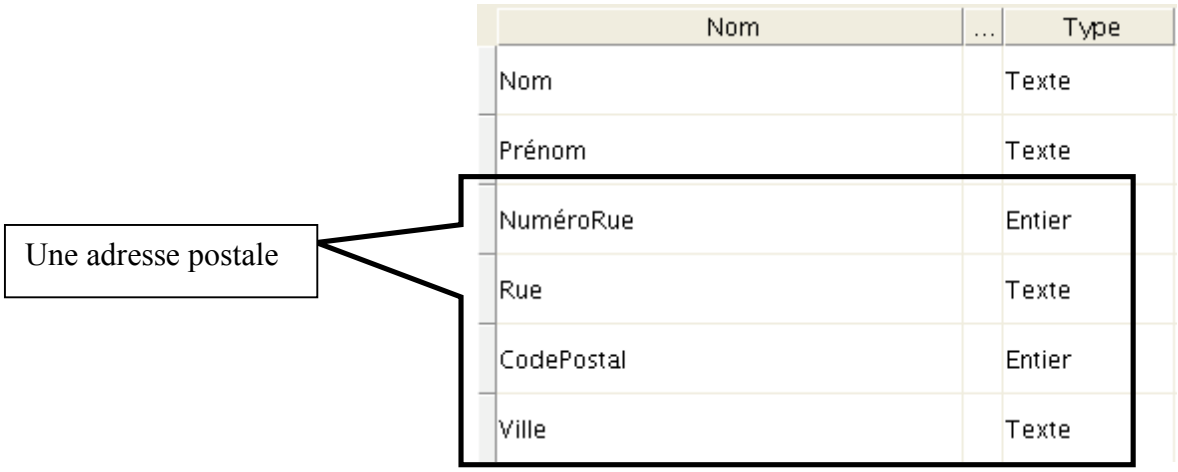


Figure 4.1.9 : Les attributs d'un objet *FichePersonnel*

Les objets concrets sont organisés dans des groupes. La sémantique usuelle d'un *Groupe* est la composition de plusieurs éléments. Cependant, comme dans K-MADe, aucun objet concret n'existe en dehors d'un groupe, il est nécessaire de définir un groupe même lorsqu'une seule instance d'un objet abstrait est utilisée dans toute l'activité. La définition de **groupes composés d'un objet unique** viole alors la sémantique usuelle des groupes.

De plus, les mêmes types de limitations ont été détectés pour la définition des groupes que pour la définition des objets, c'est-à-dire les **types des éléments disponibles** les composant (i.e : les objets concrets). En effet, chaque groupe ne peut contenir qu'un seul type d'objet concret, par exemple, il est impossible de définir un groupe *panier à provision* composé de *fruits*, *légumes*,…

Parmi les **structures de groupes disponibles** pour organiser les objets concrets, l'accès direct à un élément du groupe n'est possible que s'il s'agit du premier élément (liste) ou du dernier (pile). Pour certaines expressions, l'accès à un élément d'un groupe en fonction de sa position (quelle qu'elle soit) peut être utile, par exemple pour placer un pion dans une combinaison du jeu de Mastermind. De même, les concepteurs du *mailer* n'ont pas pu exprimer que le groupe des emails triés est composé du groupe des emails *personnels* et du groupe des emails *professionnels*.

En plus des objets qui sont associés à la tâche via l'utilisation des expressions formelles, K-MADe permet d'**associer les utilisateurs aux tâches** qu'ils exécutent. Ce type de définition n'est pas adapté à l'expression d'activités de groupes d'utilisateurs (comme dans le cas du laboratoire *Genindexe*). EUTERPE permet la définition d'un groupe comme étant un agent spécifique. Il est nécessaire d'étendre K-MAD pour qu'il permette également ce type d'*utilisateur*.

La modélisation de l'activité du laboratoire *Genindexe* a également permis d'utiliser K-MAD pour la modélisation d'une activité dans laquelle **plusieurs systèmes différents interviennent** (les différents ordinateurs ainsi que les machines utilisées pour les analyses). Dans K-MAD, lorsqu'une tâche est exécutée par une machine, elle est

identifiée comme étant une tâche *Système*. L'attribution de cette caractéristique d'exécution ne permet pas de préciser quel est le système en question. AMBOSS permet d'associer à une tâche les acteurs humains de la tâche (comme K-MADe le permet) mais également les acteurs systèmes. Cela permet de modéliser des activités faisant intervenir plusieurs systèmes tout en précisant le(s) système(s) qui interviennent dans l'exécution de la tâche.

Afin de répondre à tous ces points soulevés (et de ce fait, augmenter le pouvoir expressif de K-MAD), différentes modifications du modèle sont présentées pour répondre à ces limitations (section 4.2).

4.1.2.2. Réponses de K-MAD aux limitations détectées dans les autres modèles

Les modèles de tâches peuvent être utilisés pour concevoir des applications interactives. Dans une étude complémentaire à la notre, Sinning et al. [Sinning, et al. 2007], identifient quelques manques d'expressivité communs pour la conception de tous les types d'applications interactives : la **terminaison infructueuse des tâches** ; l'**expression d'un choix indéterminé** (choix fait sans l'intervention de l'utilisateur) ; l'**exécution concurrente sur différentes « instances »**. D'après cette étude, menée à partir de modèles de tâches CTT, Sinning et al. définissent ces points comme ne pouvant pas être exprimés en utilisant les outils de modélisation de tâches bien que leur expression permette d'accroître l'utilisation des modèles de tâches pour concevoir les applications interactives. Cette observation amène les auteurs à proposer des extensions à la notation CTT. Lors de notre utilisation de K-MAD pour réaliser les modèles de tâches des études de cas, nous avons particulièrement étudié comment ce formalisme réagit aux manques soulevés.

Terminaison infructueuse des tâches. Dans le contexte de l'utilisation de K-MADe, la terminaison infructueuse des tâches ne peut pas être spécifiée (les utilisateurs ne souhaitent pas la terminaison infructueuse de leurs tâches et K-MADe a été développé pour exprimer l'accomplissement des buts des utilisateurs). Comme nous l'avons montré dans la section 3.3, les modèles de tâches ne contiennent pas tous les éléments nécessaires à l'expression complète du dialogue des applications interactives actuelles. Bien que la simulation ressemble au dialogue des applications, certaines considérations ergonomiques imposent l'enrichissement du modèle de dialogue des applications obtenu à partir du modèle de tâches [Palanque, et al. 2003], c'est le cas de la prise en compte de la terminaison infructueuse des actions. Les deux autres limitations identifiées (l'expression d'un choix indéterminé et l'itération d'instances) sont du ressort de la sémantique de K-MAD.

Expression d'un choix indéterminé. Les choix indéterminés sont des choix qui peuvent être faits sans la participation des utilisateurs. Par exemple, la tâche de feedback après l'envoi de l'email peut être : l'indication que l'email est un email envoyé ou le message d'erreur du système d'envoi (comme pas de connexion). K-MADe permet l'expression de ce type de choix : la tâche de feedback est composée de deux sous-tâches systèmes avec des pré-conditions (voir Figure 4.1.10). Comme les conditions sont évaluées

lors de la simulation des modèles de tâches, la tâche système exécutée est celle pouvant être déclenchée en fonction de l'état du monde, indépendamment du souhait de l'utilisateur.

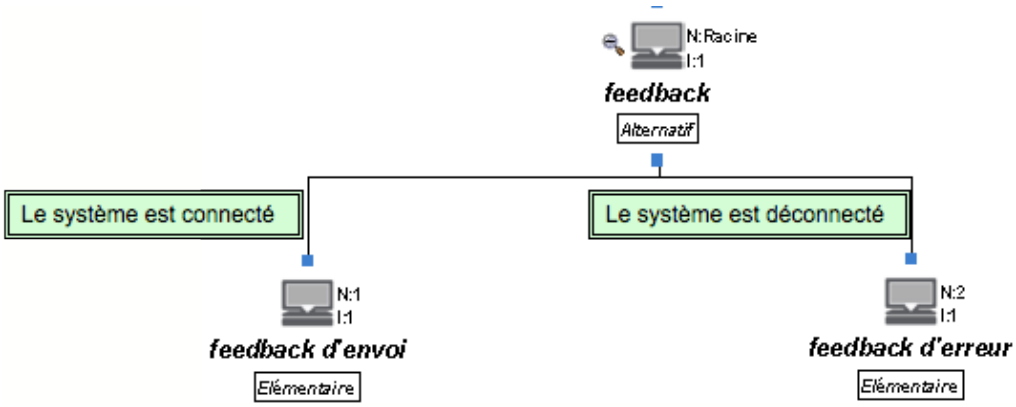


Figure 4.1.10 : Un choix indéterminé dans K-MAD

Itération d'instances. L'itération d'instances est définie comme l'exécution d'une tâche sur différentes *instances* d'un objet. Par exemple, pour l'exécution de la tâche de communication par email (production, édition et envoi), l'itération d'instance correspond à l'exécution concurrente d'une tâche (caractéristique d'itération) manipulant différents objets concrets d'un même objet abstrait (dans cet exemple, plusieurs emails). Dans notre étude de cas, en utilisant des conditions et l'opérateur de choix (*alternatif*), la tâche itérative *envoyer un nouvel email* permet la production, l'édition et l'envoi de différents emails en même temps (voir Figure 4.1.11).

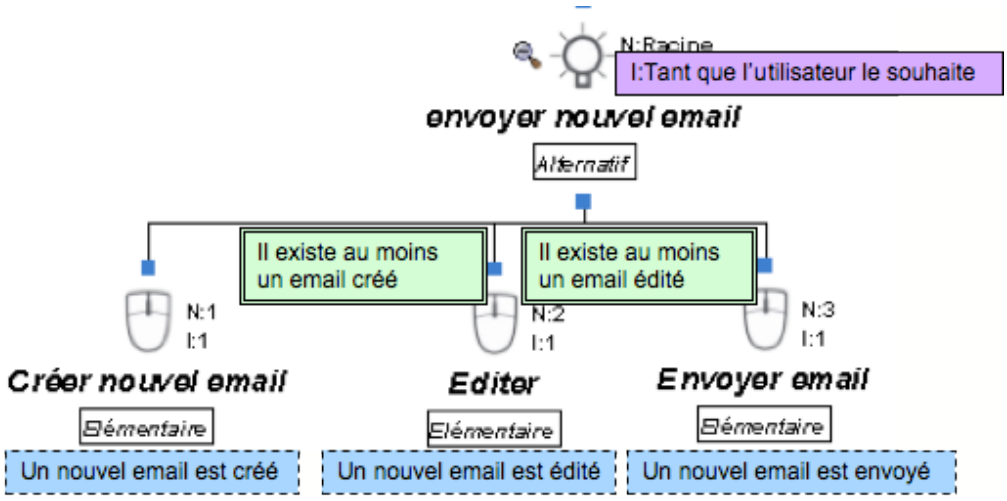


Figure 4.1.11 : Itération d'une tâche manipulant plusieurs emails

Même si cette modélisation produit des scénarios qui créent, éditent et envoient différents emails, ces instances ne sont pas visuellement présentées dans le modèle de tâches.

4.1.2.3. Bilan de l'utilisation de K-MADe pour les études de cas

Nous avons présenté dans cette section les bénéfices de l'utilisation du modèle K-MAD par rapport aux autres modèles de tâches et les limitations d'expressivité de ce même modèle. Ces observations ont été faites à partir des études de cas présentées dans la section 4.1.1. Ces bénéfices sont principalement attribués à l'utilisation de la définition formelle des objets manipulés par les tâches.

En effet, K-MAD permet notamment grâce à l'intégration d'objets formels dans le modèle, de compléter l'expression de l'ordonnancement en permettant d'exprimer la condition de fin d'une itération, le choix entre deux tâches en fonction de l'état du système…

Cependant, l'expressivité des objets et des expressions formelles les manipulant doit être enrichie afin de faciliter leur utilisation et d'accroître le pouvoir d'expression de K-MAD pour la conception des applications interactives. Nous avons, par exemple, montré que les types de composants disponibles ne permettent pas l'expression d'une représentation de tous les objets (physiques) manipulés dans les activités réelles. L'expression des objets est limitée par les types disponibles, les règles de composition (des groupes et des objets) et les composants prédéfinis pour exprimer les conditions.

Enfin, nous avons également montré comment les objets de K-MAD pouvaient être utilisés pour passer outre une limitation détectée dans d'autres modèles dans le cas de l'itération d'instance. Cependant, cette solution n'est pas optimale puisqu'elle ne permet pas d'exprimer la différence entre cette itération et les autres, aussi une étude plus approfondie doit être réalisée afin de proposer des modifications à K-MAD pour prendre en compte les itérations sur les instances ainsi que les autres limitations soulevées dans 4.1.2.1.

4.2. Propositions et extensions

Les leçons que nous avons apprises de nos études de cas ont abouti à l'identification de difficultés que des modifications de K-MAD peuvent résoudre. Ces modifications ont pour but de pallier les difficultés rencontrées lors de la définition des tâches (§ 4.2.1), de la définition des objets (§ 4.2.3 et § 4.2.4), des acteurs (§ 4.2.5) et d'augmenter l'expressivité des modèles de tâches via l'utilisation des objets (§ 4.2.2). Nous les proposons à partir du modèle K-MAD défini dans [Lucquiaud 2005b].

4.2.1. Les tâches

La première modification du modèle que nous proposons n'est pas issue de l'étude décrite dans la section précédente (§ 4.1) mais de l'étude du méta-modèle formel, décrit

dans [Lucquiaud 2005b]. Dans ce méta-modèle, l'une des caractéristiques d'une tâche est sa **position** dans l'arbre de tâche.

La position d'une tâche dans l'arbre de tâche est un élément intéressant dans notre étude car elle détermine (combinée aux autres éléments d'ordonnancement) l'ordre de parcours dans l'arbre de tâche. Par exemple, lorsqu'une tâche est décomposée en sous-tâches séquentiellement exécutées, l'ordre suit le sens de lecture (de la tâche de position la plus à gauche à la tâche de position la plus à droite). Ainsi, pour *emprunter un livre* à la bibliothèque, en suivant la modélisation proposée sur la Figure 4.2.1, la première tâche à accomplir est d'*aller à la bibliothèque*, avant de *choisir un livre*, de *remplir le formulaire d'emprunt* et de *ramener le livre chez soi*.

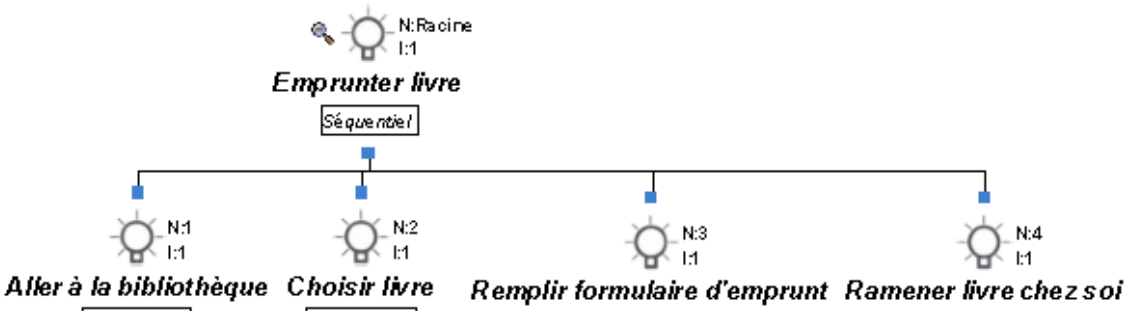


Figure 4.2.1 : Modèle de tâche d'*emprunter un livre*

Dans le méta-modèle EXPRESS de la tâche de K-MAD, la position des tâches est exprimée par deux entiers représentant les coordonnées physiques de la tâche (Figure 4.2.2a). Nous proposons de modifier cette définition liée à la représentation physique de la tâche pour définir la position d'une tâche comme étant obtenue de manière calculée (DER en EXPRESS) en fonction de la place de la tâche dans la liste des enfants de sa tâche mère (un extrait du modèle modifié est présenté sur la Figure 4.2.2b). Cette position est alors définie comme la liste des places de tous les parents dans leur fratrie à laquelle est ajoutée la propre place de la tâche dans sa fratrie.

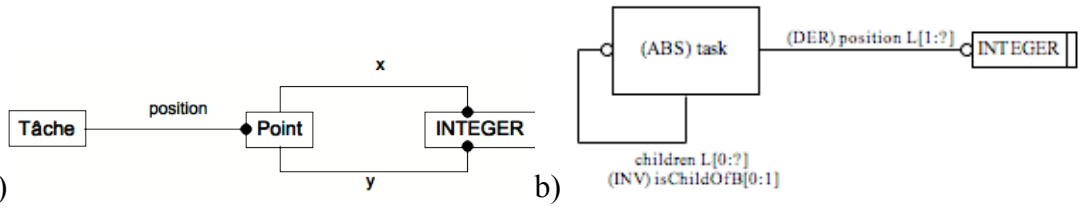


Figure 4.2.2 : Extrait du méta-modèle EXPRESS d'une tâche exprimant la position d'une tâche dans a) le méta-modèle original et b) dans le méta-modèle modifié

Par exemple, le calcul de la position de la tâche T10 dans l'arbre de tâche représenté sur la Figure 4.2.4a en suivant l'algorithme présenté sur la Figure 4.2.3 est représenté sur la Figure 4.2.4b.

```
POSITION (T:Tache) : LISTE

If (T.IsChildOf()==null){
   return {1};
   else {
     P:= place(T, T.IsChildOf.Child());
     return(concat(POSITION(T.IsChilOf), {P}));
   }
}
```

Figure 4.2.3 : Algorithme de calcul de la position d'une tâche

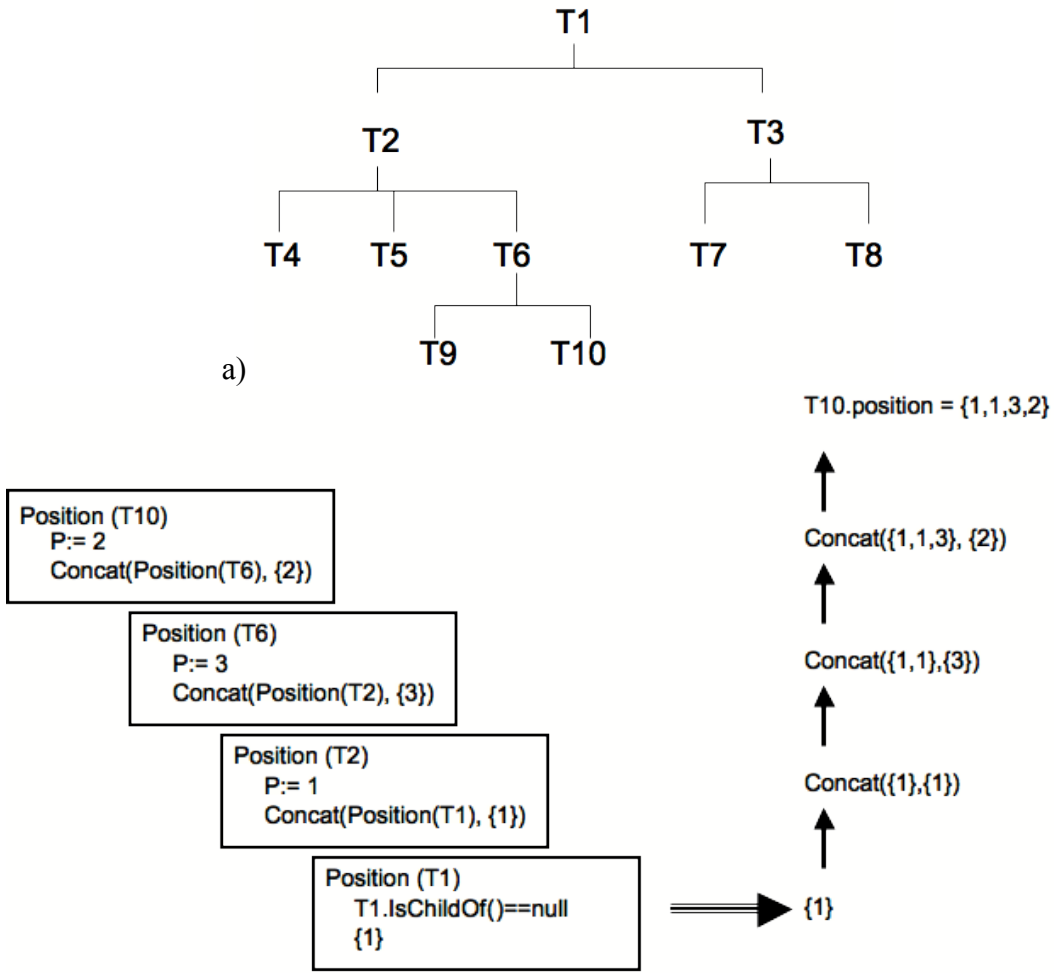


Figure 4.2.4 : Exemple d'attribution de la position d'une tâche

Comme nous l'avons indiqué précédemment, l'utilisation de K-MADe pour la conception des études de cas a mis en lumière une difficulté d'utilisation du modèle implémenté, due à la terminologie utilisée pour exprimer la modification des valeurs. Nous avons **modifié le terme utilisé de post-condition** à *action*. L'expression décrite dans cet

attribut de la tâche représente les modifications sur les objets engendrées par l'exécution de la tâche.

Nous proposons d'**ajouter une véritable post-condition** pour définir une expression booléenne devant être validée après l'exécution de la tâche. Ces deux attributs de la tâche représentent deux concepts différents. Par exemple, la tâche *Poser Pion* a pour *action* : « un pion de couleur c est placé à la place p » ; alors que la post-condition est : « un pion supplémentaire est ajouté à la combinaison en cours ».

Les composants affectés par ces modifications dans le méta-modèle implémenté dans l'outil et le méta-modèle modifié sont représentés sur la Figure 4.2.5.

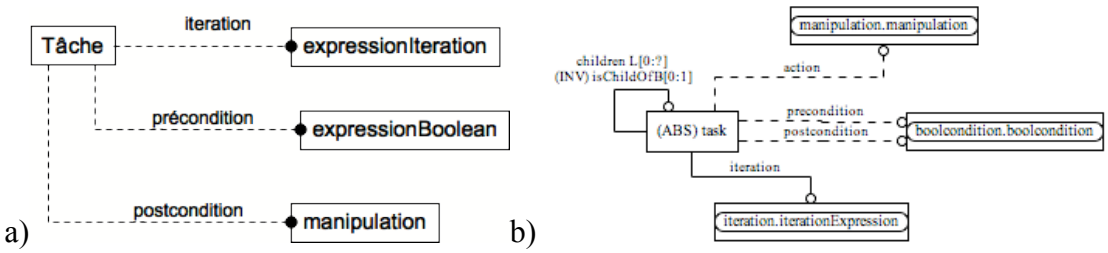


Figure 4.2.5 : Extrait du méta-modèle EXPRESS d'une tâche exprimant les conditions de terminaison d'une tâche dans a) le méta-modèle de l'outil et b) dans le méta-modèle modifié

4.2.2. Expressivité de l'utilisation des objets

K-MAD dispose d'expressions associées aux tâches qui manipulent les objets : les *pre-conditions* (les conditions pour que les tâches soient exécutables), les *post-conditions* de K-MADe (les actions résultant de l'exécution des tâches) et les *conditions d'itération* (condition à la répétition des tâches). En plus de la distinction entre la *post-condition* et l'*action* que nous proposons dans 4.2.1, nous avons identifié un besoin de **compléter la variété des conditions d'itération disponibles**. Nos propositions sur ce point sont décrites dans la section 4.2.2.1. De plus, pour éditer les expressions exécutables, les concepteurs utilisent les calculettes de condition (voir la Figure 4.1.6).

Notre étude de la modélisation pour la conception a montré que **l'ensemble des opérateurs disponibles** devait être étendu. Nous présenterons nos propositions pour étoffer l'ensemble des opérateurs disponibles dans la section 4.2.2.2.

4.2.2.1. Expressions de l'itération

Dans K-MADe, l'itération des tâches est gérée par une expression. Cette expression peut être définie par un nombre (nombre fixe de répétitions de la tâche) ou une expression

booléenne (la tâche est répétée tant que la condition est vraie). Ces expressions d'itération ne couvrent pas toutes les itérations que le concepteur doit exprimer comme nous l'avons montré dans 4.1.2.1 et 4.1.2.2, certaines itérations ne sont pas définies par des expressions évaluables mais par l'initiative des utilisateurs. Par exemple, un utilisateur ajoute les destinataires des emails autant de fois qu'il le veut. Afin de représenter cette condition spécifique, une **expression de condition d'itération** qui collecte le **souhait de l'utilisateur** peut être définie.

Cependant, la fin des itérations d'une tâche par un utilisateur est sémantiquement différente d'une fin due à la modification de l'état d'un objet. Donc, nous préférons diviser le contrôle de l'itération en deux contrôles distincts : un contrôle dirigé par le modèle et un contrôle dirigé par l'utilisateur.

L'étude présentée dans [Sinning, et al. 2007] présente une autre limitation : le fait que le concepteur ne peut pas exprimer les tâches réalisées sur différentes instances d'un même objet (nommé *instance iteration*) en même temps, comme l'**édition de plusieurs emails en même temps.** Nous avons montré dans la section 4.1.2.2 que K-MAD permet la représentation de ce type d'itération mais sans utiliser de moyen expressif particulier représentant l'itération sur des instances. Sinning et al. proposent d'ajouter une autre caractéristique temporelle (un autre opérateur temporel) qui spécifie cette itération particulière.

Donc, deux différents types d'itérations spécifieront les caractéristiques d'itération des tâches : l'itération séquentielle des tâches (les exécutions des tâches sont séquentiellement exécutées) telle que les outils de modélisation des tâches le proposent ; et, l'itération d'instances des tâches. Cette proposition permet la distinction des deux différents concepts d'itération. Nous souhaitons également intégrer un nouvel opérateur afin de représenter le rôle des objets concrets dans l'itération. En résumé, K-MADe permettra trois différents types d'itération : (1) l'itération des tâches sur les *instances* et les itérations contrôlées ((2) l'itération contrôlée par le modèle et (3) l'itération contrôlée par l'utilisateur). Le Tableau 4-3 résume les itérations disponibles dans le méta-modèle de K-MAD original et dans le méta-modèle de K-MAD modifié.

	K-MAD(v1)	K-MAD(v2)
itération sur les objets	-	√
nombre défini d'itération	√	√
expression d'itération	√	√
itération contrôlée par l'utilisateur	-	√

Tableau 4-3 : Présence des itérations dans K-MAD

4.2.2.2. Les opérateurs

Une partie des calculettes présente l'ensemble des opérateurs mathématiques (Figure 4.2.6). L'étude de modèles de description de données (UML/OCL [Object Management Group 1997] et EXPRESS [EXPRESS 1994]) a montré la nécessité de treize

opérateurs mathématiques : trois opérateurs logiques (AND, OR, NOT) ; quatre opérateurs arithmétiques (+, -, *, /) ; et six opérateurs relationnels (=, <, >, !=, >=, <=).

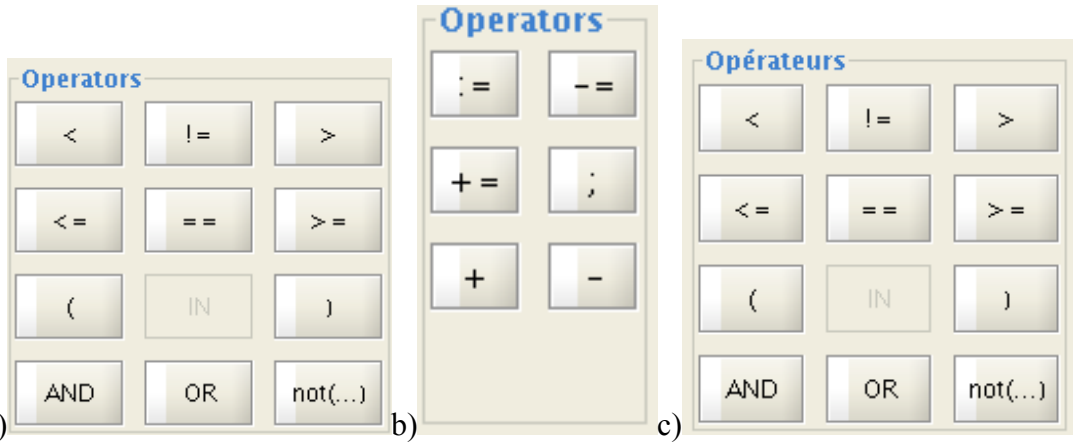


Figure 4.2.6 : Les opérateurs disponibles dans les calculettes de K-MADe pour l'édition a) d'une pré condition b) d'une post-condition et c) d'une itération

Comme le montre le Tableau 4-4, tous les opérateurs logiques et relationnels existent dans K-MADe. La calculette d'édition des post-conditions possède deux opérateurs arithmétiques : l'opérateurs d'addition et celui de soustraction. Aucune explication sur l'absence des deux autres opérateurs arithmétiques (*, /) n'ayant été trouvée, en plus de ces opérations arithmétiques, les opérations de multiplication et de division doivent être ajoutées.

	Pré condition	Post condition	Itération
AND, OR, NOT	√	-	√
+, -, *, /	-	√ (+, -)	-
=, <, >, !=	√	-	√

Tableau 4-4 : Comparaison des opérateurs dans les calculettes de K-MAD

Nous observons une distance entre les ensembles d'opérateurs et de fonctions qui sont disponibles dans les différentes fenêtres d'édition (les *pre-conditions*, *post-conditions*, et *conditions d'itération*). Cette distance est illustrée sur la Figure 4.1.6 qui montre explicitement que les ensembles d'opérateurs (Figure 4.2.6) et les ensembles de fonctions (getValue(), set(), isEmpty()…) (marque 2 sur la Figure 4.1.6) sont composés différemment. Cette distance peut être expliquée par la différence de valeur des deux types d'expression. Alors que les *pré conditions* et les *conditions d'itération* expriment des conditions booléennes, les *post-conditions* (du modèle K-MAD initial) expriment les modifications des valeurs des objets. Cependant, pour exprimer l'une de ces conditions, le concepteur peut avoir besoin des opérations disponibles pour éditer le second type de conditions. Par exemple, un attribut booléen peut être le résultat de la comparaison de deux nombres (comme dans le cas de la terminaison d'un set (Figure 4.1.8)). L'affectation d'une valeur à un attribut (:= dans la fenêtre de post-condition) nécessite alors de mettre le résultat de cette comparaison (opérateur relationnel de la fenêtre de pré-condition) de deux

valeurs (d'attributs d'objet ou entrés). Afin de permettre la composition de fonctions et d'opérations booléennes, K-MADe doit fournir toutes les opérations et les fonctions dans toutes les fenêtres de définition d'expressions.

4.2.3. La définition des attributs des objets

Afin de définir les objets de K-MAD, des attributs sont définis. Ces attributs sont composés de leur nom et de leur type. Le type définit le domaine de valeur de l'attribut. Les types disponibles dans K-MADe sont : *entier*, *booléen* et *texte*.

L'utilisation de K-MADe pour concevoir différents modèles de tâches met en lumière certaines limitations dues aux types disponibles (§ 4.1.2.1). Nous divisons cet ensemble en deux groupes différents en fonction des niveaux de définition : les attributs de type simple et les attributs de type construit. Les attributs dits de type *simple* sont les types d'attributs qui ne sont pas composés. Dans K-MADe, tous les types prédéfinis sont de type simple (*textes*, *booléens*, *entiers*). Cependant, cet ensemble de types simples disponibles ne couvre pas tous les types d'attributs que le concepteur peut vouloir décrire. Sans en proposer autant qu'il y en a dans les modèles de domaine (ensemble non exhaustif comme le précise la méta-modélisation du modèle de domaine proposée dans [Morfeo]), nous proposons d'élargir l'ensemble des types disponibles. En plus de ces attributs *simples*, des attributs *construits* sont nécessaires pour compléter la définition des objets.

4.2.3.1. Les attributs de type simple

En plus des types suscités (*texte, entier* et *booléen*), le noyau K-MAD dispose d'un type *énuméré* qui permet la définition d'une liste de valeur possible (comme pour un attribut *couleur*, les valeurs *jaune, rouge, vert* ou *bleu* par exemple). Bien que prévu dans l'outil, ce type d'attributs n'a pas été implémenté alors que sa présence nous aurait permis de préciser les valeurs des couleurs de pions du Mastermind par exemple. Exceptée cette absence qui permet de préciser des valeurs, aucune autre limitation n'a été détectée à propos de la définition des attributs textuels (*texte*) et des attributs binaires (*booléen*), la couverture des attributs numériques (*entier*). Au contraire, nous avons observé que K-MAD ne prenait pas en compte toutes les valeurs numériques possibles.

Afin d'exprimer les valeurs numériques, EXPRESS [EXPRESS 1994] et UML/OCL [Object Management Group 1997] (deux autres langages d'expression de données) possèdent différents types numériques (*réel*, *entier*). En étudiant la large possibilité de définition pour les deux langages, nous proposons d'augmenter le pouvoir expressif des attributs des objets en ajoutant le type *réel* au type *entier*.

Les deux langages (UML/OCL et EXPRESS) sont utilisés par les informaticiens qui connaissent les noms des types comme *real* et *integer*. Au contraire, les utilisateurs de K-MADe peuvent ne pas être des experts en informatique. Nous considérons le coût

d'apprentissage des différences de types numériques trop important par rapport aux bénéfices engendrés. C'est pourquoi, nous proposons de présenter le type *Numérique* composé du type *Entier* et du type *Réel*. En fonction du contexte d'utilisation de ce type *numérique*, il indiquera soit un entier, soit un réel.

Le Tableau 4-5 représente les types d'attributs simples disponibles dans le méta-modèle implémenté dans K-MADe pour définir les objets et dans le méta-modèle de K-MAD modifié.

		K-MAD(v1)	K-MAD(v2)
Numérique	entier	√	√
	réel	-	
Texte		√	√
Booléen		√	√

Tableau 4-5 : Les types disponibles dans le méta-modèle original et dans le méta-modèle modifié

4.2.3.2. Les attributs de type construit

Les types simples d'attribut ne couvrent pas l'ensemble du domaine de définition des attributs. Comme nous l'avons illustré dans 4.1.2.1, certains objets ne sont pas construits à partir des types simples (*Texte*, *Entier*, *Booléen*) uniquement.

Afin d'intégrer les types définis par les utilisateurs de K-MADe, nous ajoutons les objets abstraits à l'ensemble des types d'attribut. La définition de l'objet *FichePersonnel* à l'aide de cette extension est composée de trois attributs : le nom (texte), le prénom (texte) et l'adresse (objet abstrait : adresse postale). Le type *adresse postale* peut alors être défini comme un objet abstrait composé d'un numéro (entier), d'un nom de rue (texte), d'un code postal (entier) et d'un nom de ville (texte). La Figure 4.2.7 représente la définition des deux objets abstraits dans une représentation UML.

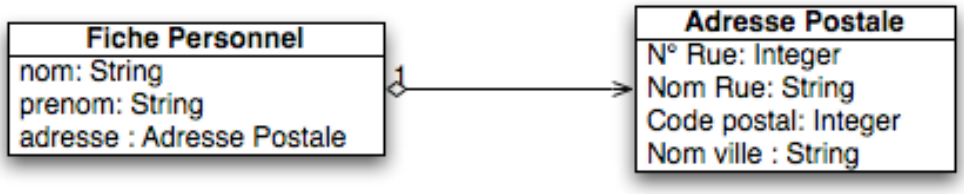


Figure 4.2.7 : La représentation UML de la définition de l'objet *adresse postale*

Avec cette modification, les attributs des objets peuvent être du *texte*, un *numérique* ou un *booléen* ou un autre *objet abstrait*.

Chaque attribut contient alors uniquement un seul élément. Cette condition est une autre limitation à la description des objets. Par exemple, elle implique la limitation du nombre de destinataire d'un email (seul un destinataire peut être défini pour chaque email alors que dans le monde réel, un même email peut être envoyé à différents destinataires). Afin d'exprimer les attributs qui contiennent une ensemble d'entités (comme une liste de destinataires de l'email), nous proposons d'ajouter aux types d'attributs : la collection d'objets abstraits. K-MAD définit les collections, nommées *groupe* (liste, pile…). Nous choisissons d'utiliser ce concept comme étant un type d'attribut supplémentaire. Dès lors, le nouvel ensemble des types disponibles est : *booléen*, *numérique*, *texte*, *objet abstrait* (un objet abstrait parmi les objets abstraits définis par l'utilisateur) et *groupe* (un groupe parmi les groupes définis par les utilisateurs).

4.2.4. Définition des groupes d'objets

Dans l'outil K-MADe, « *un groupe est une sorte de structure (pile, liste, ensemble ou unique) qui consiste à stocker les instances d'un même objet abstrait* » (définition issue de [Baron et Scapin]) (voir Figure 4.2.8). Trois règles régissent le lien entre les objets concrets et les groupes :

- un groupe contient des instances (objets concrets) d'un seul type d'objet abstrait
- chaque objet concret appartient à un groupe
- chaque groupe organise les objets concrets d'un type de structure (Pile, Liste, Ensemble, Unique)

Ces règles restreignent la définition des groupes d'objets et leur utilisation dans K-MADe. Nous avons identifié trois limitations :

- l'appartenance obligatoire des objets concrets à un groupe,
- la composition des groupes avec uniquement des objets concrets (aucune autre entité)
- les structures utilisées pour organiser les objets.

4.2.4.1. Les objets concrets indépendants des groupes

La définition des objets concrets dans K-MADe implique qu'un objet concret est composé d'un nom, d'observation à son propos et du groupe auquel il appartient. Le nom et le groupe sont des attributs obligatoires d'un objet concret. Donc, un objet concret ne peut ni exister indépendamment d'un groupe, ni être contenu dans plusieurs groupes différents.

Pour exprimer une unique instance, le concepteur doit définir un groupe (comme par exemple un groupe contenant un seul email en cours d'édition, voir Figure 4.2.8). Cette définition n'est pas intuitive pour les utilisateurs [Caffiau, et al. 2008a] qui ne comprennent pas la sémantique d'un groupe voué à ne contenir qu'un seul objet. Afin de

sémantiquement redéfinir les groupes dans K-MADe (afin d'être conforme à la sémantique des groupes communément utilisée), nous proposons de modifier les relations entre les objets concrets et les groupes (Figure 4.2.9). Cette modification implique que l'existence des objets concrets devienne indépendante des groupes définis et qu'un groupe soit défini comme étant un conteneur de différents types d'objets concrets.

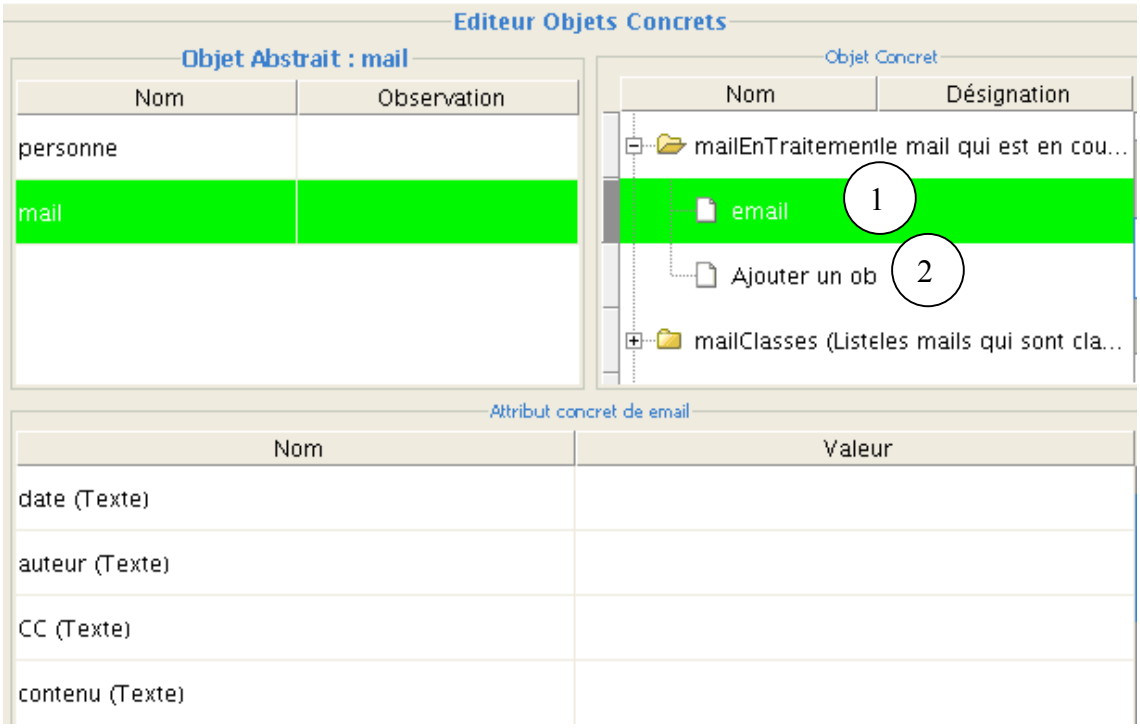
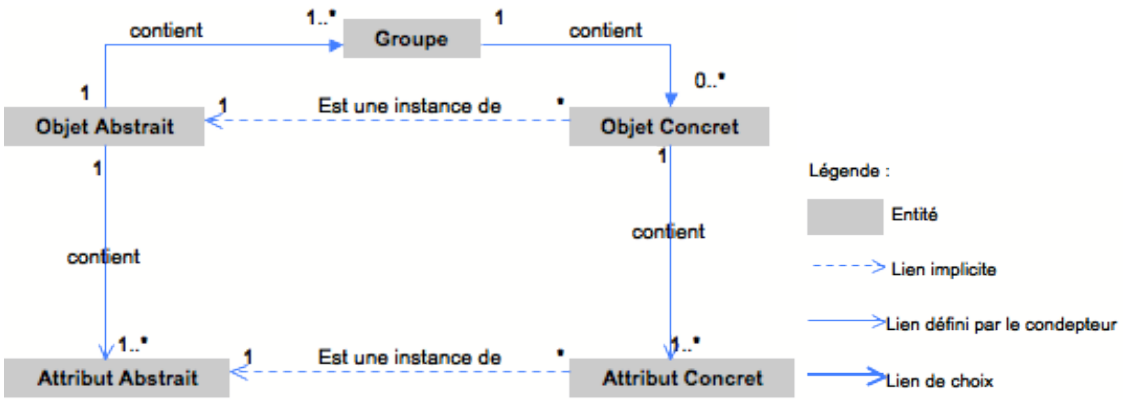


Figure 4.2.8 : Définition d'une unique instance d'objet

Cette modification de relation rend possible de définir un même objet concret comme appartenant à plusieurs groupes différents. Cette double appartenance peut être nécessaire pour concevoir certains modèles de tâches (par exemple, une personne peut être à la fois un employé et un client d'un même magasin).

a)



b)

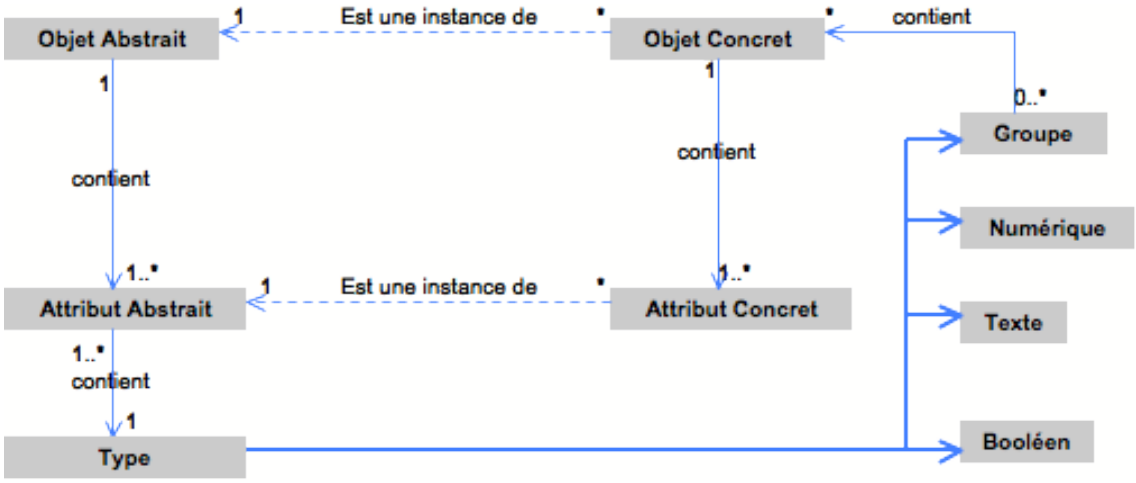


Figure 4.2.9 : Extraits des relations a) originales (issu de [Baron et Scapin]) et b) modifiées entre les objets abstraits, les objets concrets et les groupes

4.2.4.2. Les groupes composés d'objets de types différents

Enfin, la définition d'un groupe comme étant une structure stockant des objets concrets d'uniquement un seul objet abstrait, empêche la conception d'un groupe contenant des objets de types différents (comme un groupe représentant un panier à provision contenant des articles différents). La modification de cette relation entre les objets concrets et les groupes permet d'abstraire cette relation comme une caractéristique entre deux entités différentes. Donc, un groupe contient des objets concrets sans aucune restriction due à son objet abstrait de référence.

4.2.4.3. Groupes composés de différents groupes

Dans le méta-modèle de K-MAD défini dans [Lucquiaud 2005b]. Un groupe n'est pas composé uniquement d'un nom et de son type de collection (liste, ensemble…) mais également par les objets concrets qu'il contient. Cette définition implique l'expression de groupe à un seul niveau. Cependant, certains groupes peuvent être des collections d'autres. Par exemple, un grand nombre de *mailers* proposent de définir des dossiers qui contiennent d'autres dossiers qui contiennent des emails, comme un dossier *relations professionnelles* composé de cinq dossiers correspondant à chacun des cinq projets en cours.

Dans K-MADe, la composition de collections de groupes ne peut pas être exprimée. Cependant, la modification de la définition des objets qui permet d'inclure les groupes comme étant des types d'attributs disponibles (voir 4.2.3) fournit un description alternative. Dans notre exemple, le groupe *DossierEmails* peut contenir le groupe *emailProjet1* via l'objet *dossierProjet1*. La Figure 4.2.9b présente les liens définis entre les composants *groupe*, *objets* et *attributs* du méta-modèle K-MAD(v2).

4.2.4.4. Les collections

Les groupes de K-MADe organisent les objets. Les collections disponibles sont : *Unique*, *Liste*, *Pile* et *Ensemble*. Le type *unique* est dédié aux groupes qui contiennent uniquement un seul objet. Ce type de groupe est nécessaire parce qu'un objet concret n'existe pas indépendamment d'un groupe. La suppression de cette obligation ne rend plus indispensable le type *Unique*. La définition des objets concrets indépendamment des groupes (voir 4.2.4.1) nous permet de supprimer le type de groupe *Unique*.

Les trois autres types de groupes (*pile*, *liste* et *ensemble*) sont des structures différentes qui organisent les objets concrets. Une *liste* est une structure dans laquelle le premier objet concret stocké est le premier extrait (FIFO). Une *pile* est une structure dans laquelle le dernier objet concret stocké est le premier extrait (LIFO). Un *ensemble* est une structure dans laquelle aucun ordre n'est spécifié, l'objet concret extrait est désigné par l'utilisateur. En utilisant ces types pour exprimer les groupes, nous avons observé que les concepteurs ne peuvent pas exprimer l'accès aux objets concrets en fonction de leur place dans les collections. L'accès direct peut être utilisé pour comparer, par exemple, la valeur du troisième jeton dans un code du Mastermind. Les groupes numérotés sont des éléments habituellement utilisés dans le domaine informatique (cela correspond à la notion de tableau dans la plupart des langages informatiques). C'est pourquoi, nous proposons de l'ajouter dans l'ensemble des collections disponibles.

Les types de groupes disponibles nécessaires à la description de l'activité sont : *liste*, *pile*, *ensemble* et *tableau*. Ces types sont les collections exprimables dans les modèles de données UML/OCL et EXPRESS. Il est à noter qu'EXPRESS dispose de deux types d'ensembles : les SET et les BAG, l'un autorisant la présence des doublons dans l'ensemble alors que l'autre non. Une structure d'ensemble autorisant la présence de doublons correspond à une vue mathématique des ensembles et n'a pas de correspondance dans le monde réel, c'est pourquoi nous n'avons pas ajouté ce type de structure dans K-MAD(v2).

Les types proposés sont proches des types utilisés en informatique et pas des catégories de groupes utilisées dans le monde des utilisateurs. Afin de couvrir ces notions informatiques, nous proposons que K-MADe propose un outil d'assistance qui affecte automatiquement les types d'agrégation pour les collections en fonction des réponses des concepteurs à différentes questions (Figure 4.2.10).

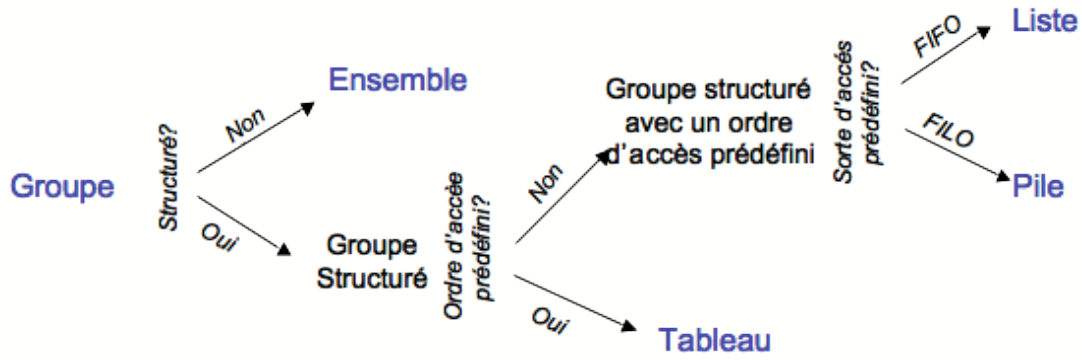


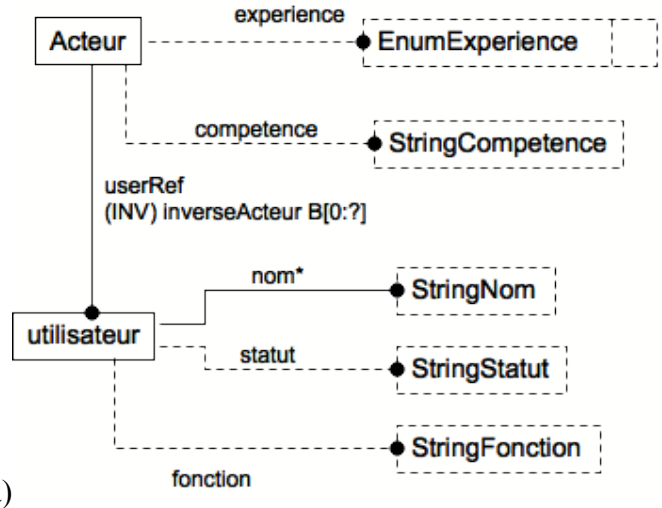
Figure 4.2.10 : Arbre de décision pour le type de groupe

4.2.5. Les acteurs

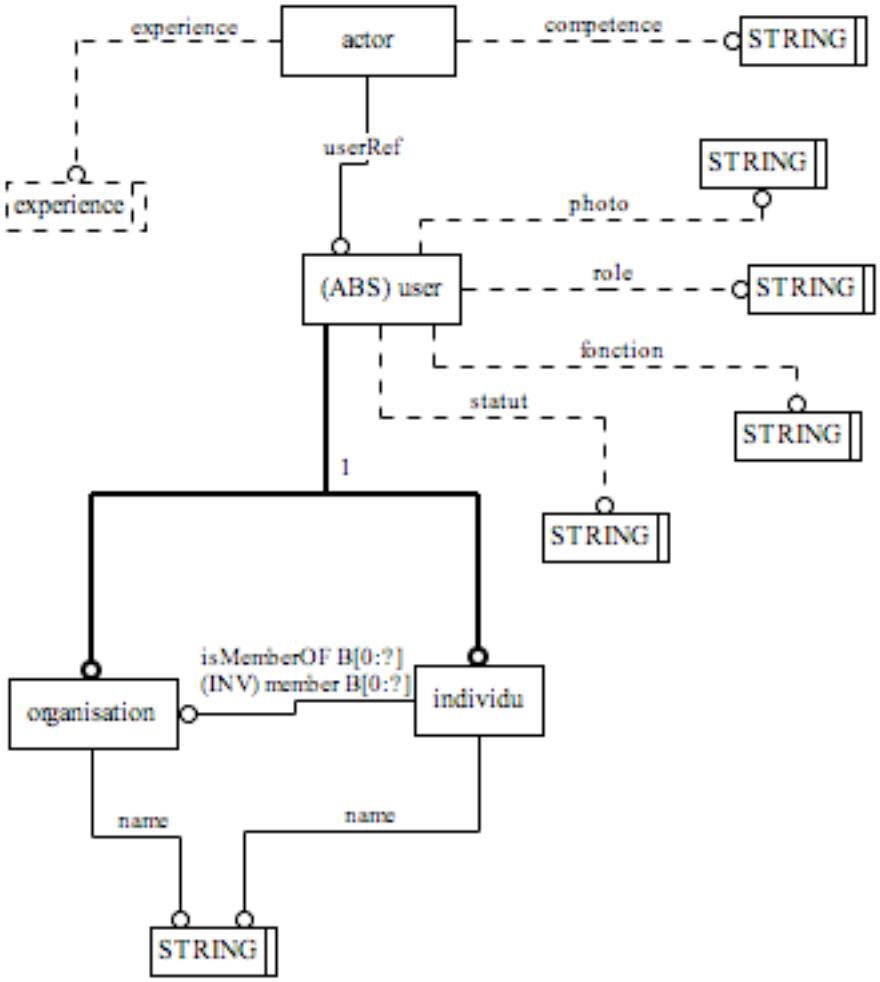
Des acteurs peuvent être spécifiés parmi les utilisateurs précisés. Cependant, ceux-ci sont des acteurs humains, il n'est donc pas possible de préciser des acteurs pour les tâches systèmes (§ 4.2.5.2). De plus, il est possible d'associer plusieurs acteurs à une même tâche mais uniquement en spécifiant l'ensemble des acteurs et non pas un groupe d'acteurs, alors que fondamentalement, une activité peut concerner un acteur individuel ou un groupe d'acteurs (comme une entreprise) (§ 4.2.5.1).

4.2.5.1. Les acteurs humains

Il est possible de spécifier plusieurs acteurs pour une même tâche (à condition qu'un individu intervienne dans son exécution). Dans K-MAD, cela signifie qu'ils interviennent tous dans l'exécution de celle-ci. Les acteurs peuvent donc être des utilisateurs mais également des organisations, en fonction du point de vue de la modélisation. Il semble donc que deux cas sont à différencier : la modélisation de l'activité d'une personne physique (pour le recueil des besoins, l'évaluation…) et la modélisation des tâches dans une organisation. C'est pourquoi, nous proposons d'ajouter aux acteurs individuels, les acteurs *organisationnels* en permettant la définition de groupe d'utilisateur comme acteur particulier (nommé *acteur groupe*).



a)



b)

Figure 4.2.11 : Les *acteurs* dans a) K-MAD(v1) et b) K-MAD(v2)

4.2.5.2. Les acteurs machines

Lorsqu'un individu intervient dans l'exécution d'une tâche, il est possible de préciser l'acteur parmi ceux préalablement définis par le concepteur. Cependant, lorsqu'une tâche est exécutée par un système, il n'est pas possible de préciser de quel type de système il s'agit.

Par exemple, pour la modélisation de l'activité de *Genindexe* (§ 4.1.1), certaines tâches sont exécutées par un séquenceur, elles sont alors indiquées comme étant des tâches système. Il n'y a alors aucun moyen de savoir quel système accomplit quelle tâche. De même que nous pouvons créer plusieurs utilisateurs et préciser pour l'exécution d'une tâche quel est l'utilisateur qui l'accomplit, il serait parfois utile de pouvoir préciser quel est le système qui réalise une tâche système.

Nous proposons de permettre la spécification de ces acteurs systèmes en ajoutant un composant *système* au modèle (comme il existe un composant *utilisateur*) défini par un nom et pour lequel la fonction et une description textuelle peuvent être ajoutées.

De même que les utilisateurs sont liés aux tâches via la relation d'*acteur*, les systèmes seront liés aux tâches via une relation *machine*. Cette relation précisera également (par un booléen), si le système est un système informatisé ou non afin de faciliter le travail de conception pour ce système informatisé. Nous proposons donc de définir les acteurs machines en suivant le même schéma que les acteurs humains. Cet élément est méta-modélisé dans K-MAD(v2) comme illustré sur la Figure 4.2.12.

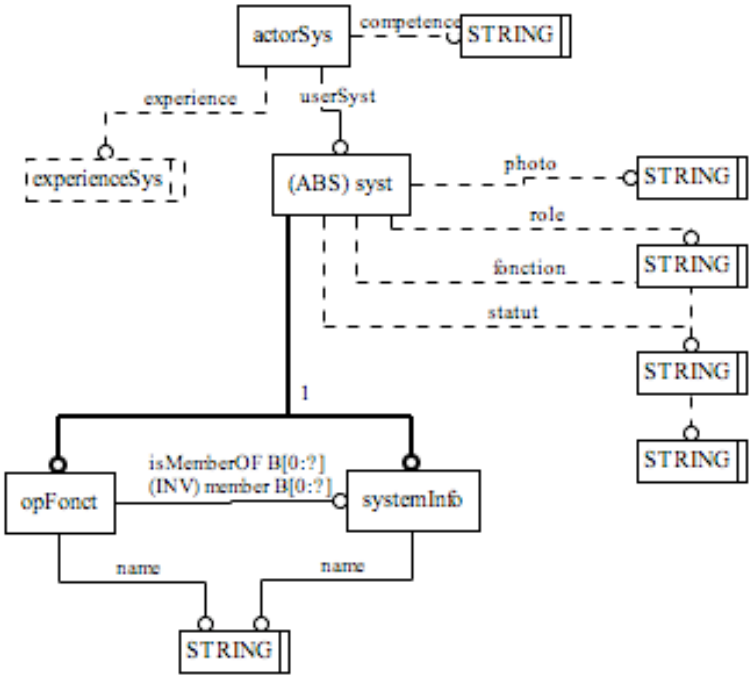


Figure 4.2.12 : Méta-modèle de l'acteur machine dans K-MAD(v2)

4.3. Passage de K-MAD(v1) à K-MAD(v2)

L'étude présentée dans ce chapitre a abouti à la définition de modifications au modèle de tâche K-MAD (section 4.2). Ces modifications ont pour but de compléter le pouvoir d'expression du modèle K-MAD pour les informations vérifiables dans la conception du dialogue des applications interactives.

Dans le but de proposer la vérification de la cohérence du modèle de dialogue à partir d'un modèle de tâches le plus expressif possible, nous souhaitons utiliser des modèles de tâches exprimés avec ce formalisme complété (que nous nommons K-MAD(v2)). Cependant, nous souhaitons également permettre l'utilisation des modèles de tâches exprimés sous forme de modèles K-MAD(v1) et édités à l'aide de K-MADe en permettant leurs complétions éventuelles. Dans ce but, nous proposons une transformation

de modèles K-MAD(v1) en K-MAD(v2). Nous présentons, dans la première section (4.3.1), les différences existant entre les deux modèles et les règles de transformation permettant de passer de l'un à l'autre pour les entités *tâches*.

En plus de la tâche, les entités *Objets* du modèle de tâches ont également subi des modifications. La seconde partie (4.3.2) détaille les modifications nécessitant des transformations de la définition des objets de K-MAD(v1) pour les inclure dans K-MAD(v2) (résumés dans le Tableau 4-6).

Enfin, nous présentons le passage des entités *Utilisateur* et *Machine* des modèles K-MAD(v1) vers les modèles K-MAD(v2).

4.3.1. Différences entre les deux formalismes pour les entités *tâche*

Lors de la conception du méta-modèle de K-MAD(v2), nous avons fait le choix de modifier la définition de la donnée **position** du modèle pour la remplacer par une liste d'entiers représentant la place de l'ensemble des tâches. Cette modification entraîne la suppression de l'entité *point*.

Nous avons également fait le choix de ne plus spécifier lorsqu'une tâche est interruptible (cas le plus fréquent) mais lorsqu'elle ne l'est pas. La valeur de l'interruptibilité d'une tâche K-MAD(v1) est alors inversée pour stipuler la valeur de NonInterruptibilité de la tâche K-MAD(v2).

Enfin, nous avons redéfini les post-conditions de K-MAD(v1) en action dans K-MAD(v2) et introduit une condition booléenne comme post-condition. Les post-conditions de K-MAD(v1) spécifient les modifications sur les objets engendrées par l'exécution de la tâches, ce qui correspond à la définition usuelle des actions. Le passage des post-conditions de K-MAD(v1) dans le modèle K-MAD(v2) consiste donc en la redéfinition de ces post-conditions en actions.

Le Tableau 4-6 résume l'ensemble des différences entre K-MAD(v1) et K-MAD(v2) nécessitant une transformation pour que la tâche soit conforme au formalisme K-MAD(v2).

K-MAD(v1)	K-MAD(v2)	Explication de la modification
Position	Position (Liste d'entiers)	Calcul automatique des valeurs en fonction de l'ordre des tâches dans les listes des tâches filles
Point	-	Suppression de la donnée
Interruptibilité (Booléen)	NonInterruptibile (Booléen)	Valeur inverse de celle d'*Interruptibilité* de K-MAD(v1)
Post-condition (Expression logique)	Action (Expression logique)	La *post-condition* de K-MAD(v1) devient *Action* dans K-MAD(v2)

	Post-condition (Expression Booléenne)	La valeur de la *post-condition* dans K-MAD(v2) est initialisée à vrai

Tableau 4-6 : Transformations des données de K-MAD(v1) pour le passage à K-MAD(v2) pour l'entité tâche.

4.3.2. Différences entre les deux formalismes pour les entités *objets*

Sept modifications ont été apportées à la définition et à l'utilisation des objets dans K-MAD(v2). Ces modifications ont été développées dans les sections 4.2.2, 4.2.3 et 4.2.4. Nous les résumons dans le Tableau 4-7.

K-MAD(v1)	**K-MAD(v2)**
Type d'attribut : Entier, Booléen, Texte	type numérique = entier + réel
	ajout du type groupe comme type d'attribut
	ajout du type objet abstrait comme type d'attribut
Type d'un groupe : liste, pile, ensemble, unique	Suppression du type *unique*
	Ajout du type *tableau*
Les objets concrets appartiennent à un seul groupe contenant un seul type d'objets abstraits	Groupes d'objets concrets de types différents
	Objets concrets indépendants des groupes

Tableau 4-7 : Différences entre les objets de K-MAD(v1) et K-MAD(v2)

Seule la suppression du type *unique* comme collection disponible nécessite une transformation des données pour le passage de K-MAD(v1) à K-MAD(v2). En effet, les autres modifications ont pour but l'extension du formalisme, les données contenues dans un modèle K-MAD(v1) sont donc incluses dans le formalisme K-MAD(v2).

Le rôle de la collection *unique* était de permettre la spécification d'un groupe composé d'un seul objet concret. Comme les modifications apportées permettent la définition d'objets concrets indépendants de groupe, la nécessité d'un type *unique* n'est plus avérée. Aussi la transformation d'un groupe *unique* de K-MAD(v1) dans K-MAD(v2) est faite de deux manières :

- si le groupe est vide : suppression du groupe
- si le groupe contient un élément : définition de l'objet concret comme indépendant

4.3.3. Différences entre les deux formalismes pour les entités *utilisateur* et *machine*

Enfin, des **acteurs** peuvent être associés aux tâches pour indiquer quels sont les utilisateurs autorisés à exécuter les tâches dans K-MAD(v1). Les acteurs ne peuvent être que des utilisateurs isolés (*individu*) cependant, les activités décrites peuvent faire intervenir des organisations. C'est pourquoi nous avons proposé d'ajouter dans K-MAD(v2), ce type d'utilisateur humain.

De même, s'il est possible de préciser les acteurs humains intervenant dans l'exécution des tâches, il n'est pas possible de préciser les acteurs système dans K-MAD(v1). K-MAD(v2) dispose d'entités *utilisateur système* (*machine* ou *système informatisé*) qui peuvent être associées aux tâches comme étant le(s) système(s) supportant l'exécution de la tâche. Cependant, ces ajouts dans K-MAD(v2) n'apportent aucune modification sur les données dans un modèle K-MAD(v1) puisqu'elles sont contenues dans K-MAD(v2).

4.4. Conclusion sur la méta-modélisation de K-MAD

Ce chapitre avait pour but de présenter les études que nous avons menées pour méta-modéliser le modèle K-MAD en EXPRESS. Nous avons pour cela procédé en trois étapes.

La première a consisté en l'analyse de plusieurs modélisations d'activité. Cette analyse a montré que bien que K-MAD propose des composants qui permettent d'accroître l'expressivité formelle des modèles de tâches (comme la définition formelle de pré conditions), l'utilisation de leur définition dans K-MADe a certaines limitations.

Répondre à ces limitations en proposant des extensions et des modifications au modèle existant constitue notre seconde étape. Ces propositions s'appliquent aux composants K-MAD pour lesquels nous avions préalablement identifié des difficultés d'utilisation : les tâches, les objets et les utilisateurs.

L'intégration de ces propositions dans K-MAD nécessite la modification d'une partie du modèle. Nous avons donc défini un modèle K-MAD modifié, appelé K-MAD(v2). La méta-modélisation de ce modèle a été faite dans un troisième temps. Bien méta-modélisées, ces modifications n'ont pas encore été intégrées à l'outil K-MADe. Une nouvelle version de cet outil est actuellement en cours de développement, elle s'appuiera sur le noyau que nous avons défini dans ce chapitre (K-MAD(v2)).

Une fois la méta-modélisation de K-MAD(v2) réalisée, nous avons décrit les transformations nécessaires à la retranscription d'un modèle K-MAD(v1) en un modèle conforme au nouveau méta-modèle.

Ce méta-modèle K-MAD(v2) (présenté dans son intégralité dans l'annexe Chapitre 9) sera utilisé afin de permettre l'échange de données entre un modèle K-MAD(v2) et des interacteurs hiérarchisés. Le chapitre 4 présente l'étude menée pour concevoir le méta-modèle des interacteurs hiérarchisés.

Chapitre 5. Méta-modélisation des Interacteurs Hiérarchisés

Le contexte d'utilisation auquel l'application (et donc le dialogue) doit être adapté peut être défini par le triplet <utilisateur, plate-forme, environnement>. Nous ne nous intéressons pas ici à une étude sur les moyens par lesquels les applications peuvent s'adapter aux différents contextes d'utilisation (automatiquement ou non) mais aux conséquences de l'expression d'un dialogue adapté par le concepteur.

Pour définir le méta-modèle d'un formalisme de dialogue permettant l'expression du modèle de dialogue d'applications répondant à différents contextes d'utilisation, nous avons procédé à des études de cas. Ces études de cas ne représentent pas l'ensemble des contextes d'utilisation possibles et leur étude doit donc être étendue.

Nous avons exprimé le dialogue de deux applications en utilisant le formalisme des interacteurs hiérarchisés qui est le formalisme que nous avons sélectionné (voir l'étude préliminaire dans 3.2). Ces études de cas ainsi que leur analyse sont présentées dans la première section (§5.1).

L'utilisation des interacteurs hiérarchisés a permis d'identifier des modifications à apporter pour adapter le formalisme à l'expression du dialogue de tout type d'applications interactives (rappelons qu'initialement, les interacteurs hiérarchisés étaient dédiés aux applications CAO [Texier 2000]). Ces modifications sont présentées dans la section 5.2. Elles ont amené à la méta-modélisation du formalisme des interacteurs hiérarchisés que nous proposons sous forme de modélisation EXPRESS (la méta-modélisation complète du formalisme est présentée en annexe (Chapitre 9).

5.1. Les études de cas

Les études de cas pour lesquelles un modèle de dialogue a été conçu sont deux des applications développées comme étude de cas pour l'étude de méta-modélisation de K-MAD (section Chapitre 4). Ces deux applications sont le jeu de *Mastermind* et le *mailer*.

Nous avons choisi en particulier ces deux applications car elles nous permettent de faire varier deux des paramètres influant sur la conception du dialogue : les types d'utilisateurs (profils) et les moyens d'interaction.

Nous souhaitons aboutir à une méthode de vérification entre les modèles de tâches et de dialogue ; les utilisateurs sont l'une des variables ayant une influence sur ces deux modèles. Dans ce contexte d'étude, faire varier les profils utilisateur est un moyen d'évaluer le pouvoir d'expression des modèles de dialogue sur l'un des points exprimé dans le modèle de tâches.

Au contraire, l'interaction utilisée dans l'application influence le dialogue (dans la mesure où c'est l'interaction qui permet le déclenchement des actions) mais ne doit pas modifier le déroulement des tâches que l'utilisateur veut pouvoir réaliser. Nous avons fait

évoluer les techniques d'interaction afin d'étudier l'impact de ce point sur la conception du modèle de dialogue.

Dans une première section (§ 5.1.1), nous complétons la description faite dans 4.1.1 des deux applications réalisées pour y préciser les éléments à prendre en compte par le concepteur du dialogue. Nous illustrerons de manière plus précise la conception du dialogue d'une de ces applications dans la section 5.1.2 : une application de jeu de Mastermind. Enfin, nous présenterons dans la section 5.1.3, les leçons que nous avons apprises de l'utilisation des interacteurs hiérarchisés pour la conception du dialogue de ces applications interactives. Les applications sont toutes conçues à partir de modèles de tâches K-MAD détaillés dans les annexes (Chapitre 9).

5.1.1. Description des études de cas

Mastermind. Le jeu de Mastermind est la répétition de parties qui ont chacune pour but la découverte d'un code. C'est un jeu pour deux joueurs : le *meneur* et le *chercheur*. Le *meneur* définit un code caché de quatre couleurs parmi six. Le *chercheur* va ensuite chercher à le découvrir. Une même couleur peut apparaître plusieurs fois dans un même code. Pour deviner le code secret, le *chercheur* compose des propositions de code (au plus huit). Une fois composée, chaque proposition est soumise à vérification. Le *meneur* indique alors le nombre de pions bien placés et le nombre de pions présents dans le code mais pas à la place proposée.

Nous avons conçu plusieurs versions d'application de Mastermind en proposant un jeu à un joueur (contre l'ordinateur) ou à deux joueurs et en modifiant les techniques d'interaction utilisées (interaction à base de boutons ou de manipulation directe), certaines présentations obtenues sont présentées sur la Figure 4.1.1.

Mailer. Le *mailer* que nous avons réalisé permet la gestion des emails reçus et envoyés. Là aussi, nous avons modifié l'interaction utilisée pour réaliser certaines actions comme l'ajout d'adresses de destinataire. Cependant, au contraire du cas du jeu de *Mastermind*, deux techniques d'interaction peuvent être disponibles en même temps pour accomplir la même action. Par exemple, l'utilisateur peut choisir d'ajouter une pièce jointe en utilisant des boutons et des fenêtres ou en déplaçant le fichier dans l'email (drag and drop).

5.1.2. Un exemple de mise en pratique du formalisme

Dans cette section, nous allons illustrer la conception du modèle de dialogue pour une partie du *Mastermind*. Cette conception est réalisée à partir du modèle de tâches présenté dans l'annexe (Chapitre 9).

Comme nous l'avons dit, nous avons choisi d'exprimer le dialogue sous forme d'interacteurs hiérarchisés. Les transitions des machines à états sont déclenchées par des jetons. Ces jetons peuvent être produits non seulement par des actions de l'utilisateur mais aussi par les transitions. Cependant, tous les éléments du formalisme ne sont pas représentés sur les figures pour une raison de visibilité du modèle. Nous n'avons pas fait apparaître les noms des états des machines à états du modèle de dialogue.

La conception du modèle a été faite en cinq étapes (en suivant le processus de conception itératif). Nous allons suivre ces étapes pour décrire cette conception.

5.1.2.1. Première étape

La première étape du processus de conception est celui de la conception du modèle de dialogue initial à partir de la description de l'activité telle qu'elle est stipulée dans les règles de jeu.

Ses règles de jeu exprimées sous forme de modèle de tâches permettent d'identifier les besoins expressifs du dialogue de l'application. Le modèle conçu est alors décrit. Enfin, nous présentons une vérification de la prise en compte des besoins identifiés dans le modèle exprimé.

Identification des besoins expressifs

La conception que nous présentons a pour point de départ le modèle de tâches présenté dans l'annexe (Chapitre 9). Celui-ci présente l'activité « Jouer au Mastermind » pour un joueur affrontant l'ordinateur (mode mono-joueur). À partir de ce modèle de tâches, le concepteur peut identifier quatre interactions principales entre le joueur et le système durant le déroulement d'une partie :

- le joueur débute un nouveau jeu (marque 1 sur la Figure 5.1.1)
- le joueur pose un pion (marque 2 sur la Figure 5.1.1)
- le système informe de l'évaluation (marque 3 sur la Figure 5.1.1)
- le système met fin à un jeu (marque 4 sur la Figure 5.1.1)

En plus de ce rôle de communication entre les acteurs de l'application (le joueur et le système), le contrôleur de dialogue doit gérer les droits d'action de l'utilisation :

- le joueur ne peut poser un pion sur la proposition n°j que si la proposition j-1 a été évaluée comme n'étant pas la combinaison à trouver et que $j < 9$
- le joueur ne peut placer un pion à la place i que si cette place est vide (nous considérons le cas du respect stricte des règles de jeu, il est donc impossible de modifier un pion posé)
- le joueur ne peut débuter un nouveau jeu que si aucun jeu n'est en cours

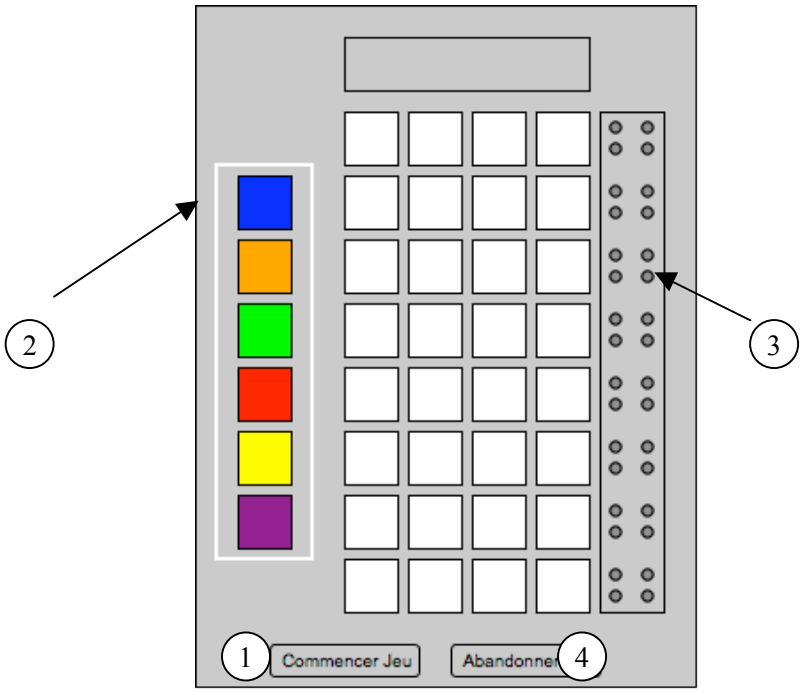


Figure 5.1.1 : Schéma d'une interface possible pour jouer au *Mastermind* en mode mono-joueur

Description du modèle

La première étape de la modélisation du dialogue est d'exprimer un contrôleur de dialogue respectant ces sept obligations. Le modèle de dialogue représenté sur la Figure 5.1.2 est un modèle d'interacteurs hiérarchisés que nous avons réalisé.

Il est composé de trois machines à états. La première machine à états (celle de plus haut niveau hiérarchique) (*AJeu*) est composée de deux états et de deux transitions. Ces deux transitions sont *debJeu* et *fin*. La transition *debJeu* génère la combinaison à trouver et produit un jeton *debCombi* indiquant qu'une combinaison peut être proposée. L'état en cours est alors modifié. De ce second état part la seconde transition de la machine à état : *fin*. Cette transition met fin au jeu et remet *AJeu* dans son état initial.

Le second état de *AJeu* correspond à l'état de « jeu en cours ». Le déroulement d'un jeu est composé de la proposition de plusieurs codes et c'est la gestion de ces propositions qui est modélisée par la machine à états *ACombi*. Cette machine à états est composée de deux états et de trois transitions. De l'état initial, une unique transition est exécutable : *debCombi*. Cette transition est déclenchée par un jeton *JdebComb* produit par la transition *debJeu* de *AJeu*. Cette transition met le système dans un état de décryptage en cours (second état de la machine à états) en produisant un jeton *JdebCode*. De cet état, deux transitions *prop* partent. Ces transitions ont deux états d'arrivée distincts définis par leur garde. La transition *prop* ne modifiant pas l'état courant a pour garde « *nombre de proposition inférieur ou égal à 8 et nombre de pions bien placés inférieur à 4* (la combinaison n'a pas été trouvée) ». Lorsque l'une de ces conditions (ou les deux), n'est pas vérifiée, le décryptage est terminé (soit parce que la combinaison a été trouvée, soit

parce que les 8 chances sont épuisées). La transition *prop* produit alors un jeton *Jfin* représentant la fin du jeu (consommé par *AJeu*).

Enfin, le dernier niveau de dialogue modélisé (*ACode*) représente la composition d'un code. Nous avons choisi de la représenter par une machine à états composée de cinq états et cinq transitions. La première stipule le commencement de la composition d'un code (par la consommation d'un jeton *debCode* produit par la transition *prop* de *ACombi*). Les trois états suivants représentent les états dans lesquels la combinaison en cours est composée d'1, 2 ou 3 pions placés. Chacun de ces états permet le positionnement d'un pion supplémentaire. Lorsque le 4ième pion est placé (*pion*) un jeton *Jprop* est produit et remet la machine à états *ACode* dans son état initial.

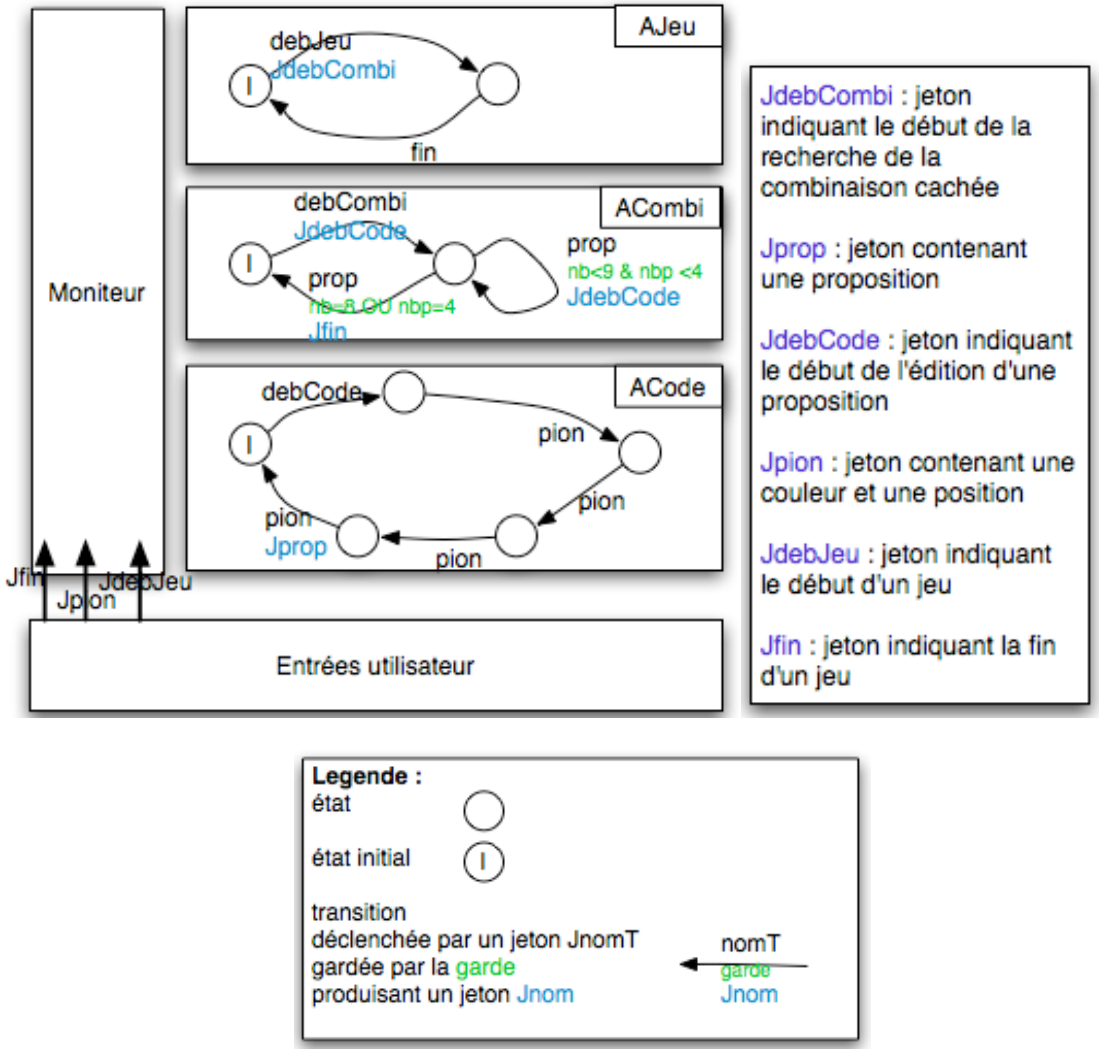


Figure 5.1.2 : Dialogue initial du jeu de Mastermind

Vérification de la prise en compte des besoins identifiés dans le modèle

Nous avons, au début de la conception du modèle de dialogue, identifié quatre interactions entre le joueur et le système. Trois d'entre elles sont des interactions du joueur sur le système :

- le joueur débute un nouveau jeu
- le joueur pose un pion
- le joueur met fin au jeu

Les interacteurs hiérarchisés représentent les actions des utilisateurs par des jetons produits par les utilisateurs et consommés par les machines à états. Il est donc nécessaire, à ce niveau de composition du modèle de tâches de s'assurer que des jetons venant de la couche utilisateur représentent ces actions et sont consommés dans les machines à états.

Dans la modélisation représentée sur la Figure 5.1.2, trois jetons sont produits par les utilisateurs : *JdebJeu*, *Jpion* et *Jfin*. *JdebJeu* est produit lorsque l'utilisateur ordonne le début d'un jeu (en cliquant sur le bouton *Commencer Jeu* de l'interface présentée sur la Figure 5.1.1 (marque 1) par exemple) et est consommé par *AJeu*. Lorsque l'utilisateur ajoute un pion à la combinaison en cours, un jeton *Jpion* est produit, il est ensuite consommé par *ACode*. Enfin, lorsque le joueur stoppe le jeu, il produit un jeton *Jfin* consommé par *AJeu*. Les trois interactions sont donc représentées dans le modèle conçu.

Trois conditions ont également été identifiées. Nous allons vérifier que ces trois conditions sont représentées dans le modèle de dialogue.

ACombi modélise la succession des propositions (transitions *prop*). Chaque transition *prop* déclenchée produit un jeton *JdebCode* qui est la première transition de *ACode* (elle permet donc de constituer un code). Une seule combinaison est donc éditable à la fois. Identifier quelle est la position de cette combinaison dans l'interface est du ressort du code réalisé par le système lors de l'exécution de la transition *prop*.

La seconde condition est l'application du même principe à la définition des pions dans la combinaison en cours. La machine à états *ACode* assure que les pions soient placés les uns après les autres et une seule fois. Ainsi, chaque pion n'est défini qu'une seule fois dans chaque combinaison. La seconde condition est donc vérifiée.

Enfin, la dernière condition (le joueur ne peut débuter un nouveau jeu que si aucun jeu n'est en cours) est gérée par la machine à états *AJeu*. Celle-ci différencie les états dans lesquels le jeu est en cours ou non. La transition *debJeu* est alors accessible uniquement dans l'état où le jeu n'est pas en cours.

5.1.2.2. Seconde étape

La première modification apportée au dialogue correspond à la modification de l'activité présentée dans le modèle de tâche. La première conception proposée suit le processus de constitution d'une proposition stipulé dans les règles de jeu. L'utilisateur ne peut donc pas modifier un pion placé (le supprimer ou le déplacer). Ce mode de

fonctionnement semble être difficile pour un joueur particulièrement lorsqu'il est novice. Les règles de jeu dans le dialogue conçu vont alors être modifiées afin de permettre la suppression et la modification d'un pion placé. La machine à états à modifier est donc *ACode*.

La nouvelle version de cette machine à états (*ACode2*) (voir Figure 5.1.3) consomme les jetons *debCode* produits par *ACombi* et produit des jetons *Jprop* comme la machine à états *ACode* proposée sur la Figure 5.1.2, les modifications n'affectent donc pas les machines à états *ACombi* et *AJeu*.

Cette machine à états est composée de quatre états et quatre types de transitions. Tout comme dans *ACode*, à partir du premier état, seule une transition peut être déclenchée : *debCode*. Les trois autres états expriment l'état lorsque aucun pion ne compose la combinaison en cours d'édition (*pas pion*), quand les 1, 2 ou 3 pions composent la combinaison en cours (*1,2,3 pions*) et quand la combinaison est complètement définie (*4 pions*). La suppression (*efface*) n'est possible qu'à partir des états *1,2,3 pions* et *4 pions* et l'évaluation d'une combinaison ne peut être réalisée (transition *valide*) qu'à partir de l'état *4 pions*. Un jeton *Jprop* est alors produit lors de l'exécution de la transition *valide*. Les états d'*ACode2* sont nommés pour faciliter leur désignation bien que les noms n'aient pas de rôle formel pour la modélisation. Les gardes des transitions *deplace*, *efface* et *pion* manipulent la variable *npion* contenant le nombre de pions placés dans la combinaison en cours d'édition.

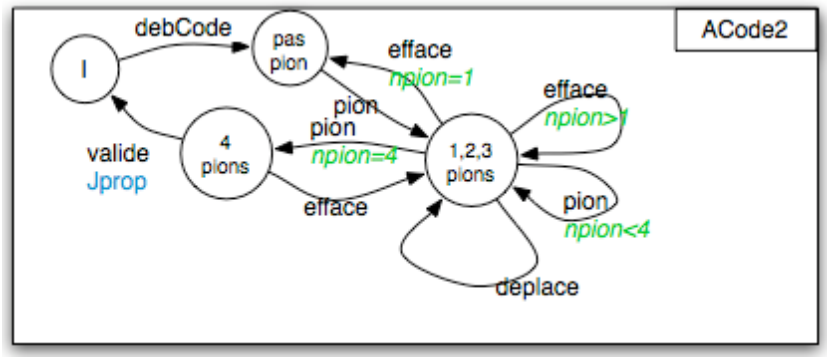


Figure 5.1.3 : La machine à états du second processus de définition d'une proposition

Cependant, en plus des jetons, *JdebCode* et *Jpion*, *ACode2* consomme des jetons *Jefface*, *Jdeplace* et *Jvalide*. Ces jetons sont produits par des actions de l'utilisateur. Nous les ajoutons donc à l'ensemble des jetons issus de la couche des entrées utilisateur (avec *Jfin*, *Jpion* et *JdebJeu*).

5.1.2.3. Troisième étape

La possibilité de modifier la combinaison en cours peut être considérée comme étant un mode d'entraînement. La seconde modification se présente dans le cas où le

concepteur souhaiterait permettre les deux modes de jeu (entraînement (modélisé par *ACode2*) et respect strict des règles (modélisé par *ACode*)). Les deux machines à états *ACode* et *ACode2* sont exprimés au même niveau d'abstraction. Le choix de l'un ou l'autre des modes doit être fait avant le début du jeu. Ce choix peut être exprimé par une transition *choixMode* bouclant sur l'état initial de *AJeu*. Les transitions *debCode* de *ACode* et *ACode2* sont alors assujetties à la vérification d'une garde testant le mode de jeu sélectionné.

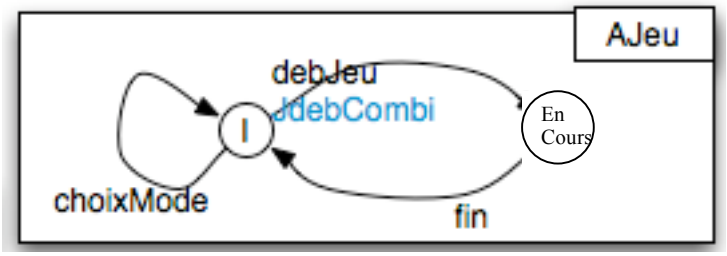


Figure 5.1.4 : La machine à état *AJeu* avec deux modes de jeu

5.1.2.4. Quatrième étape

Arrivé à ce niveau de conception, le concepteur souhaite spécifier l'interaction utilisée par les joueurs pour produire *Jfin*, *Jvalider, JdebJeu*, *Jpion*, *Jefface* et *Jdeplace*.

Jfin, *Jvalider* et *JdebJeu* sont des jetons produits par des actions sur des boutons dans notre exemple. Leur production ne nécessite donc pas l'ajout de machines à états supplémentaires. Au contraire, nous avons choisi d'utiliser la manipulation directe pour produire *Jpion*, *Jefface* et *Jdeplace*.

Les machines à états (*APlace* et *AModif*) les produisant sont présentées sur la Figure 5.1.5. La machine à états *APlace* représente le positionnement d'un pion de la combinaison en édition. La première transition de l'état initial (*debPion*) représente le début du placement d'un pion. La transition partant du second état et mettant le système dans l'état suivant est déclenchée par *JSelectCouleur* qui représente la sélection d'une couleur par l'utilisateur. Une fois la couleur du pion sélectionnée, l'utilisateur peut le déplacer (transition *bouge*). L'état courant n'est pas modifié. Enfin, lorsque l'utilisateur pose le pion à un emplacement libre (et produit *JPion*), l'état courant revient à l'état initial.

La machine à états *AModif* suit le même procédé. Cependant, l'action réalisée est définie par la dernière transition (*pose*) qui détermine une suppression d'un pion (production de *JEfface*) ou le déplacement d'un pion (produit de *JDeplace*) en fonction de l'emplacement du dépôt du pion sélectionné.

Une fois ces machines conçues, il est nécessaire de faire le lien entre ces machines à états et la machine de niveau hiérarchique supérieur *ACode2*. Un pion peut être placé dans les états *pas de pion* et *1,2,3 pions* de *ACode2*. Les transitions permettant d'atteindre

ces états doivent donc produire un jeton *debPion* (1er jeton consommé par *APlace*) lorsque la machine à états *APlace* n'a pas encore quitté l'état initial (*debCode*, *pion* et *efface*).

Les jetons *Jefface* et *Jdeplace* ne sont attendus que lorsque au moins un pion compose la combinaison en cours. *debModif* (qui est la première transition de *AModif*) doit donc être déclenchée lorsque l'état de *AModif* est toujours l'état initial (*pion* de *pas de pion*, *deplacer*, *effacer* de *1,2,3 pions* de *ACode2*). Le Tableau 5-1 résume les jetons produits par les transitions de *ACode2*.

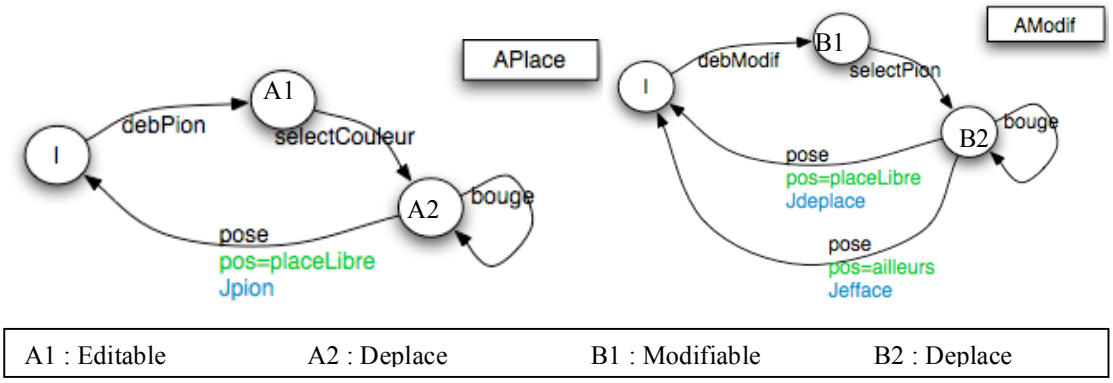


Figure 5.1.5 : Les machines à états pour la manipulation directe de l'édition des pions dans la combinaison en cours

Transition	Etat de départ	Etat de fin	Jetons déclenchées
debCode	I	pas pion	JdebPion
pion	pas pion	1,2,3 pions	JdebPion JdebModif
efface	1,2,3 pions	pas pion	JdebPion
efface	1,2,3 pions	1,2,3 pions	JdebModif
pion	1,2,3 pions	1,2,3 pions	JdebPion
deplace	1,2,3 pions	1,2,3 pions	JdebModif
pion	1,2,3 pions	4 pions	
efface	4 pions	1,2,3 pions	JdebPion JdebModif
valide	4 pions	I	Jprop

Tableau 5-1 : Les jetons déclenchés par les transitions de *ACode2*

5.1.2.5. Cinquième étape

La dernière modification apportée vise à inclure le dialogue modélisé dans un dialogue d'une application à deux joueurs (le *meneur* est joué par le second joueur). Pour modéliser la gestion d'une partie, nous proposons la machine à états présentée sur la Figure 5.1.6. Cet automate *AJeu2* utilise les jetons produits par les niveaux d'abstractions inférieurs (*ACombi*, *ACode* et *ACode2*) sans qu'il soit nécessaire de les modifier.

De l'état initial de *AJeu2*, la transition *debJeu* modifie l'état courant et permet la définition d'un code (production de *JdebCode*). La production du code à trouver suit le même procédé que pour la définition d'un code lors de la recherche du code secret. Les machines à états *ACode* et *ACode2* peuvent donc être utilisées pour produire le code à découvrir. À partir de l'état atteint, la transition *fin* permet de mettre fin à la manche menée par le première joueur. La même séquence de transition (*prop* et *fin*) modélise le dialogue de la manche du second joueur. L'état revient donc à l'état initial et un second jeu peut débuter.

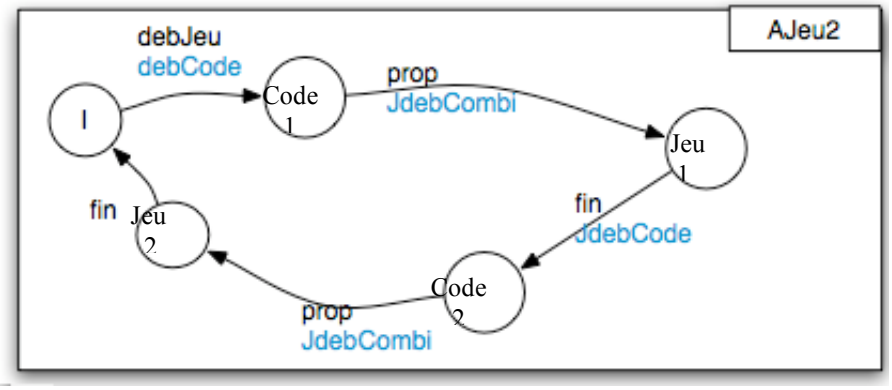


Figure 5.1.6 : Machine à états d'un jeu à deux joueurs

5.1.3. Leçons apprises des études de cas

L'utilisation du formalisme des interacteurs hiérarchisés pour exprimer le dialogue des applications de jeu de *Mastermind* et de *mailer* nous a permis de constater les avantages et les limites à son utilisation. Nous avons identifié quatre avantages à l'utilisation des interacteurs hiérarchisés comme formalisme pour modéliser le dialogue : les bénéfices liés à la modularisation, la facilité d'intégration des modifications, l'adaptation aux raffinements de la conception et l'expression du dialogue indépendamment du type d'interaction utilisée. En plus de ces avantages, nous avons noté la nécessité d'apporter quelques modifications au formalisme défini initialement.

Le premier bénéfice de l'utilisation des interacteurs hiérarchisés pour exprimer le dialogue est sa **modularisation**. Les bénéfices de la division du code sont les mêmes que pour toutes les modularisations mises en place dans le domaine technique, c'est-à-dire la définition de module dédié à chaque partie spécifique et la réutilisation des modules.

En effet, l'un des premier attraits de la modularisation est la décomposition d'un élément. Cette décomposition permet de diviser la taille de l'élément et également de développer des méthodes adaptées à chacun des composants obtenus. Par exemple, l'utilisation de modèles d'architectures (voir § 2.2.3.2) permet d'organiser le code de manière à développer des méthodes adaptées à chaque partie (comme la partie dédiée au dialogue).

De plus, la modularisation permet la réutilisation d'un ou plusieurs des modules pour différentes applications pour peu qu'ils soient assez génériques. Par exemple, un module dédié à la gestion du dialogue d'une session de jeu à 2 (comme *AJeu2* dans notre exemple 5.1.2) peu être réutilisé pour la plupart des applications de jeu à 2. La définition d'automates regroupés par niveau hiérarchique permet d'exprimer le dialogue sous forme de modules se focalisant sur un niveau hiérarchique. Notons que si nous avons confirmé cet avantage avec le formalisme des interacteurs hiérarchisés, nous aurions pu le constater avec d'autres formalismes permettant la décomposition du dialogue.

L'utilisation d'une structure hiérarchique dans la modélisation du dialogue permet de l'organiser par niveaux d'abstraction. D'ailleurs comme nous l'avons indiqué dans 3.2, les formalismes des interacteurs hiérarchisés et des machines à états hiérarchiques proposent explicitement de baser la décomposition sur les niveaux d'abstraction.

Cette décomposition par niveaux hiérarchiques illustre un second apport pour exprimer le dialogue : la **centralisation des modifications à apporter pour intégrer les modifications de spécifications**. En effet, comme nous l'avons dit précédemment, la conception des applications interactives est itérative. Cet aspect itératif est en partie dû à la modification des spécifications des besoins utilisateurs en cours de conception. Dans une conception centrée-utilisateur, ces spécifications sont en partie exprimées sous forme de modèles de tâches, donc hiérarchiquement. La définition de modules dialogue-tâches basés sur les niveaux hiérarchiques permet d'isoler des modifications à apporter. Sur l'exemple présenté sur la Figure 5.1.2, si une modification est apportée sur le modèle de tâches concernant la composition d'une proposition (comme l'ajout de la possibilité de modifier un pion placé), seule la partie du dialogue associée à son niveau d'abstraction (*ACode*) est à modifier dans le modèle de dialogue.

En plus de l'itération due aux différentes modifications apportées par les spécifications utilisateur, les étapes du processus de conception sont répétées pour y inclure les précisions apportées au fur et à mesure de la conception.

De par leur définition [Norman et Draper 1986, Sebillotte 1994, Sebillotte et Scapin 1994] les niveaux hiérarchiques des modèles de tâches sont conçus en fonction des compléments d'informations apportés. L'aspect hiérarchique d'un modèle de dialogue permet **d'adapter la conception du dialogue aux différents raffinements de la conception du modèle de tâches**.

Enfin, les modèles de tâches sont définis pour exprimer les activités à partir de données collectées. Ces données n'impliquent pas nécessairement la description des interactions utilisées. Cependant, les approches qui étudient la conception du dialogue à partir des modèles de tâches ajoutent les informations sur l'interaction afin de faire le lien entre les tâches et les transitions du dialogue (comme [Luyten, et al. 2003]). L'ajout de ces informations peut être source d'erreurs qui ne sont pas liées au contrôle du dialogue tel qu'il est défini dans [Pfaff 1985] (voir § 3.3).

La description d'un dialogue hiérarchique intègre les techniques d'interaction à un niveau d'abstraction qui ne dépend pas de la définition des modèles de tâches. Par exemple, un niveau d'abstraction du dialogue qui permet de passer d'une interaction basée sur l'utilisation de boutons à une interaction utilisant la manipulation directe. De ce fait, la

validation des modèles de dialogue en fonction des modèles de tâches ne dépend pas de l'interaction utilisée (dont la validation peut faire l'objet d'études dédiées). L'interaction utilisée pour produire les jetons correspondant aux entrées utilisateur est spécifiée par les niveaux inférieurs lorsque cela est nécessaire comme pour le placement d'un pion en utilisant la manipulation directe, ces niveaux sont extérieurs au modèle de dialogue.

L'un des grands intérêts du modèle des interacteurs hiérarchisés réside dans son faible couplage. Afin de conserver un niveau d'indépendance maximum, la communication entre les différents niveaux d'abstraction est assurée par l'utilisation de jeton. Le *moniteur* gère la communication entre les différents niveaux hiérarchiques. Lorsqu'un jeton est produit par un automate, le moniteur le propose à l'automate de niveau directement supérieur, puis, s'il n'est pas consommé, il le propose à celui du dessus.

Dans la version initiale du formalisme des interacteurs hiérarchisés, la transmission des jetons est effectuée de bas en haut et si le jeton n'est pas consommé par l'un des automates lorsqu'il est arrivé au dernier niveau d'abstraction, il est perdu (il ne sera pas consommé).

Ce mode de distribution nous a posé des difficultés lors de la conception des modèles. En effet, nous avons constaté que si la **communication** de bas en haut est la plus courante, afin de transmettre certaines informations entre les automates, il nous a parfois été nécessaire de permettre la communication de haut en bas. Par exemple, pour le jeu de Mastermind, le plus bas niveau correspond à la définition (pour l'utilisateur) d'un pion (abstraction au niveau pion), le second niveau, à la composition d'un code (abstraction au niveau combinaison) et le plus haut niveau à la gestion d'une partie de jeu (abstraction au niveau partie de jeu). Il est alors nécessaire que les jetons soient transmis de bas en haut (une partie est composée de plusieurs propositions de combinaisons qui sont elles-mêmes composées du positionnement de quatre pions) mais également de haut en bas (le positionnement des pions dépend de l'évaluation de la précédente combinaison proposée).

Dans le formalisme des interacteurs hiérarchisés, les jetons manipulés sont de deux types : les jetons *Commande* et les jetons *Données*. Ces deux types de jetons permettent l'expression du lien entre le noyau fonctionnel et le contrôleur de dialogue. Ainsi, les jetons *Commande* représentent les fonctions du noyau fonctionnel et les jetons *Données* les données utilisées pour exécuter ces fonctions. Par exemple, la fonction d'évaluation d'une combinaison a besoin d'une proposition-combinaison en donnée.

Cependant, à partir de l'exemple décrit dans la section 5.1.2, nous avons également identifié l'existence d'un troisième type de jetons, les jetons utilisés pour la communication entre les différentes machines à états. Par exemple, dans le dialogue du jeu de Mastermind représenté sur la Figure 5.1.2 *debProp* est un jeton qui indique le début des propositions. Ce jeton lie *AJeu* (qui le produit) et *Aprop* (qui le consomme) pour indiquer que c'est *AJeu* qui autorise la définition d'une proposition. Ces jetons sont nécessaires car nous avons choisi de représenter les liens d'autorisation des niveaux hiérarchiques par des jetons. Un autre possibilité aurait été d'ajouter ce service à l'ensemble des services proposés par le contrôleur.

Bien que l'exemple que nous avons décrit dans 5.1.2 ne l'illustre pas, le formalisme des interacteurs hiérarchisés propose deux services : la *gestion de l'activation et de la désactivation des commandes* et la *possibilité de l'annulation des actions réalisées* (*Undo*).

Le contrôle « gendarme » consiste à n'autoriser que les commandes attendues à chaque moment de l'utilisation du système. Pour cela, le moniteur connaît l'ensemble des jetons attendus par tous les automates du dialogue. Pour chacun d'entre eux, il faut alors qu'il fasse en sorte que leurs productions soient permises. Ceci est réalisé par deux fonctions : une qui fait le lien entre le contrôleur et la présentation (dans l'implémentation du contrôleur) et la seconde qui active les commandes (dans l'implémentation de la présentation de l'application).

Dans [Texier 2000], le « Undo » est géré par l'enregistrement systématique de tous les jetons joués dans une liste et par le rejeu de tout ces jetons sauf du dernier une fois les machines à états remises dans leur état initial. Le rejeu est déclenché par la production d'un jeton particulier : *JUndo* est reçu. Le rejeu des actions est systématiquement possible. Ce mécanisme n'est pas adapté à toutes les applications interactives dès lors que le résultat d'une fonction peut être aléatoire. Par exemple, le début d'un jeu (*debJeu* sur la Figure 5.1.2) implique la définition de la combinaison à trouver. Cette définition est aléatoire ainsi, à chaque fois que *debJeu* est exécutée, une nouvelle combinaison est définie. Il n'est donc pas possible d'utiliser le procédé proposé pour implanter le *Undo* (rejeu de tous les jetons sauf le dernier à partir de l'état initial).

De plus, le retour en arrière après certaines actions doit être prohibé comme après l'évaluation d'une proposition ou l'envoi d'un email (il est alors techniquement impossible de le « rattraper »).

Enfin, l'utilisation des applications conçues a montré que des actions mettant en scène plusieurs automates pouvaient être répétées. La fin de l'exécution de l'action à son plus haut niveau (comme la composition d'une combinaison) implique alors nécessairement la réinitialisation de tous les automates de niveaux inférieurs. Par exemple, si deux moyens permettent la définition du placement d'un pion (comme textuellement et à l'aide de boutons) et que la constitution d'une combinaison est terminée alors les deux automates (d'un même niveau hiérarchique) qui permettent la production d'un *Jpion* doivent être remis à leur état initial.

Donc, l'initialisation d'un automate implique l'initialisation des automates de niveau inférieur dans ce modèle. Le moniteur doit prendre en charge l'initialisation des automates en initialisant tous les automates qui sont hiérarchiquement inférieurs à l'automate s'initialisant (si l'état courant de l'automate de niveau 3 revient à l'état initial alors les états courants des automates de niveau 2 et 1 sont également mis dans leurs états initiaux).

5.2. Modifications et Expression du méta-modèle

L'étude présentée dans 5.1 a mis en lumière des points d'amélioration du formalisme de modélisation du dialogue. Les modifications apportées pour accroître

l'expressivité du formalisme vont être présentées dans cette section. Elles concernent les *jetons* du formalisme et les services.

Les modifications apportées ont amené à une méta-modélisation du formalisme des interacteurs hiérarchisés adaptée au dialogue de toutes les applications interactives.

C'est pourquoi, nous allons dans cette section, présenter la méta-modélisation des jetons et des transitions. L'ensemble de la méta-modélisation des interacteurs hiérarchisés se trouve en annexe (Chapitre 9).

5.2.1. Les jetons

Suite à notre étude sur des cas de conception d'application, les modifications apportées aux formalismes sur les jetons concernent la distribution des jetons par le moniteur et les types de jetons disponibles dans le modèle.

5.2.1.1. La transmission des jetons

Avec les interacteurs hiérarchisés, le dialogue est exprimé par des automates organisés par niveau d'abstraction. Chaque niveau étant alors composé d'un ou plusieurs automates. Les transitions dans les machines à états sont déclenchées par des jetons transmis par le moniteur. Comme nous l'avons indiqué dans la section précédente, le sens de circulation proposé dans le formalisme des interacteurs hiérarchisés (de bas en haut) n'est par suffisant pour exprimer tous les échanges d'information entre les machines à états.

Nous proposons de modifier la circulation des jetons pour permettre la communication des automates de hauts niveaux vers ceux de bas niveaux en définissant la circulation comme cyclique. Cependant, pour ne pas prendre le risque d'une circulation infinie (dûe à la production d'un jeton non consommable), chaque jeton sera détruit lorsqu'il est présenté à l'automate qui l'a produit. De plus, l'échange des jetons des automates de haut niveau vers les automates de bas niveau étant moins courant que celui de bas vers haut, c'est ce dernier sens de circulation que nous avons privilégié.

Une fois proposé à l'automate de plus haut niveau, si celui-ci ne le consomme pas, le moniteur le renvoie au niveau le plus bas. Les jetons sont ainsi proposés jusqu'à ce que l'automate qui en a besoin le consomme pour déclencher sa prochaine transition ou qu'il atteigne l'automate qui l'a produit.

Ainsi, dans le cas présenté sur la Figure 5.2.1 (qui représente la communication des jetons dans le dialogue de Mastermind), un jeton produit par l'un des automates de *ACode* sera successivement présenté aux automates de *ACombi* et *AJeu* avant d'être abandonné si aucun des automates ne l'a consommé. Notons que les cas où un jeton est abandonné sans

avoir été consommé représente un cas d'exécution pour lequel le concepteur a commis une erreur de conception (oubli de prendre en compte une action possible de l'utilisateur, par exemple).

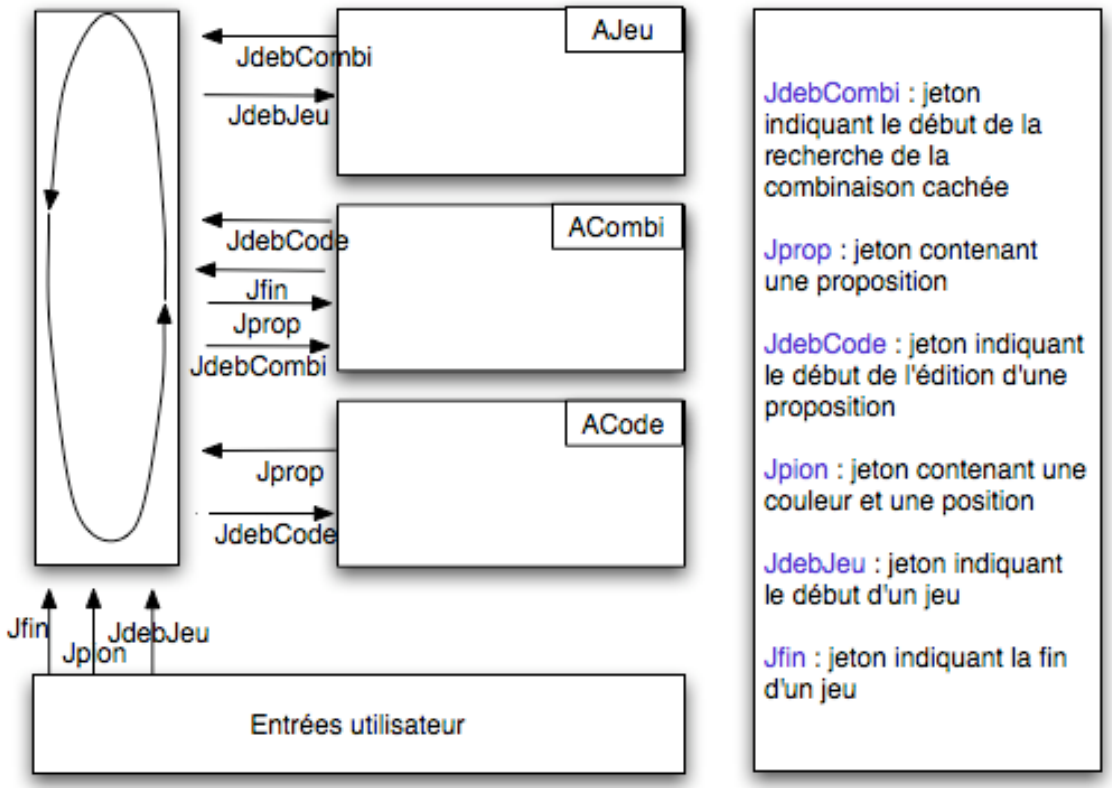


Figure 5.2.1 : Circulation des jetons dans le dialogue du jeu de Mastermind

5.2.1.2. Les types de jetons

Nous avons identifié un type de jeton supplémentaire qui n'a pour rôle ni de faire le lien avec le noyau fonctionnel, ni de représenter une entrée utilisateur, mais qui permet la communication entre les différents automates. C'est le cas du jeton *JdebCode* qui communique aux autres niveaux d'abstraction la poursuite ou non de la partie en fonction de l'évaluation de la dernière proposition et du nombre de propositions ayant été faites.

Ces informations peuvent être transmises par le moniteur ou par des jetons. Nous avons choisi cette seconde possibilité pour conserver le maximum d'indépendance entre les niveaux hiérarchiques.

Trois types de jetons sont donc définis dans le formalisme : les jetons *Commande* (*jetonOrdre* sur la Figure 5.2.2), les jetons *Données* (*jetonDonnée* sur la Figure 5.2.2) et les jetons *Communication* (*jetonCom* sur la Figure 5.2.2). Quelque soit le jeton, il a un nom et un type. Cependant, un jeton sans type n'a pas d'existence, c'est pourquoi l'entité *jeton* est

abstraite et instanciée une fois son type connu. Seuls les jetons *Données* contiennent des informations modélisées sous forme de *variables*.

Le diagramme EXPRESS-G exprimant la méta-modélisation des jetons dans le modèle des interacteurs hiérarchisés est représenté sur la Figure 5.2.2.

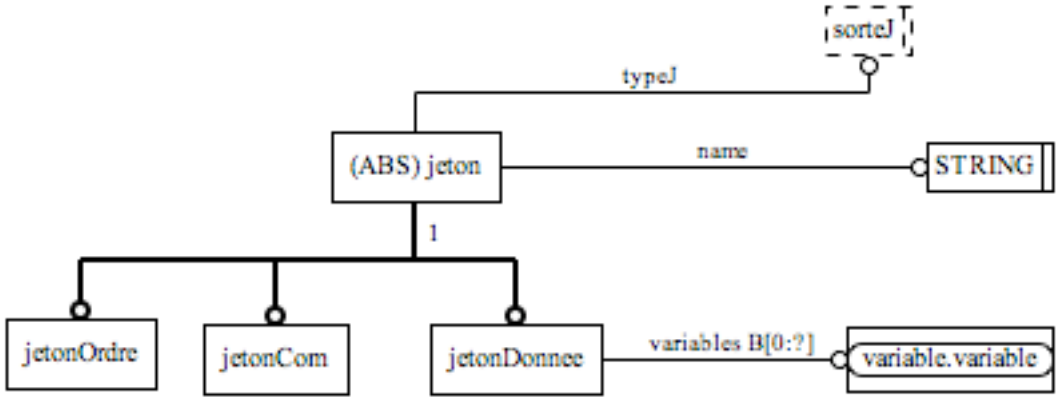


Figure 5.2.2 : Schéma EXPRESS-G de l'entité *jeton*

5.2.2. Les services

L'un des rôles du modèle de dialogue est de permettre l'expression du contrôle des commandes de l'application. C'est pourquoi, en plus de la communication entre les différents niveaux hiérarchiques du dialogue, le modèle propose trois services au dialogue : la gestion de l'activation et de la désactivation des commandes (nommée dans la suite sous le nom de contrôle « gendarme »), le « undo » et l'initialisation du contrôleur. Ces trois services sont intégrés dans le dialogue.

Le Tableau 5-2 résume les services proposés par le formalisme et leur signification.

Gendarme	Gère l'activation et la désactivation des commandes en fonction des jetons attendus par toutes les transitions du contrôleur de dialogue
Undo	Gère la commande spécifique d'annulation
Initialisation	Permet de mettre les machines à états dans un état cohérent lors de l'initialisation (retour à l'état initial) d'une d'entre elles

Tableau 5-2 : Les services proposés dans les interacteurs hiérarchisés

5.2.2.1. Le contrôle « gendarme »

Ce service impose que le concepteur établisse le lien entre le contrôleur de dialogue et les autres parties de l'application (présentation et noyau fonctionnel). Nous proposons

que ce lien soit spécifié par la *traduction* des actions utilisateur en jeton et des jetons en activation (*enable*). Cette traduction est précisée hors du contrôleur de dialogue comme dans la présentation par exemple (c'est l'implémentation que nous avons choisie de faire dans nos études de cas).

Par exemple, pour le jeu de Mastermind un extrait de l'implémentation de la traduction des actions utilisateur en jeton et des jetons en activation de commande est présenté sur la Figure 5.2.3.

a)

```
bnouveau = new JButton("Commencer Jeu");

bnouveau.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e){
        //Production d'un jeton debJeu transmit
        //au controleur
        control.jetonPlayed(new JDebJeu());
    }
});
```

b)

```
public void activerJDeb(boolean b){
    bnouveau.setEnabled(b);
}

public void refresh(Class c){

    if (c==JDebJeu.class){
        this.activerJDeb(true);
    }
    …
}
```

Figure 5.2.3 : Extraits de code établissant le lien entre la présentation est le contrôleur de dialogue du Mastermind

5.2.2.2. Le retour en arrière « Undo »

Le service de retour en arrière proposé dans le formalisme initial s'applique à toutes les transitions. Nous souhaitons permettre de préciser ces transitions. Nous souhaitons permettre la précision des transitions pouvant être annulées et celle ne le pouvant pas.

Le retour en arrière est pris en compte en spécifiant les transitions qui peuvent être « annulées ». Ces transitions possèdent les mêmes caractéristiques que les autres (états de départ et d'arrivée, garde…) mais elles possèdent en plus une action qui est réalisée lorsqu'un jeton spécifique (de type *JUndo*) les déclenche (lien avec la fonction de retour en arrière qui doit être prévue dans le noyau fonctionnel [Fekete 1996]).

La liste des jetons joués (dans la version initiale du formalisme) est alors remplacée par la liste des transitions annulables jouées. Cette liste est remise à vide dès qu'une transition non annulable (n'ayant pas de fonction *undo* correspondante) est déclenchée.

Tout comme un jeton, une transition est une entité abstraite. Elle n'est instanciée qu'une fois son caractère *annulable* défini. Cependant, qu'elle soit annulable ou non, une transition est définie par ses états de départ et d'arrivée, le jeton qui la déclenche (*triggered*), la condition à laquelle elle est soumise (*guard*) et l'action qui est réalisée.

Cette action peut avoir comme conséquence la production d'un jeton (*product*). En plus de ces caractéristiques, une transition annulable a une action qui est réalisée lorsque la transition est annulées. La Figure 5.2.4 illustre le schéma EXPRESS-G de la méta-modélisation de l'entité *transition* des interacteurs hiérarchisés.

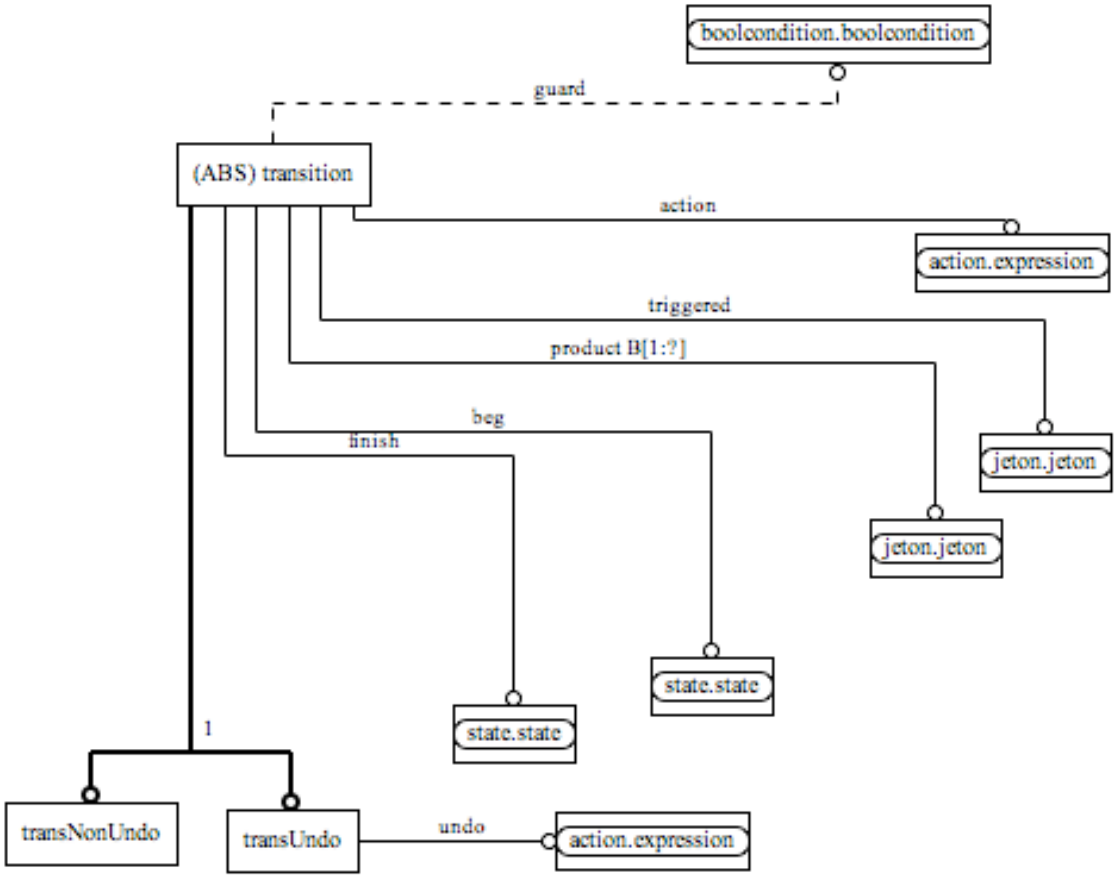


Figure 5.2.4 : Schéma EXPRESS-G de l'entité *transition*

5.2.2.3. L'initialisation

Afin de s'assurer que l'initialisation d'une machine à états d'un niveau hiérarchique implique obligatoirement l'initialisation des machines à états des niveaux hiérarchiques inférieurs, nous proposons d'ajouter cette fonctionnalité au moniteur des interacteurs hiérarchisés. Celle-ci est une action réalisée chaque fois qu'un état initial est atteint.

5.3. Conclusion sur la méta-modélisation des interacteurs hiérarchisés

Notre approche propose d'utiliser les méta-modèles des modèles de tâches et de dialogue (K-MAD et les IH) pour permettre l'échange de données entre ces deux modèles lors de la validation en conception. C'est pourquoi nous avons cherché à réaliser le méta-modèle des interacteurs hiérarchisés. Nous avons présenté dans cette section l'étude ayant été menée pour aboutir à cette méta-modélisation en EXPRESS.

Nous avons procédé à l'aide de deux études de cas pour analyser les modifications à apporter pour adapter le formalisme des interacteurs hiérarchisés à l'expression du dialogue d'applications interactives non CAO. Ces études de cas ont été conçues à partir des modèles de tâches correspondants et en utilisant une bibliothèque en développement *dialogHierarchy* pour implémenter le dialogue exprimé sous forme d'interacteurs hiérarchisés. Cependant, de par notre démarche (par étude de cas), nous n'avons pas couvert l'ensemble des applications interactives.

Des besoins spécifiques à certains types d'applications interactives, peuvent de ce fait nécessiter un nouvel enrichissement du modèle des interacteurs hiérarchisés. Cependant, nous avons tout de même défini certaines modifications à apporter et nous les avons intégrées au méta-modèle que nous proposons.

Nous avons également montré les avantages de l'expression du dialogue avec le formalisme des interacteurs hiérarchisés. Les deux premiers bénéfices identifiés ont été illustrés dans la section 3.2 : la **réutilisation** des machines à états conçues et l'**adaptation à la conception itérative**.

L'exemple de conception que nous avons présentée a montré la facilité d'intégrer, à chaque étape, de nouvelles modifications correspondant à l'ajout de précisions apportées par l'utilisateur, l'ajout de nouvelles fonctionnalités (souhaitées par l'utilisateur ou le concepteur). Ces ajouts sont facilement intégrables dans le modèle grâce au double mécanisme de la hiérarchisation et des jetons.

Lors de la prise en compte de ces modifications, il est également apparu que l'utilisation d'une communication des niveaux hiérarchiques, à base de jetons, permettait de réutiliser des modèles de dialogue déjà définis. Par exemple, lors de la conception d'un *mailer* permettant une gestion des emails reçus et l'envoi des emails, il est possible de réutiliser le modèle de dialogue d'un mailer ne permettant que l'envoi des emails. Cependant, il reste à étudier les différentes combinaisons possibles entre les modèles-composants.

L'utilisation de jetons produits à un niveau hiérarchique pour déclencher les transitions des niveaux supérieurs permet également de définir des **niveaux indépendants de l'interaction utilisée**. Il en est de même lorsque la présentation est modifiée (pour s'adapter à deux plateformes différentes par exemple), dès lors que la présentation permet la production des jetons attendus au plus bas niveau du modèle de dialogue, celui-ci reste le même.

Le processus de vérification n'est donc pas à reproduire. Cela permet la réutilisation du modèle de dialogue défini pour différents modèles d'interaction et différentes présentations d'une même application.

Chapitre 6. Utilisation des méta-modèles

Notre objectif final consiste à proposer une base pour la conception et la validation du dialogue des applications interactives s'appuyant sur les modèles de tâches. Nous avons montré précédemment les limites des approches purement génératives, tant dans la nécessité d'ajouter aux modèles de tâches des notions qui ne correspondent pas à leur niveau d'abstraction, que dans l'inadéquation de cette génération avec les pratiques de conception itérative, courantes, voire nécessaires, en conception centrée-utilisateur.

Notre approche consiste à étudier dans quelle mesure des règles peuvent être établies entre modèles de tâches et modèles de dialogue. Au lieu d'une relation bijective entre éléments de ces modèles, nous souhaitons établir des règles de cohérence entre ces mêmes modèles. Pour cela, nous avons construit le méta-modèle de chacun des modèles que nous avons étudiés (chapitres 4 et 5), et nous étudions dans ce chapitre les règles d'équivalence et de transformation entre ces mêmes modèles, à travers leurs méta-modèles.

Ces règles peuvent être utilisées pour générer un squelette de dialogue à partir du modèle de tâche, comme les approches génératives, mais peuvent également être utilisées pour valider le modèle de dialogue réalisé par rapport au modèle de tâches prescrites. Nous avons ainsi expérimenté un cycle complet de conception, itératif et incrémental, mettant en œuvre une bonne part des règles que nous avons établies.

La première étape de notre étude consiste à établir un lien formel entre les deux modèles, permettant d'associer chaque partie du modèle de tâche K-MAD(v2) avec la partie correspondante du modèle de dialogue IH correspondante. Ce lien repose sur la nature hiérarchique de ces deux modèles. Chaque niveau hiérarchique correspond à un niveau de raffinement de l'itération de la conception. En associant les niveaux hiérarchiques des deux modèles, la vérification de la cohérence des modèles pourra ainsi être réalisée tout au long du processus de conception.

À partir de ce principe, nous établissons des relations entre les composants des deux modèles. La définition de ces liens ainsi que leur justification est présentée dans la section 6.1.

L'établissement de ces liens entraîne certaines considérations, liées au fait que les formalismes hiérarchiques permettent l'expression de modèles équivalents quoique différents. Nous présentons ces cas ainsi que les méthodes que nous avons appliquées pour les circonscrire dans la section 6.2. La section 6.3 présente l'utilisation des liens pour définir les règles de cohérence entre les deux modèles. Enfin, nous montrons la mise en pratique de ces règles dans le processus de conception (section 6.4).

6.1. Liens d'association entre modèles de tâches et de dialogue

Pour permettre la vérification du dialogue des applications interactives par rapport au modèle de tâches exprimant les spécifications des besoins utilisateur, il nous faut

associer les modèles de dialogue et de tâches. Nous proposons d'utiliser la structure hiérarchique comme principe de base du lien entre ces deux modèles (voir la section 6.1.1).

Quatre associations ont ensuite été identifiées. La première association considérée, entre *tâches* et *transitions* (section 6.1.2) est la plus exploitée dans les travaux de conception du dialogue à partir du modèle de tâches (voir section 2.3.2). Elle constitue la base du lien entre les deux modèles. La seconde exploite l'aspect de *spécification*, qui caractérise la décomposition hiérarchique des modèles. Il est ainsi possible de lier une tâche décomposée du modèle K-MAD(v2) avec un automate du modèle de dialogue (section 6.1.3). Son utilisation est très différente des travaux effectués avec CTT, en raison du fait que, dans ce formalisme, les opérateurs ne sont pas associés à la décomposition hiérarchique des tâches.

Ces travaux utilisent une notation qui ne définit pas formellement les objets, et donc ne les utilise pas formellement. Au contraire, les objets de K-MAD sont formels et sont utilisés formellement dans les expressions. Cette définition formelle nous permet de définir les deux derniers liens, entre les objets du modèle de tâches et les variables du modèle de dialogue d'une part (section 6.1.4), et entre expressions de chacun des modèles d'autre part (section 6.1.5).

L'expression de ces liens entre modèles peut également être méta-modélisée. Cette dernière, exprimée en langage EXPRESS, est fournie en annexe (Chapitre 9).

6.1.1. Philosophie générale : s'appuyer sur les niveaux hiérarchiques

K-MAD propose une décomposition hiérarchique des tâches (suivant le processus de décomposition des actions proposé par Norman [Norman et Draper 1986]) alors que le formalisme des Interacteurs Hiérarchisés permet de structurer hiérarchiquement le dialogue. Les deux modèles étant exprimés sous forme hiérarchique ce qui incite à exploiter ces niveaux hiérarchiques pour définir des liens entre eux. L'association entre niveaux de l'arbre de tâche et machines à états du même niveau d'abstraction (interacteurs) permet de définir des modules de même niveau d'abstraction. Une illustration de modules définis en utilisant les niveaux hiérarchiques comme lien entre les deux modèles est présentée sur la Figure 6.1.1.

Comme nous allons le montrer, cette association *dialogue-tâche* par modules hiérarchiques apporte des intérêts pour la conception itérative des applications interactives.

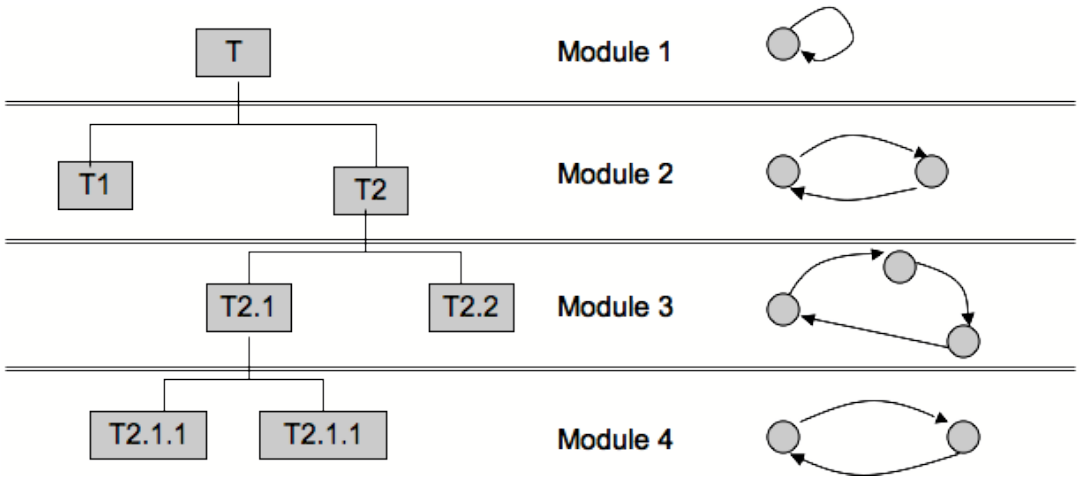
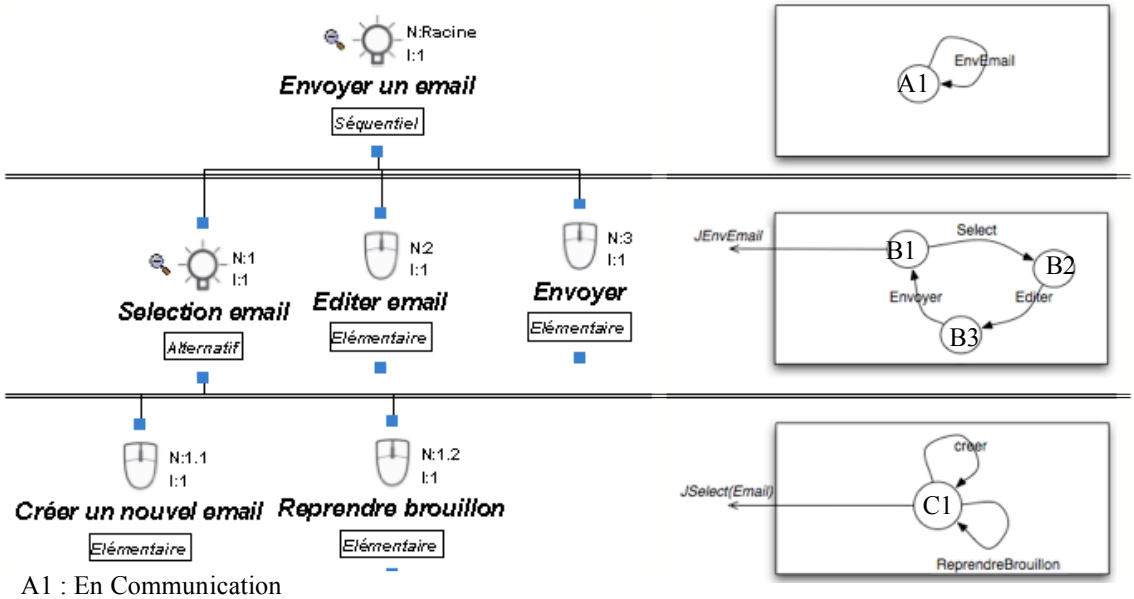


Figure 6.1.1 : Modules dialogue-tâches

Pour illustrer cela, reprenons l'exemple du dialogue conçu dans la section 5.1.2. Le modèle initial correspond au modèle de tâche représenté sur la Figure 6.1.2.a. Pour notre exemple, ce modèle de tâche sera considéré comme étant le modèle de tâche initial de notre conception. Les modifications de spécification ayant amené à modifier le dialogue en conception (les modifications décrites dans la section 5.1.2) sont exprimées sous forme de modèles de tâches.

Nous allons dans cette partie, illustrer les apports d'une structure hiérarchique pour établir le lien entre les tâches et le dialogue. Nous présentons sur la Figure 6.1.2, la définition des modules de ces modèles initiaux (le modèle de tâches (que nous représentons de nouveau sur la Figure 6.1.2a)) et le modèle de dialogue (Figure 6.1.2b)).



A1 : En Communication
B1 : Pas d'email en cours B2 : Edition d'email B3 : Email édité
C1 : Sélection

Figure 6.1.2 : Modèles a)de tâches et b)de dialogue initiaux

Cette description du dialogue permet d'illustrer l'un des premiers apports de l'utilisation d'une structure hiérarchique pour une conception basée sur les modèles de tâches : l'adaptation de la conception du dialogue aux raffinements de la conception puisque chacune des machines à états correspond à un niveau d'abstraction différent de la description de l'activité.

Modification 1 : Reprendre un email déjà envoyé

Cette première modification ajoute une tâche à un niveau d'abstraction donné (*Reprendre Envoye*) (Figure 6.1.3a) et comme nous l'avons indiqué dans le section 5.1.2.2, la modification est transcrite dans le modèle de dialogue par l'ajout d'une transition dans l'automate d'identification d'email.

Cette modification n'affecte que la sous-tâche *Sélectionner email* et l'automate A3 de la Figure 6.1.2b (représenté sur la Figure 6.1.3b). Ces deux composants sont du même niveau d'abstraction, leur association en un même module permet de centraliser les modifications à apporter dans l'un des modèles en fonction des modifications de l'autre. Ceci illustre la facilité qu'apporte la modularisation du dialogue pour intégrer les modifications apportées lors de la conception.

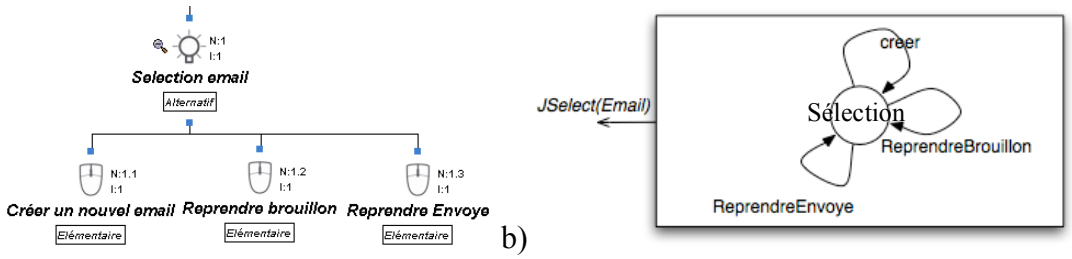


Figure 6.1.3 : Prise en compte de la première modification dans les modèles a) de tâches et b) de dialogue

Modification 2 : Extension des fonctionnalités du *mailer*

Cette modification vise à réutiliser un dialogue déjà conçu pour une autre application. Les machines à états permettant la gestion du dialogue de ces nouvelles fonctionnalités sont alors ajoutées aux machines à états du dialogue du *mailer* tel qu'il est décrit dans 5.1.2.3.Dans le modèle de tâche, cette modification est exprimée par l'intégration du modèle de la Figure 6.1.2a dans un modèle de tâche plus important. La tâche racine du modèle de la Figure 6.1.2a devient alors tâche fille d'une tâche de plus haut niveau d'abstraction (Figure 6.1.4). Les modules réalisés pour le *mailer* sans ces fonctionnalités sont alors réutilisables pour cette seconde version du mailer développé et donc leur validation par rapport au modèle de tâche est toujours valable.

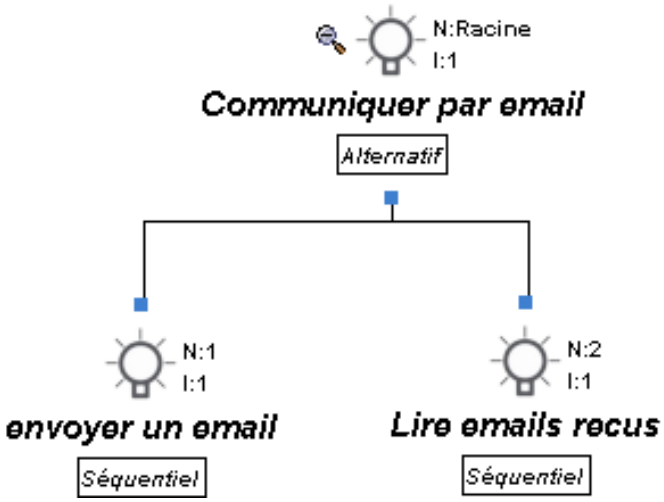


Figure 6.1.4 : Modèle de tâches d'un mailer pour l'envoi et la lecture des emails

Modification 3 : Ajout de l'interaction

Les modèles de tâches décrivent les activités et n'ont pas pour but d'exprimer les caractéristiques d'interaction. L'ajout/modification de l'interaction dans le modèle de dialogue n'est donc pas exprimé dans le modèle de tâches.

Bilan de l'association des niveaux hiérarchiques

L'association par niveaux hiérarchiques permet une indépendance entre les niveaux hiérarchiques. Par exemple, le niveau d'abstraction le plus bas (propre à l'interaction utilisée) peut être ajoutée/modifié, sans que les niveaux supérieurs (qui peuvent être validés par rapport au modèle de tâche) n'aient à être modifié. De même, les modifications apportées dans le modèle de tâches, (comme l'ajout d'une tâche dans le modèle), n'influent que sur l'interacteur du niveau qui lui est associé.

De plus, comme les niveaux d'abstraction du dialogue sont associés aux niveaux d'abstraction des modèles de tâches, à partir des modifications apportées aux modèles de tâches, nous pouvons induire quel niveau d'abstraction du dialogue reflète cette modification. Ceci permet de minimiser le nombre de modifications à apporter aux modèles de dialogue.

Nous avons identifié trois types d'intégrations de modification dans le modèle de dialogue :

- l'addition d'un niveau d'abstraction supplémentaire
- le changement de la machine à états complet du module dans lequel appartient la tâche modifiée
- l'apport de complément dans un automate (comme l'ajout d'une transition).

L'implication des modifications dans le modèle de tâches vers le modèle de dialogue exploite les autres liens que nous présentons ci-dessous (sections 6.1.2 à 6.1.4). Ces modifications ont pour but :

- d'apporter un complément d'information sur l'activité (en spécifiant les tâches)
- de modifier l'activité exprimée (comme pour inclure un modèle de tâches dans une activité plus large)

6.1.2. Première association : tâches/transitions

Ce lien est le plus évident et celui le plus couramment exploité dans les travaux de génération du dialogue à partir du modèle de tâche. Ces travaux, comme TERESA et Dygimes framework (voir 2.3.2.1 et 2.3.2.2), associent à chaque tâche (non réalisée entièrement par un utilisateur) une transition dans le dialogue de l'application, définissant ainsi un lien bijectif entre les deux entités.

Cependant, comme nous l'avons montré dans 3.3, ce lien entre les transitions et les tâches n'est pas un lien bijectif, par exemple, le dialogue doit prendre en compte les transitions de retour en arrière alors que le modèle de tâche ne les détaille pas nécessairement. Le *modèle de mapping* qui permet de passer d'une tâche T à une transition D doit donc être défini par le concepteur. Cependant, de nombreuses règles de cohérence peuvent être établies, comme nous le verrons dans la section 6.3.

Dans la suite, nous nommerons t la relation liant une tâche du modèle de tâches à une transition du modèle de dialogue telle que $t(T_1)=D_1$ avec T_1 une tâche du modèle de tâches et D_1 une transition du modèle de dialogue.

Dans le méta-modèle détaillant les liens entre ces deux modèles, cette association s'exprime à travers les jetons du modèle de dialogue.

6.1.3. Deuxième association : tâche décomposée/automate

Comme le montre l'ajout de l'interaction dans le processus de conception du *mailer* (modification n°3), le modèle de dialogue descend plus bas dans le niveau d'abstraction que le modèle de tâche. Toutes les tâches décomposées dans l'arbre de tâches ont donc une représentation dans le modèle de dialogue.

La décomposition d'une tâche T_a dans un modèle de tâches permet de préciser la manière dont T_a est réalisée. Elle indique donc que l'accomplissement de plusieurs tâches est nécessaire à sa réalisation. Dans le dialogue, la réalisation d'une succession de tâches est représentée par un automate.

Donc à chaque tâche décomposée du modèle de tâche, correspond au moins un automate dans le niveau d'abstraction du modèle d'IH inférieur.

Notons d, la relation entre les tâches décomposées du modèle de tâche et les automates du modèle de dialogue, avec :

$d(T_a)=\{A\} \setminus \forall A_i\ n_{Ai}<n_{T1}$

avec

T_a est une tâche
$\{A\}$ un ensemble d'automates
n_{Ai} le niveau hiérarchique de l'automate A_i
n_{Ta} le niveau hiérarchique de la tâche T1

6.1.4. Troisième association : objets/variables

Le dernier lien que nous avons identifié entre les entités *tâches* et *transitions* des modèles est la représentation de l'état des modèles. Les tâches et les transitions ont tous deux un état de départ et un état d'arrivée. Dans le modèle de dialogue, ces états sont explicitement décrits alors que dans un modèle K-MAD, l'état des objets est déductible des valeurs des objets concrets au moment de l'exécution de la tâche.

Dans les automates représentant le dialogue, les états représentés décrivent l'état de l'ensemble des variables. Certaines des variables du système sont les représentations des objets du monde de l'utilisateur, par exemple l'objet abstrait *email* du modèle de tâches peut être représenté informatiquement par une instance d'une classe *Email* (Figure 6.1.5).

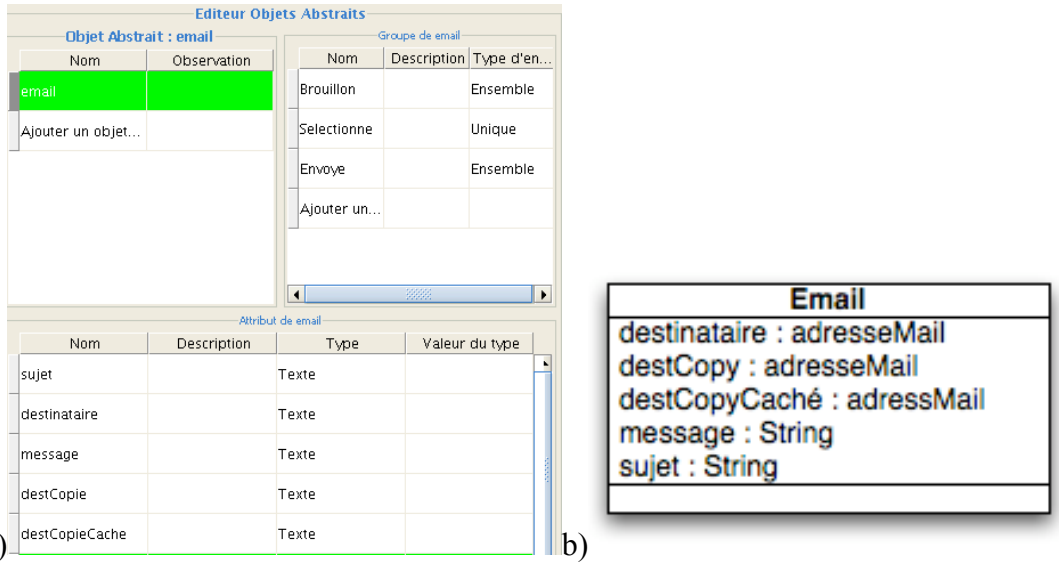


Figure 6.1.5 : Représentation d'un objet Email a) dans K-MAD et b) dans l'implémentation d'un mailer

Soit r, la relation qui associe a un objet abstrait du modèle de tâche, une variable du système telle que $r(OA_1)=V_1$ avec OA_1 un objet abstrait et V_1 une variable du système. La

relation r n'est pas bijective et ne peut donc être automatiquement définie. De plus, il est à noter que cette relation r s'étend aux attributs de l'objet OA_1.

6.1.5. Quatrième association : les expressions

La dernière association possible entre modèles découle de l'existence d'expressions dans les deux modèles. Les pré-conditions des tâches peuvent ainsi être associées aux gardes des transitions exprimées dans le modèle de dialogue. De même, les actions du modèle de tâches peuvent être mises en correspondance avec les actions utilisées dans les automates. Dans la suite de ce travail nous ne détaillerons pas cette quatrième association, qui nécessite des études spécifiques.

6.1.6. Bilan

Nous avons identifié dans cette section, quatre liens exprimables entre les modèles de tâches et de dialogue, prenant appui sur la structure hiérarchique de ces deux modèles :

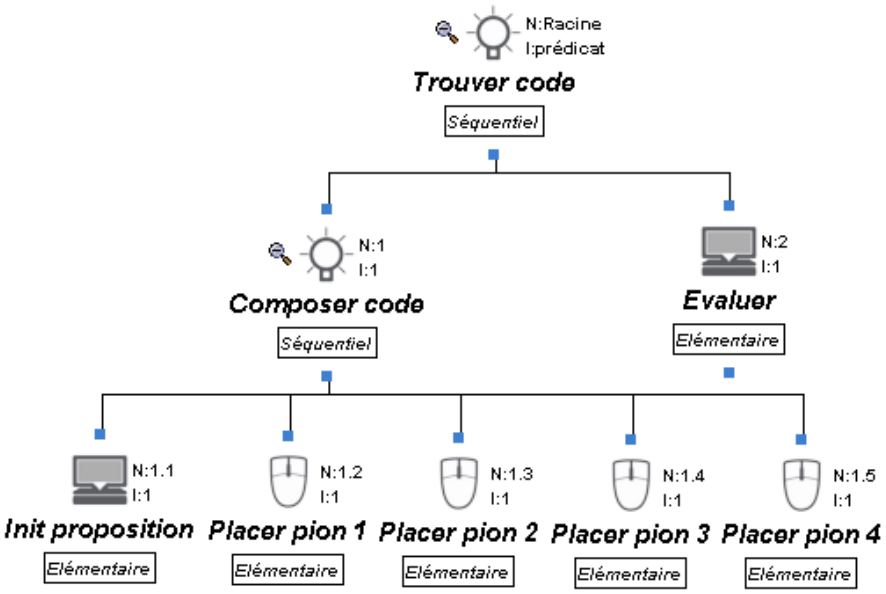
- un lien entre les transitions et les tâches
- un lien entre les tâches décomposées et les automates composant les interacteurs du modèle de dialogue
- un lien entre les objets exprimés dans le modèle de tâches et les variables utilisées dans le modèle de dialogue
- enfin un lien entre les expressions exprimées dans chacun des modèles

L'expression formelle de ces liens est donnée en annexe (Chapitre 9). De ces liens découlent des règles qui permettent d'exprimer le niveau de cohérence entre les modèles. Ces règles sont exprimées dans la section 6.3. Avant cela, nous considérons la notion d'équivalence entre modèles de tâches (section 6.2).

6.2. Notion d'équivalence de modèles K-MAD(v2)

Notre approche consiste à associer les niveaux hiérarchiques du modèle de tâches avec les niveaux hiérarchiques du modèle de dialogue. Cependant, une même activité peut être représentée par des modèles de tâches distincts, tous équivalents, mais organisés de manière différente. Par exemple, sur la Figure 6.2.1, les deux modèles de tâches représentent la composition d'une proposition au Mastermind mais le modèle a) est composé de trois niveaux alors que le modèle b) est composé de deux niveaux hiérarchiques.

a)



b)

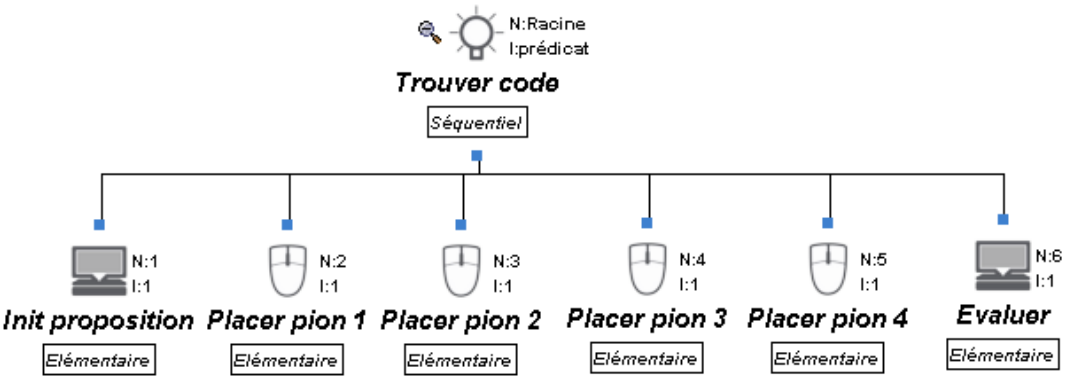


Figure 6.2.1 : Deux représentations d'une même activité

Afin de comparer efficacement les modèles de tâches et de dialogue, il semble nécessaire de normaliser les modèles de tâches (une autre approche est de concevoir les modèles tels qu'ils ont été conçus) et de définir des transformations plus complexes lors de l'étape d'association des modèles. Dans TERESA (section 2.3.2.1), on trouve également cette préoccupation, dans le but de limiter les écrans générés à partir du modèle de tâches. Cependant, bien que la réécriture des modèles permet de simplifier le traitement dans le reste de l'approche, il est à noter que cette approche ne doit pas ajouter de contraintes supplémentaires pour le concepteur, c'est pourquoi nous proposons des transformations à appliquer une fois le modèle de tâches conçu. Nous avons identifié deux possibilités de transformation des modèles K-MAD(v2) qui permettront de simplifier les modèles de tâches construits : la limitation des niveaux hiérarchiques intermédiaires (comme la tâche de la Figure 6.2.1) et la limitation des tâches identiques dans l'arbre de tâches.

Comme pour cette étude la sémantique des niveaux hiérarchiques des modèles de tâches a une grande importance, la première transformation de la phase de préparation du modèle de tâches vise à limiter les niveaux hiérarchiques aux niveaux portant une

sémantique. Pour cela, nous avons étudié les raisons qui ont pour conséquence l'ajout de niveaux « intermédiaires » qui peuvent être transformés (voir la section 6.2.1).

La comparaison des niveaux hiérarchiques des modèles de tâches et de dialogue repose sur le constat que les tâches des modèles de tâches et les transitions des modèles de dialogue sous forme de machine à états sont très proches. Lors de la description d'activité, des tâches réalisées de manière très proche peuvent être décrites très loin l'une de l'autre dans l'arbre de tâche. La seconde transformation proposée a pour but la réécriture de l'arbre K-MAD(v2) pour limiter la présence de telles tâches (voir la section 6.2.2).

Ces deux transformations sont réalisées itérativement jusqu'à ce que plus aucune transformation ne puisse être réalisée (Figure 6.2.2).

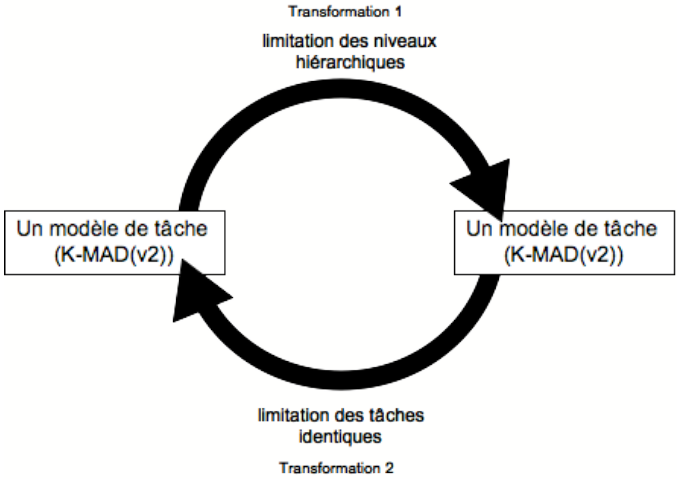


Figure 6.2.2 : Les étapes de transformations de préparation

6.2.1. Limitations des niveaux hiérarchiques

Afin de faciliter la compréhension des modèles de tâches, des niveaux hiérarchiques peuvent être ajoutés par le concepteur des modèles de tâches. Par exemple sur la Figure 6.2.3, l'activité représentée est la même alors que les modèles de tâches différent d'une tâche : *Editer avec K-MADe*, qui n'ajoute aucune information sur l'ordonnancement des tâches. Les transformations présentées dans cette section visent à limiter les niveaux hiérarchiques des modèles de tâches en supprimant ces tâches de « synthèse » qui n'apportent rien pour la vérification du dialogue de l'application.

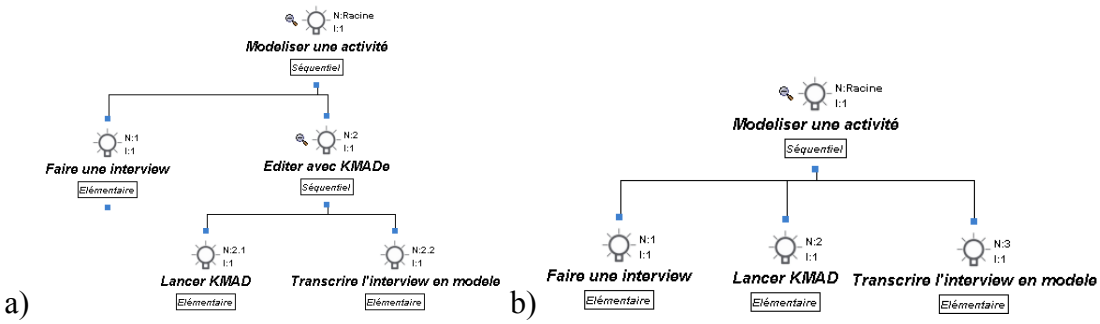


Figure 6.2.3 : Exemple de modélisations identiques avec un niveau de différence

Ces suppressions de niveaux sont réalisées lorsque le même opérateur de décomposition est utilisé sur deux niveaux qui se suivent, comme l'opérateur *séquentiel* sur le modèle illustré sur la Figure 6.2.3a. Nous allons décrire, pour chaque opérateur de décomposition, les modifications apportées par la suppression de niveaux hiérarchiques et les cas où cette suppression est impossible. Les règles présentées dans la suite (RNH1 à RNH4) ont pour but de produire des modélisations présentées sous la forme de la Figure 6.2.3b à partir de modélisation telle qu'elle est présentée sur la Figure 6.2.3a.

On notera que la suppression d'un niveau hiérarchique (par suppression d'une tâche intermédiaire) ou la réorganisation des tâches n'est pas seulement un problème d'opérateurs. Il faut également examiner le cas des différents attributs de la tâche supprimée, afin de déterminer dans quelle mesure ces derniers doivent être pris en compte dans les autres tâches.

Cette suppression de niveau hiérarchique ne peut pas être réalisée chaque fois que deux opérateurs identiques se suivent. En effet, lorsque la tâche intermédiaire est itérative (précisant ainsi que la séquence de ses sous-tâches est répétée), la tâche intermédiaire n'a pas uniquement un rôle d'aide à la compréhension du modèle mais également un rôle sémantique. Une condition à la suppression de tâches intermédiaires est donc que ces tâches ne soient pas itératives.

Il en est de même sur le caractère optionnel de la tâche. Lorsque la tâche est optionnelle, elle ne peut pas être supprimée. En effet, le caractère *optionnel* de la tâche intermédiaire implique que ses sous-tâches sont toutes exécutées ou toutes « passées » pour un même scénario. Cette sémantique n'est exprimable d'aucune autre manière à partir des seules sous-tâches. Nous spécifierons pour tous les autres cas le comportement des autres caractéristiques des tâches.

Dans la suite, la tâche T1 est la tâche sujette à être supprimée (remplissant toutes les caractéristiques nécessaires – non itérative et non optionnelle).

∃ T1 \ T1.opt=false & T1.iter=1

RNH1 : Cas de succession de l'opérateur *séquentiel*

Ce cas de figure est celui illustré sur la Figure 6.2.3. Lorsque l'opérateur de décomposition qui compose successivement deux niveaux hiérarchiques est l'opérateur de

décomposition *séquentiel*, la modification consiste en la suppression du niveau intermédiaire (représenté par la tâche « Editer avec K-MADe » sur la Figure 6.2.3a).

Certaines des caractéristiques de la tâche intermédiaire doivent alors être introduites sur ses sous tâches. C'est le cas, de la pré-condition et de l'événement déclenchant qui doivent être ajoutés à la pré-condition et comme événement déclenchant de la première sous-tâche. De même, ses événements déclenchés, sa post-condition et son action complètent les événements déclenchés, la post-condition et l'action de la dernière tâche fille. Enfin, si la tâche supprimée est non interruptible alors ses sous-tâches doivent également l'être. Les autres caractéristiques des sous-tâches ne sont pas modifiées pour y inclure les caractéristiques de la tâche intermédiaire.

```
(T1.op==séquentiel) => (T1.filles(1)).pre += T1.pre
                       (T1.filles(1)).evtDecl += T1.evtDecl
                       (T1.filles(n)).post += T1.post
                       (T1.filles(n)).action += T1.action
                       (T1.filles(n)).evtTrig += T1.evtTrig
                       => arbre.delete(T1)
```

RNH2 : Cas de succession de l'opérateur *alternatif*

La règle à appliquer dans le cas de deux décompositions successives *alternatif* (comme l'exemple illustré sur la Figure 6.2.4.) est la même que celle appliquée lorsque les décompositions sont *séquentielles*, c'est-à-dire la suppression de la tâche intermédiaire.

Cependant, dans ce cas, la pré-condition, l'événement déclenchant, l'action, la post-condition et les événements déclenchés sont ajoutés à l'ensemble des sous-tâches de la tâche intermédiaire. Si la tâche intermédiaire est non interruptible alors les sous-tâches sont également non interruptibles. Cette caractéristique doit alors être appliquée à l'ensemble des sous-tâches de la tâche intermédiaire. Ce sont les seules caractéristiques des tâches qui sont affectées par la suppression de la tâche intermédiaire.

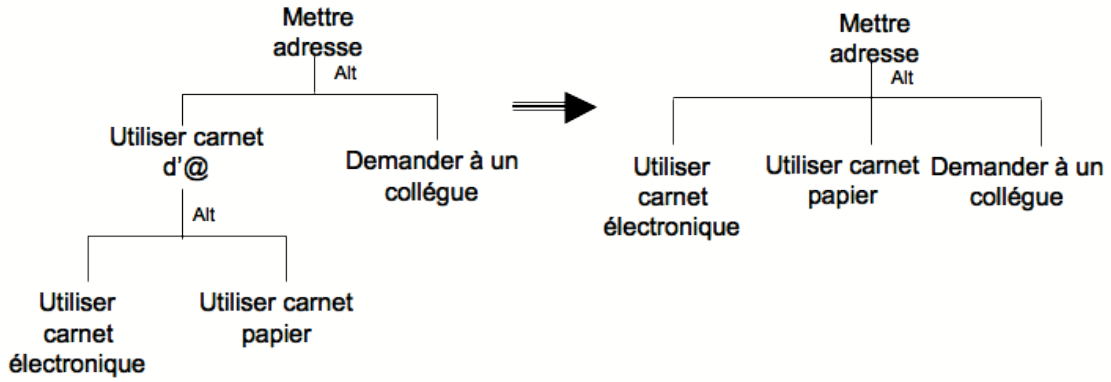


Figure 6.2.4 : Exemple de transformation d'opérateurs alternatifs

(T1.op==alternatif) =>	(T1.filles).pre += T1.pre (T1.filles).evtDecl += T1.evtDecl (T1.filles).post += T1.post (T1.filles).action += T1.action (T1.filles).evtTrig += T1.evtTrig si (T1.nonInter==true) => T1.fille.nonInter==true => arbre.delete(T1)

RNH3 : Cas de succession de l'opérateur *parallèle*

Dans le cas de la succession d'opérateurs *parallèle*, la tâche intermédiaire peut être supprimée. Cependant, les conditions pour que cette modification puisse être appliquée sont plus strictes que dans le cas des opérateurs de décomposition *séquentiel* et *alternatif*. En effet, si la tâche intermédiaire possède une pré-condition, une action, une post-condition, un événement déclenchant ou des événements déclenchés, la suppression de cette tâche intermédiaire ne peut pas être réalisée. L'ordre de l'exécution des sous-tâches n'étant pas déterminé à l'avance et l'exécution des tâches pouvant modifier l'état du monde, il n'est pas possible de *transmettre* les caractéristiques de la tâche intermédiaire aux sous-tâches.

(T1.op==parallèle)	& T1.pre==true & T1.action==void & T1.post==true & T1.evtDecl==∅ & T1.evtTri==∅ => arbre.delete(T1)

RNH4 : Cas de succession de l'opérateur *sans ordre*

La succession d'opérateurs de décomposition *sans ordre* ne permet d'appliquer aucune règle supprimant un niveau hiérarchique. En effet, au contraire de la composition *parallèle*, l'utilisation d'opérateurs *sans ordre* implique que les sous-tâches de la tâche intermédiaire soient toutes impérativement exécutées avant (ou après) que les tâches du niveau supérieur le soient.

6.2.2. Limitation des tâches équivalentes

En observant la conception des modèles de tâches ainsi que les modèles conçus, nous avons constaté que la même tâche pouvait être décrite plusieurs fois dans le même arbre de tâche. Cette répétition de tâche peut compromettre l'établissement du lien entre les modèles de tâches et les modèles de dialogue. Dans certains cas, la transformation de l'arbre de tâche peut permettre la limitation des répétitions des tâches. Dans cette section, nous présentons un procédé pour limiter ces tâches équivalentes dans un arbre de tâche. Cette difficulté fût également détectée dans TERESA (section 2.3.2.1) qui propose plusieurs *regroupement* des tâches pour limiter le nombre d'états qui en découlent.

Nous procédons en deux étapes. Tout d'abord, une détection des tâches équivalentes est réalisée. Puis, nous proposons plusieurs règles de transformations permettant de supprimer cette répétition.

6.2.2.1. Détection des tâches équivalentes

Tout d'abord, pour réaliser le regroupement de tâches équivalentes, nous avons défini ce que nous considérons comme étant deux tâches équivalentes. K-MAD ne propose pas de définition des tâches équivalentes. De plus, l'identificateur des tâches dans un modèle K-MAD (dans les versions 1 et 2) est constitué du numéro qui est calculé en fonction de la place occupée par la tâche dans l'arbre, il ne peut donc pas servir à identifier deux tâches équivalentes. Pour ces deux raisons, nous proposons une définition propre à cette étude. C'est cette définition que nous allons présenter dans un premier temps.

Définition de deux tâches équivalentes

Les conditions définies comme nécessaires à l'identification de deux tâches équivalentes sont issues de l'identification des caractéristiques qui définissent l'unicité des tâches. Les caractéristiques des tâches dans K-MAD(v2) sont rappelées dans le Tableau 6-1.

Comme nous l'avons indiqué précédemment, le **numéro** est calculé automatiquement en fonction de la place occupée par la tâche, ce numéro est donc unique à chaque instance des tâches, nous n'avons donc pas retenu cette caractéristique pour définir les conditions d'identification des tâches équivalentes.

Dans K-MAD(v2), l'**observation**, le **but**, le **média** et la **durée** sont des caractéristiques informatives sur la tâche, elles sont renseignées textuellement par le concepteur et ne peuvent donc pas être utilisées pour une identification formelle.

Le **nom** est lui aussi textuellement défini par le concepteur, cependant comme il est souvent utilisé par celui-ci pour identifier de manière informelle la tâche, nous avons choisi de le retenir comme un des critères d'identification.

Nous considérons le caractère **optionnel** d'une tâche ainsi que la condition d'**itération** comme n'étant pas une condition pour définir deux tâches équivalentes mais comme étant lié à l'environnement d'exécution de la tâche. En effet, deux tâches peuvent être équivalentes même si dans une décomposition elle est optionnelle et dans l'autre elle est obligatoire.

Toutes les autres caractéristiques doivent être équivalentes pour considérer que deux tâches sont équivalentes. Dans cette étude, nous considérons donc deux tâches comme étant équivalentes lorsque ces deux tâches ont le même **nom**, le même **exécutant** (et **modalité** lorsque l'exécutant fait intervenir un être humain), la même **fréquence (à voir)**, la même **importance**, la même décomposition (**opérateur de décomposition** et **ensemble des sous-tâches**), les mêmes conditions d'exécution (**pré-condition** et

événements déclencheurs), les mêmes conséquences sur l'état du monde (**post-condition**, **action**, **événements déclenchés**), les mêmes intervenants (**objets**, **acteurs** et **machines**).

Nom de la caractéristique	Type	
numéro	numéro calculé	-
nom	texte	√
observation	texte	-
but	texte	-
média	texte	-
durée	texte	-
optionalité	booléen	-
non interruptabilité	booléen	√
acteurs	$\{ACTEUR_1, ACTEUR_2,\ldots, ACTEUR_N\}$	√
machines	$\{MACHINE_1, MACHINE_2,\ldots, MACHINE_N\}$	√
objets manipulés	$\{OBJET_1, OBJET_2,\ldots,OBJET_N\}$	√
exécutant	utilisateur, système, interactif, abstrait	√
fréquence	très fréquent, fréquent, peu fréquent	√
importance	très importante, moyennement importante, peu importante	√
modalité	cognitif, sensori-motrice	√
décomposition	choix, parallèle, séquence, sans ordre, élémentaire	√
pré-condition	EXPRESSION	√
post-condition	EXPRESSION	√
action	EXPRESSION	√
condition d'itération	EXPRESSION	-
événement déclenchant	EVENEMENT	√
événement déclenché	EVENEMENT	√
sous-tâches	$\{TACHE_1, TACHE_2,\ldots TACHE_N\}$	√

Tableau 6-1 : Les caractéristiques des tâches dans K-MAD(v2)

Tâches potentiellement équivalentes

Le seul identifiant formel des tâches dans K-MAD(v2) est le numéro de celle-ci. La définition que nous avons proposée dans le paragraphe précédent ne peut en aucun cas être considérée comme formelle et donc comme étant absolue. C'est pourquoi plutôt que de considérer les conditions suscitées comme étant indispensables à la détection de tâches équivalentes, nous proposons de les utiliser pour définir des niveaux de similarité pouvant mener à l'identité. Ainsi deux tâches ayant le maximum des niveaux de correspondance (toutes les caractéristiques identifiées précédemment sont les mêmes) seront automatiquement considérées comme équivalentes. Pour les autres niveaux, le concepteur sera prévenu du niveau de correspondance et pourra, s'il le souhaite, modifier les autres caractéristiques pour que les deux tâches deviennent totalement équivalentes.

6.2.2.2. Limitation des tâches équivalentes

Une fois deux tâches identifiées comme équivalentes, nous avons défini des règles de réécriture permettant de regrouper les tâches pour éliminer les répétitions de tâches dans l'arbre. Dans la suite, les tâches nommées TI sont les tâches ayant été identifiées comme équivalentes selon le processus décrit précédemment.

Comme nous l'avons dit précédemment, deux tâches sont équivalentes lorsque ces tâches sont décomposées de la même manière, c'est-à-dire ayant le même opérateur de décomposition et les mêmes tâches de composition. De ce fait, nous avons procédé à l'établissement des règles de réécriture en commençant par les tâches élémentaires liées par un seul opérateur de décomposition (les tâches sœurs) puis les tâches plus éloignées pour lesquelles la composition des décompositions est à prendre en compte (nommées tâches parentes). Pour chaque cas identifié, nous présenterons la transformation que nous proposons et les modifications que cela implique sur les caractéristiques des tâches (RTI1 à RTI5). Dans la section 6.2.2.1, nous avons identifié les caractéristiques devant être équivalentes pour que deux tâches soient considérées comme équivalentes. Les modifications que nous proposons ne concernent donc pas ces caractéristiques mais les autres, c'est-à-dire : le numéro, le but, l'observation, le média, la durée, l'optionalité et l'itération.

Le **numéro** des tâches n'est pas pris en compte par les modifications que nous proposons du fait que sa valeur est automatiquement calculée.

Le **but** de deux tâches équivalentes est le même, cependant les termes utilisés par le concepteur pour le décrire peuvent être différents. De ce fait, lorsque deux tâches équivalentes ayant des buts exprimés différemment sont regroupés en une seule, il nous faut prévoir l'attribution d'une valeur pour le but de cette tâche. Nous aurions pu choisir d'attribuer l'une des valeurs des deux tâches à la tâche résultante ; cependant, nous avons préféré indiquer conjointement les deux buts en laissant au concepteur le choix de repréciser le but par la suite. Nous avons pris le même parti pour les caractéristiques de **l'observation**, le **média** et la **durée**.

L'**optionalité** et l'**itération** sont deux caractéristiques pour lesquelles les valeurs sont redéfinies pour chaque transformation.

RTI1 : Cas des tâches sœurs en séquence

Dans le cas où une tâche T est décomposée par l'opérateur de décomposition *séquence* avec au moins deux tâches équivalentes élémentaires qui se suivent, une transformation de l'arbre peut être faite. Les deux tâches TI sont alors « fondues » en une seule qui est répétée un nombre de fois correspondant à la somme des itérations des deux tâches TI.

Un exemple est illustré sur la Figure 6.2.5 avec les tâches TI initialement exécutées une seule fois chacune. Dans K-MAD(v2), le nombre d'itération n'est pas défini comme étant une fourchette mais un nombre fixe ou une expression. Il est donc impossible d'exprimer dans l'itération d'une tâche qu'elle est exécutée N_{TI2} ou (N_{TI2} + N_{TI1}) fois.

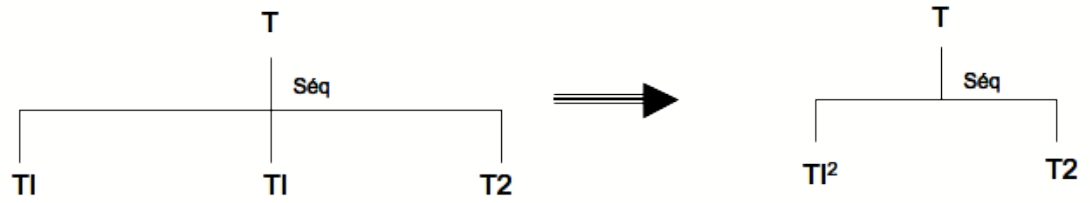


Figure 6.2.5 : Transformation d'un arbre de tâches avec deux tâches équivalentes en séquence

Lorsque l'une des deux tâches équivalente est optionnelle, l'itération de l'exécution de TI est soumise à condition. En effet, si TI_1 est la tâche optionnelle et TI_2 ne l'est pas, la tâche TI est alors itérée ($N_{TI2} + N_{TI1}$) fois dans le cas où TI_1 est exécutée ou (N_{TI2}) fois dans le cas où TI_1 n'est pas exécutée. C'est pourquoi, nous proposons de définir la tâche TI comme étant décomposée alternativement en deux tâches TI1 ayant comme condition d'itération ($N_{TI2} + N_{TI1}$) et une tâche TI2 ayant une condition d'itération (N_{TI2}) (Figure 6.2.6).

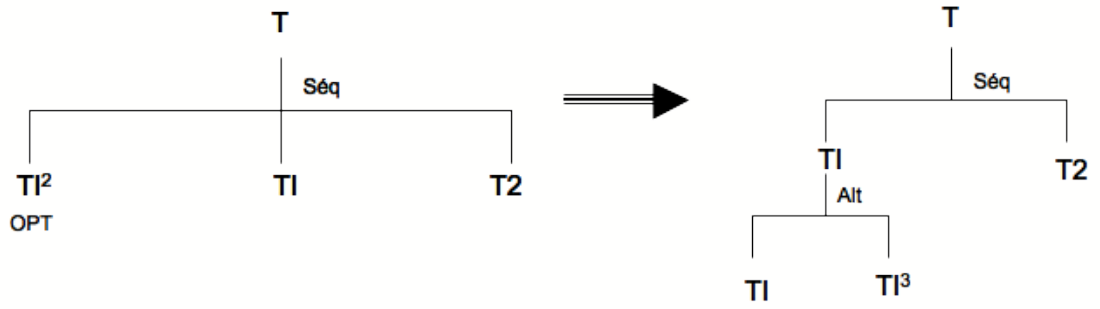


Figure 6.2.6 : Les tâches itératives équivalentes en séquence avec une tâche optionnelle

Lorsque les deux tâches équivalentes sont toutes deux sous-tâches d'une même tâche séquentiellement composée mais pas l'une à la suite de l'autre (une (ou plusieurs) autre(s) tâche(s) les sépare(nt)) alors l'arbre de tâches ne peut pas être modifié pour supprimer l'occurrence de TI.

RTI2 : Cas des tâches sœurs alternatives

La décomposition d'une tâche alternative en deux tâches équivalentes n'a de sens que si les conditions d'itération ne sont pas les mêmes. Dans ce cas, aucune modification ne peut être apportée au modèle proposé tant que des modifications sur l'expression des conditions n'ont pas été apportées.

Si les tâches équivalentes ont la même condition d'itération alors l'arbre peut être modifié (bien que la sémantique d'une telle décomposition soit contestable). La modification consiste alors en la suppression d'une des deux tâches T1. Cette transformation n'est pas conditionnée par la place qu'occupent les tâches TI dans la

décomposition. Un exemple de cette transformation est illustré sur la Figure 6.2.7. Dans le cas où l'itération des tâches T1 est égale à 1.

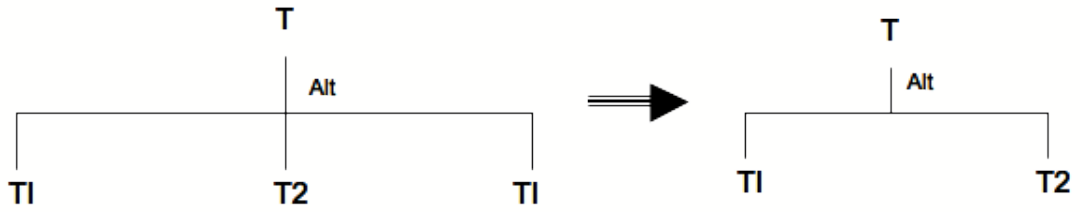


Figure 6.2.7 : Transformation d'un arbre de tâches avec deux tâches équivalentes en alternatif.

RTI3 : Cas de tâches sœurs parallèles et sans ordre

Quelles que soient les valeurs d'optionalité et d'itération des tâches TI, il n'y a pas de sens à les considérer comme pouvant être deux sous-tâches d'une tâche décomposée en tâches parallèles dans K-MAD(v2).

De plus, lorsque les tâches TI sont élémentaires, la liaison par l'opérateur *parallèle* est équivalente à celle par l'opérateur *sans ordre* puisque K-MAD(v2) ne considère pas le parallélisme des tâches élémentaires.

RTI4 : Cas de tâches parentes liées par une composition *alternatif* et *séquentiel*

Une fois identifiés les cas où les tâches équivalentes sont toutes deux des tâches élémentaires, nous nous intéressons à présent aux cas de tâches équivalentes liées par au moins deux opérateurs de décomposition (combinaison de composition). Les tâches équivalentes sont de ce fait filles de tâches différentes. Rappelons que si les tâches équivalentes sont les tâches décomposées (c'est-à-dire que toutes les sous-tâches des tâches équivalentes sont également équivalentes) on applique alors les mêmes règles que pour les tâches sœurs élémentaires équivalentes sur les tâches composées (T1 et T2 sur la Figure 6.2.8a). Nous ne traiterons pas les cas où les opérateurs de décomposition des deux niveaux hiérarchiques se suivant sont les mêmes car ces cas de Figure sont étudiés dans la section 6.2.1.

Lorsque les tâches équivalentes sont des tâches « cousines » nous avons détecté un cas de composition d'opérateurs de décomposition qui permet la suppression de l'itération des tâches équivalentes.

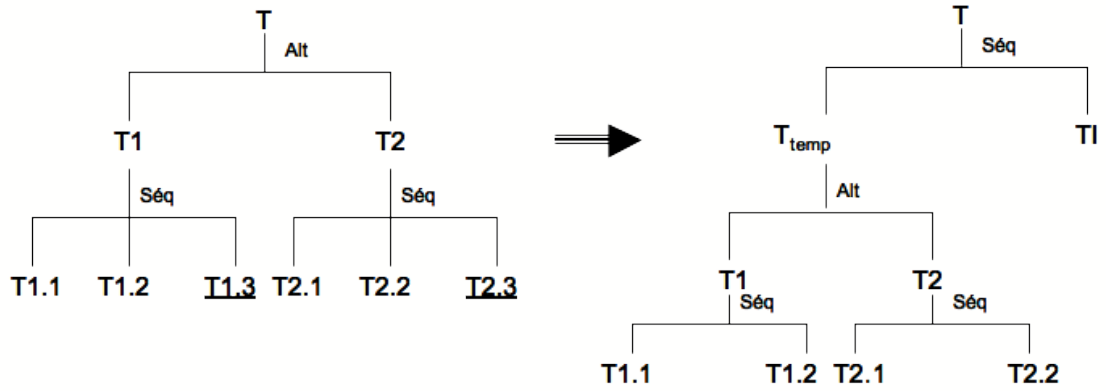
La structure de cette composition est celle présentée sur la Figure 6.2.8a. L'opérateur de décomposition de la tâche du niveau supérieur (T) est l'opérateur *alternatif* et ses sous-tâches T1 et T2 (où T1 et T2 sont les tâches mère des tâches équivalentes) sont décomposées *séquentiellement*. Cependant, la suppression des tâches équivalentes n'est pas réalisable pour tous les cas de Figure. Il est nécessaire que les tâches équivalentes soient toutes aux extrémités des tâches séquentielles. Par exemple pour le cas représenté

sur la Figure 6.2.8a, il est possible de supprimer la répétition des tâches équivalentes si ces tâches sont T1.1 et T2.1 mais pas si ce sont les tâches T1.2 et T2.2.

La transformation à appliquer consiste alors en l'échange de niveaux de composition des opérateurs de décomposition et en un regroupement des tâches équivalentes. Les tâches non équivalentes sont groupées sous une tâche décomposée avec l'opérateur *alternatif* et cette tâche est appliquée comme sous-tâche de la tâche du niveau supérieur. Cette tâche a alors pour opérateur de décomposition, l'opérateur *séquentiel*. La tâche équivalente est alors ajoutée à la composition de la tâche de niveau supérieur.

Par exemple, dans le cas présenté sur la Figure 6.2.8a, si les tâches équivalentes sont les tâches T1.3 et T2.3 alors l'arbre de tâches modifié est celui présenté sur la Figure 6.2.8b.

Aucune tâche n'est supprimée dans cette transformation, il n'est donc pas nécessaire de prévoir des modifications des caractéristiques pour y inclure les caractéristiques de la tâche supprimée comme dans les cas présentés précédemment.



a) b)

Figure 6.2.8 : Structure d'une composition avec tâches équivalentes a)original b)modifié

RTI5 : Tâche décomposée en une seule sous-tâche

À la suite des transformations que nous proposons, il peut être possible que l'arbre de tâches ne remplisse plus la grammaire de K-MAD(v2). Nous avons par exemple détecté qu'une tâche T peut être décomposée en une seule sous-tâche T1 (ce qui n'est pas permis dans le modèle de K-MAD). La tâche T est alors remplacée par la tâche T1 à laquelle sont ajoutées les conditions d'exécution, et les conséquences sur l'état du monde de T. La tâche T1 est optionnelle si T ou T1 l'est et non optionnelle si aucune des deux tâches initiales n'est optionnelle.

6.3. Les règles de cohérence

Nous allons, dans cette section, présenter les règles formelles de cohérence pouvant être vérifiées dans les modèles. Pour cela, les liens que nous avons présentés dans la section 6.1, basés sur la structure hiérarchique des deux modèles, sont exploités. Les règles

sont des conditions minimales devant être vérifiées pour considérer les deux modèles comme étant cohérents l'un par rapport à l'autre. Nous allons les présenter en suivant l'ordre des liens que nous avons définis dans la section 6.1.

6.3.1. Les niveaux hiérarchiques

L'association pour laquelle nous proposons d'établir les premières règles de cohérence est celle que nous avons définie dans 6.1.1 : l'association par niveau d'abstraction.

RC 1

Cette association fait correspondre à chaque niveau hiérarchique du modèle de tâche un niveau hiérarchique du modèle de dialogue. Cependant, comme nous souhaitons laisser le concepteur le plus libre possible dans son processus de conception, nous ne pouvons pas garantir que ce lien soit bijectif. Par exemple, sur la Figure 6.3.1a les tâches *Editer sujet, Editer destinataire* et *Editer message* sont à un niveau différent de la tâche *Envoyer* alors que dans le dialogue (sur la Figure 6.3.1b) les transitions *Editer message*, *Editer dest*, *Editer sujet* et *Envoyer* sont sur le même automate (donc même niveau d'abstraction dans le modèle de dialogue).

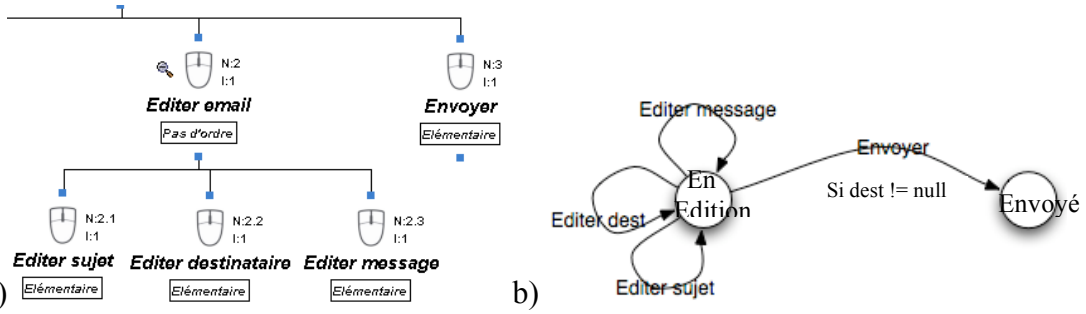


Figure 6.3.1 : Exemple de lien a) tâches b) dialogue non bijectif

Même si l'association des tâches et des machines à états n'est pas bijective, il est tout de même possible d'en établir une règle entre les deux niveaux hiérarchiques des modèles. Les niveaux hiérarchiques du modèle de dialogue doivent respecter l'ordre des niveaux hiérarchiques établi dans le modèle de tâche.

$n_{T1} = n_{T2}$, $t(T_1) = D_1$ et $t(T_2) = D_2 \Leftrightarrow n_{D1} = n_{D2}$
avec T_1, T_2 deux tâches du modèle de tâche
D_1, D_2 deux transitions du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions
n_{T1}, n_{T2} les niveaux hiérarchiques des tâches T_1 et T_2
n_{D1}, n_{D2} les niveaux hiérarchiques des transitions D_1 et D_2

$n_{T1} < n_{T2}$, $t(T_1) = D_1$ et $t(T_2) = D_2 \Leftrightarrow n_{D2} \geq n_{D1}$

avec T_1, T_2 deux tâches du modèle de tâche

D_1, D_2 deux transitions du modèle de dialogue

t la relation de correspondance entre les tâches et les transitions

n_{T1}, n_{T2} les niveaux hiérarchiques des tâches T_1 et T_2

n_{D1}, n_{D2} les niveaux hiérarchiques des transitions D_1 et D_2

6.3.2. Tâche décomposée/Automate

RC 2

Le lien principal entre une tâche et les automates du modèle de dialogue est le fait que toute tâche décomposée est représentée dans le modèle de dialogue comme étant un automate. L'automate du modèle de dialogue qui représente une tâche décomposée dans le modèle de tâches est à un niveau d'abstraction inférieur.

T1.décomposition != élémentaire => $\exists$ A \ $n_A <= n_{T1} - 1$

De plus, l'opérateur de décomposition qui organise temporellement les sous-tâches joue un rôle sur les séquences de transitions de l'automate correspondant. Chacun des opérateurs de décomposition implique des règles particulières sur les automates correspondants. Soit une tâche T décomposée en deux sous-tâches T1 et T2. Nous allons décrire pour chaque opérateur de décomposition les conséquences engendrées sur l'automate de dialogue.

RC 3

Si l'opérateur de décomposition est *séquentiel,* l'état d'arrivée de la transition correspondant à T1 est le même que l'état de départ de la transition correspondant à T2.

T_0.décomposition = séquentiel, T_1 et $T_2 \in T_0$.filles, $T_2 = next(T_1)$,

$t(T_1) = D_1$, $t(T_2) = D_2 \Leftrightarrow D_1.end = D_2.beg$

avec T_0, T_1, T_2 trois tâches du modèle de tâche

D_1, D_2 deux transitions du modèle de dialogue

t la relation de correspondance entre les tâches et les transitions

RC 4

Si l'opérateur de décomposition est *alternatif,* les transitions correspondant aux tâches T1 et T2 partent du même état.

T_0.décomposition = alternatif, T_1 et $T_2 \in T_0$.filles, $t(T_1)=D_1$, $t(T_2)=D_2$

$\Leftrightarrow D_1.beg=D_2.beg$

avec T_0, T_1, T_2 trois tâches du modèle de tâche

D_1, D_2 deux transitions du modèle de dialogue

t la relation de correspondance entre les tâches et les transitions

RC 5

Si l'opérateur de décomposition est *sans ordre,* quel que soit le chemin emprunté dans l'automate, l'état d'arrivée est le même. Dans le cas de notre exemple avec les deux sous-tâches T1 et T2, l'automate se présente comme sur la Figure 6.3.2.

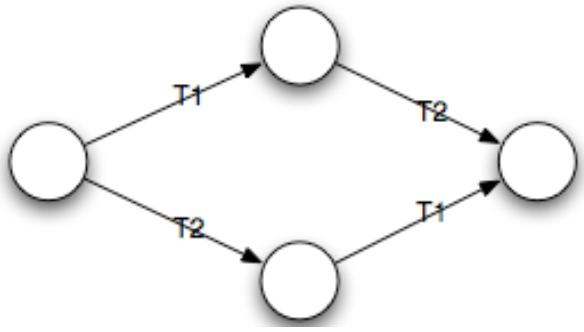


Figure 6.3.2 : Exemple de machine à état correspondant à l'opérateur sans ordre

T_0.décomposition = sans ordre, T_1 et $T_2 \in T_0$.filles, $t(T_1)=D_1$, $t(T_2)=D_2$ ⇔ quel que soit le chemin pris, l'état d'arrivée est le même

avec T_0, T_1, T_2 trois tâches du modèle de tâche
D_1, D_2 deux transitions du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions

RC 6

En plus des opérateurs temporels, les tâches des modèles de tâches possèdent une autre caractéristique qui donne des indications sur la structure des automates : l'itération. Dans le modèle K-MAD(v2), deux types d'itération ont été définis : l'itération contrôlée par le modèle et l'itération contrôlée par l'utilisateur. Lorsque l'itération de T est contrôlée par le modèle, l'état d'arrivée de la transition correspondante est le même que son état de départ, ou un autre état lorsque la condition d'itération n'est pas remplie. Au contraire lorsque l'itération de T est contrôlée par l'utilisateur alors la transition correspondant à T a le même état de départ et d'arrivée et la première transition réalisée lorsque l'itération a pris fin part de cet état pour en changer.

∀ T_1.iterationModelControl & ∀ T_1.iterationUserControl, $t(T_1)=D_1$ ⇔ D_1.end=D_1.beg

avec T_1 une tâche du modèle de tâche
D_1 une transition du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions

6.3.3. Transitions/Tâches

Bien que le lien entre les transitions du modèle de dialogue et les tâches du modèle de tâches ne soit pas un lien bijectif, intuitivement, il semble exister de manière forte. Nous allons, dans cette section, chercher à définir formellement des règles de cohérence entre les transitions et les tâches en exploitant la relation définie dans 6.1.2.

6.3.3.1. Les types de tâches

Les tâches des modèles de tâches K-MAD peuvent être de quatre types différents : Abstraite (dons décomposées), Utilisateur, Système ou Interactif. Dans un dialogue, les transitions sont des associations entre un déclencheur (actions produites par l'utilisateur ou résultant d'une de ces actions) et l'exécution par le système d'une (ou des) action(s). Le point de départ de toute transition est donc une action de l'utilisateur sur le système (Figure 6.3.3).

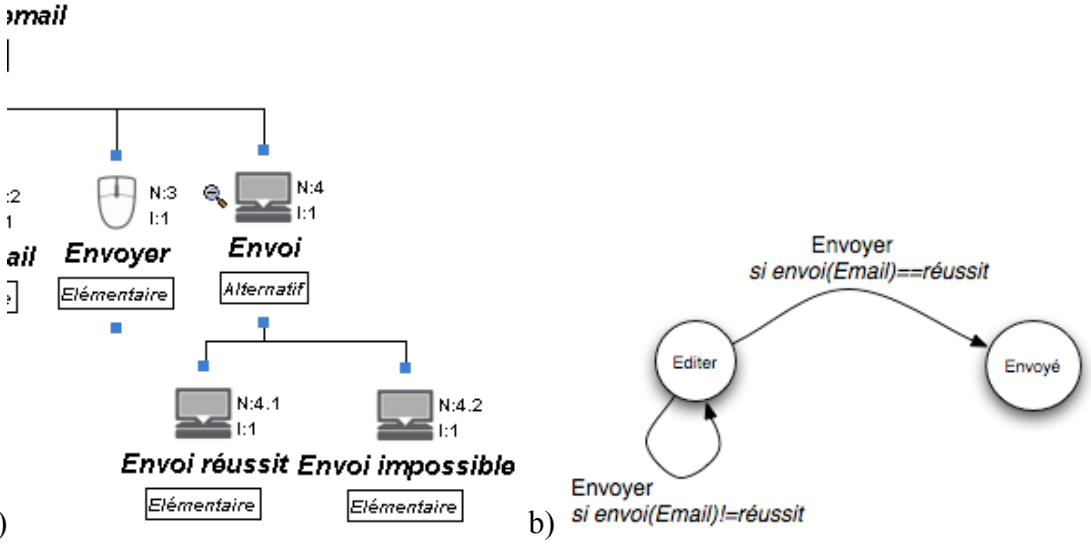


Figure 6.3.3 : Exemple de réaction du système à l'impossibilité d'envoi a) Extrait du modèle de tâches et b) Extrait du modèle de dialogue

RC 7

$t(T_1)=D_1 \Leftrightarrow T_1.type \neq Utilisateur$

avec T_1 une tâche du modèle de tâche

D_1 une transition du modèle de dialogue

t la relation de correspondance entre les tâches et les transitions

RC 8

De ce fait, les tâches réalisées uniquement par un utilisateur ne peuvent être représentées dans le dialogue de même que les tâches exécutées par un système seul ne sont pas représentées par des transitions mais « fondues » dans des transitions correspondant à l'exécution des tâches interactives. Par exemple, lorsqu'un email est envoyé (l'ordre d'envoi est donné par l'utilisateur) le système réalise l'envoi et si celui-ci n'est pas possible (taille trop importante, adresse invalide…), le système renvoi un *feedback* indiquant à l'utilisateur que l'envoi n'a pas été réalisé et éventuellement la raison. Cette tâche de *feedback* peut être ajoutée dans le modèle de tâches (Figure 6.4.3a). Cependant, dans le modèle de dialogue les tâches de *feedback* (*Envoi réussi* et *Envoi impossible*) seront confondus avec la tâche interactive *Envoyer* (Figure 6.4.3b).

$\forall$ T / T.type = Interactif, $\exists$ t(T)
avec T une tâche du modèle de tâche
t la relation de correspondance entre les tâches et les transitions

NB : T.type = Système $\Leftrightarrow \exists$ T_2 / $t(T_2)$=t(T) avec T_2.type = Interactif

6.3.3.2. Machines du modèle de tâches

RC 9

Le dialogue modélisé est celui d'un unique système. De ce fait, toutes les tâches doivent être associées à une même machine (qui est celle dont le dialogue est modélisé).

$\forall$ T_1 / $t(T_1) \in$ D, T.machine = m
avec T_1 une tâche du modèle de tâche
D, l'ensemble des transitions du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions

6.3.4. Objets/Variables

Les objets et les variables sont utilisés dans les modèles pour décrire des conditions d'exécution ou des modifications de l'état du monde représenté. Les associations décrites précédemment permettent d'établir des règles de cohérences sur l'utilisation des objets et variables dans les conditions d'exécution et les actions.

6.3.4.1. Les actions

Transitions et tâches ont des actions associées. Les deux actions traduisent l'exécution qui peut modifier l'état du monde (des variables et des objets concrets). Bien qu'un lien sémantique semble évident, il semble difficile de le formaliser (les actions du dialogue étant implémentées).

RC 10

Cependant, un premier lien peut être exploité sur les variables/objets manipulés dans ces actions. Pour cela, le lien détaillé dans 6.1.4 est utilisé. En effet, si les variables sont liées (totalement ou partiellement) aux objets du modèle de tâche, la cohérence de leur utilisation dans le code de l'action peut être vérifiée par rapport aux objets manipulés dans les actions de la tâche correspondante.

Par exemple, la tâche *Créer nouvel email* a pour action : `create($EmailSelect, 'Email1')`. Si le lien entre l'objet abstrait *Email* (composant le groupe *$EmailSelect*) et la classe Email a été préalablement établi alors une vérification de l'utilisation d'une variable de cette classe peut être faite.

$r(OA_1)=V_1$, $t(T_1)=D_1 \Leftrightarrow V_1 \in VarAct_{D1}$
avec T_1 une tâche du modèle de tâche
D_1 une transition du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions
OA_1 un objet abstrait utilisé dans l'action de T_1
V_1 une variable
$VarAct_{D1}$ l'ensemble des variables utilisées dans l'action de D_1
r la relation de correspondance entre les objets et les variables

6.3.4.2. Les conditions d'exécution

L'exécution des tâches et des transitions peut être soumise à des conditions d'exécution sur l'état du monde. Dans K-MAD(v2), ces conditions sont nommées *pré-conditions* et dans les machines à états, cette condition est nommée *garde*. Les deux expressions sont des expressions booléennes devant être évaluées à *vraie* pour permettre l'exécution de la tâche ou de la transition.

RC 11

Le même type de vérification de cohérence que pour les actions peut être mise en place, celle-ci permet de vérifier que les objets du modèle de tâche et les variables qui leur correspondent sont présents dans les deux types d'expression.

$r(OA_1)=V_1$, $t(T_1)=D_1 \Leftrightarrow V_1 \in VarGard_{D1}$

avec T_1 une tâche du modèle de tâche
D_1 une transition du modèle de dialogue
t la relation de correspondance entre les tâches et les transitions
OA_1 un objet abstrait utilisé dans l'action de T_1
V_1 une variable
$VarAct_{D1}$ l'ensemble des variables utilisées dans l'action de D_1
r la relation de correspondance entre les objets et les variables

6.4. Exemple de mise en pratique des liens entre les méta-modèles définis

Nous présentons ici, une application de nos travaux dans le processus de conception et de validation du dialogue d'une application interactive permettant l'envoi d'*emails*. Nous présentons dans une première section 6.4.1, le processus général que nous suivons et qui s'appuie sur les travaux présentés dans les sections précédentes. Chaque étape d'application est alors l'objet d'une section particulière (6.4.2, 6.4.3 et 6.4.4).

6.4.1. Le processus global

Le point de départ des processus d'application de notre approche sont composés d'un (ou plusieurs) modèle(s) de tâches. Lorsque plusieurs modèles de tâches sont spécifiés alors le processus est appliqué à chacun des modèles indépendamment les uns des autres (nous n'avons défini aucune règle dans le but de vérifier la cohérence des modèles de tâches entre eux).

Nos travaux reposant sur les modèles de tâches exprimés en K-MAD(v2) il est indispensable que ces modèles soient « transcrits » en K-MAD(v2) s'ils sont exprimés en K-MAD(v1). Cette étape automatique de transcription, qui est décrite dans la section 4.3, est une étape en préambule de l'application des règles exploitant les liaisons entre modèle de tâches et modèle de dialogue.

L'association du modèle de tâche avec le modèle de dialogue nécessite une seconde étape de réécriture du modèle de tâche. Cette seconde étape vise à limiter la présence des tâches équivalentes et le nombre de niveaux hiérarchiques composant le modèle de tâche.

Une fois les niveaux hiérarchiques et les tâches équivalentes réduites au minimum, le modèle de tâches peut être associé au dialogue en conception. Nous considérons alors deux cas pouvant se présenter :

- Cas 1 : le dialogue n'est pas encore conçu,
- Cas 2 : il existe un dialogue exprimé sous forme d'interacteurs hiérarchisés conçu pour supporter l'activité exprimée par le modèle de tâche

Dans le cas n°1, un squelette du dialogue (à compléter par le concepteur) peut être déduit des règles définies dans la section 6.3. Ces règles reposent sur les liens d'association définies dans 6.1.

Au contraire, dans le cas n°2, cas où le point initial du processus contient un dialogue exprimé (en IH) préalablement conçu (totalement ou partiellement), il est nécessaire d'expliciter les liaisons entre les modèles. Ce sont ces liaisons qui sont alors exploitées lors de la vérification des règles de cohérence.

Si toutes les règles sont vérifiées (correctes), le concepteur peut mettre fin à la conception, sinon le processus est itéré. Le concepteur peut modifier l'un des deux modèles (ou les deux), soit pour que les règles soient vérifiées, soit pour compléter le processus de conception.

Après une modification apportée aux modèles, le processus de vérification est répété. Si le modèle de tâche a été modifié, la première étape de l'itération réalisée est l'étape de réécriture pour limiter les niveaux hiérarchiques et les tâches équivalentes sinon, c'est l'étape d'association des deux modèles.

La Figure 6.4.1 représente la succession entre les différentes étapes du processus de conception proposé. Nous y avons également fait figurer des indications sur les acteurs intervenant lors de l'exécution des étapes. Dans les sections suivantes, nous décrirons les étapes 1 (section 6.4.2), 2 (section 6.4.3), 3 et 4 (section 6.4.4).

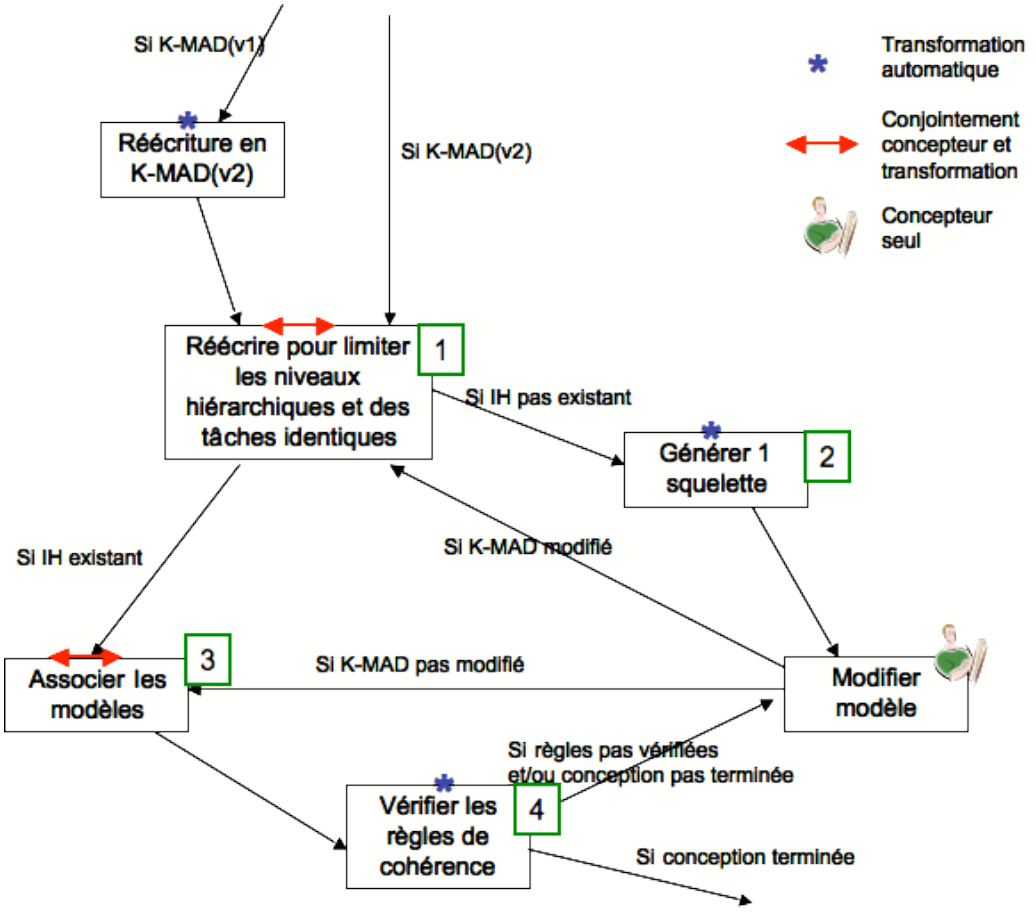


Figure 6.4.1 : Décomposition des étapes de transformation

6.4.2. Modification d'équivalence du modèle de tâches

Cette section illustre l'utilisation des transformations présentées dans la section 6.2, Pour cela, nous utilisons un modèle de tâches exprimant l'activité *Envoyer un email* (présentée sur la Figure 6.4.2).

Envoyer un email peut être réalisé en suivant deux processus différents (*alternatif*), soit en réalisant la tâche *Envoyer un nouvel email* soit en réalisant la tâche *Envoyer un email-brouillon*. *Envoyer un email* est composée de trois tâches réalisées séquentiellement (*séquentiel*) : *Créer un nouvel email*, *Editer un nouvel email* et *Envoyer*. Toutes ces tâches sont réalisées par des actions de l'utilisateur sur le système (*interactive*). *Envoyer un email-brouillon* est également décomposée en trois tâches réalisées interactivement : *Reprendre brouillon*, *Editer email* et *Envoyer*.

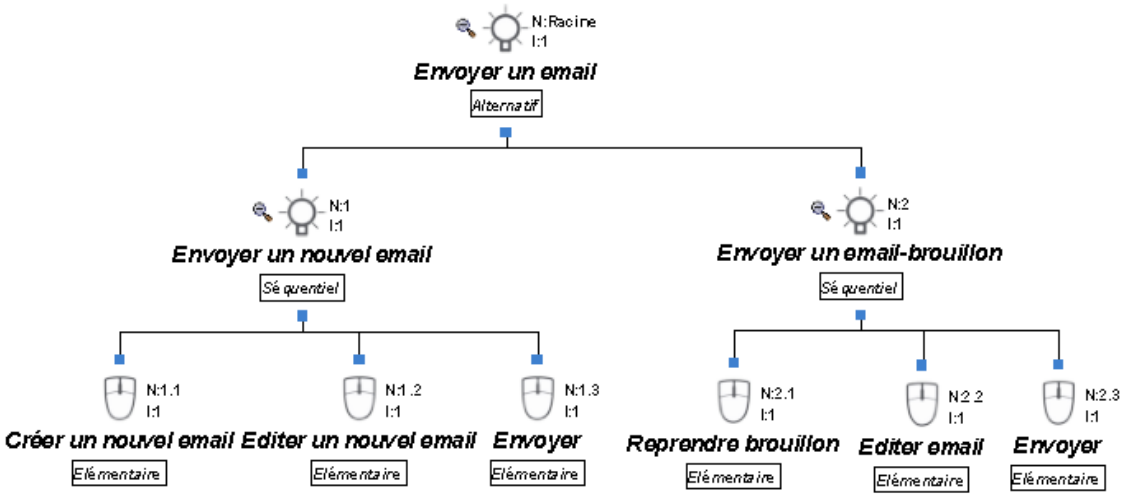


Figure 6.4.2 : Modèle de tâches initial *Envoyer un email*

6.4.2.1. Traitement des tâches équivalentes

La première étape du traitement des tâches équivalentes du modèle de tâches *Envoyer un email* (modélisé en suivant le formalisme K-MAD(v2)) est la détection des tâches équivalentes.

En suivant les conditions pour que deux tâches soient équivalentes (qui ont été présentées dans la section 6.2.2.1) le modèle de tâche *Envoyer un email* présente un et un seul cas de tâches équivalentes : les tâches *Envoyer*. En effet, les sous-tâches *Envoyer* des tâches *Envoyer un nouvel email* et *Envoyer un email-brouillon* ont même nom, mêmes conditions de déclenchement, mêmes conséquences de l'exécution. Ces tâches *Envoyer* remplissent donc toutes les conditions pour être détectées comme étant des tâches équivalentes.

De plus, les tâches *Editer un nouvel email* et *Editer email* sont des tâches qui sont détectées comme étant **potentiellement** équivalentes. Toutes les caractéristiques de ces deux tâches sont équivalentes sauf leur nom. Le concepteur peut alors modifier ces noms pour que les tâches soient considérées comme équivalentes. Dans notre exemple, nous modifions le nom de la tâche *Editer un nouvel email* pour lui donner *Editer email*.

Nous obtenons donc deux couples de tâches équivalentes : les tâches *Envoyer* (en violet sur la Figure 6.4.3) et les tâches *Editer email* (en bleu sur la Figure 6.4.3). Nous allons, à présent, illustrer l'utilisation de certaines des règles présentées dans la partie, pour essayer de supprimer la répétition de ces tâches.

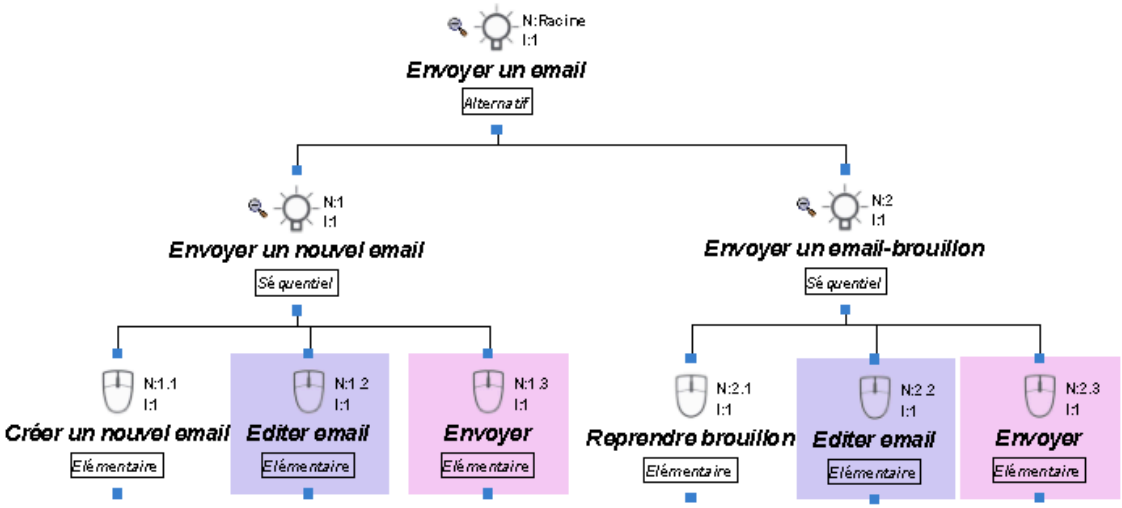


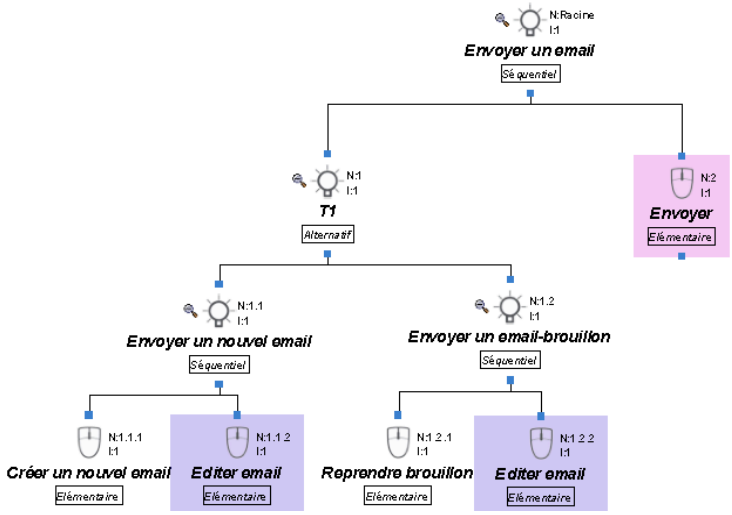
Figure 6.4.3 : Les tâches équivalentes de l'exemple

Les deux tâches équivalentes *Envoyer* ne sont pas des tâches sœurs les règles RTI1 à RTI3 de la section 6.2.2.2 ne peuvent donc pas s'appliquer. Au contraire la règle RTI4 peut être appliquée puisque ces deux tâches sont sous-tâches de tâches décomposées séquentiellement (*Envoyer un nouvel email* et *Envoyer un email-brouillon*) et que ces deux tâches sont elles mêmes sous-tâches d'une tâche *alternative* (*Envoyer un email*). La Figure 6.4.4a présente le modèle de tâches obtenu après l'application de la règle RTI4 pour la suppression d'une des tâches *Envoyer*.

Pour les mêmes raisons que pour la tâche *Envoyer*, on peut appliquer la règle RTI4 pour supprimer l'itération de la tâche *Editer email*. Le modèle obtenu est alors celui présenté sur la Figure 6.4.4b.

Ce modèle présente une incohérence par rapport au formalisme K-MAD qui est la décomposition de tâche par une seule sous-tâche. Cette incohérence est présente pour les tâches *Envoyer un nouvel email* et *Envoyer un email-brouillon*. Pour ces deux décompositions de tâches, nous pouvons appliquer la règle RTI5 indiquant que lorsqu'une tâche est décomposée par une seule sous-tâche, celle-ci remplace sa tâche mère. Ainsi les tâches *Créer un nouvel email* et *Reprendre brouillon* remplacent respectivement les tâches *Envoyer un nouvel email* et *Envoyer un email-brouillon* de l'arbre K-MAD présenté sur la Figure 6.4.5a. C'est sur ce modèle de tâches sans tâches équivalentes que nous allons chercher à limiter le nombre de niveaux hiérarchiques.

a)



b)

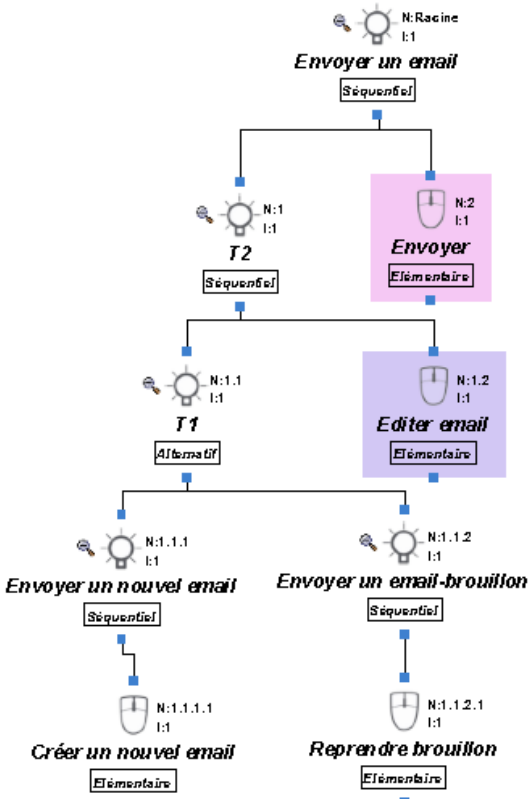


Figure 6.4.4 : modèle de tâches obtenu après traitement des tâches équivalentes a)*Envoyer* et b)*Editer email*.

6.4.2.2. Traitement des tâches intermédiaires

Une fois le nombre de tâches équivalentes limité, nous préconisons de chercher à limiter le nombre de niveaux hiérarchiques de l'arbre de tâches (dans le but de faciliter

l'association des niveaux hiérarchiques de l'arbre de tâches et du modèle de dialogue présentée dans). Le modèle initial sur lequel nous allons illustrer l'utilisation des règles présentées ci-dessus est celui présenté sur la Figure 6.4.5a. Ce modèle est composé de quatre niveaux hiérarchiques. Le premier niveau est composé de la tâche racine *Envoyer un email*. Les deux sous-tâches (T2 et *Envoyer*) de cette tâche racine composent le second niveau. T1 et *Editer email* composent le troisième niveau. Enfin, le quatrième niveau est composé des tâches *Créer nouvel email* et *Reprendre email-brouillon*.

Notons que les tâche T1 et T2 auraient pu être renommées par le concepteur. Nous avons fait le choix de ne renommer les tâches qu'une fois le traitement visant à limiter le nombre de niveaux intermédiaires effectué.

Afin de réduire le nombre de niveaux hiérarchiques dans le modèle de tâches, un parcours de l'arbre permet d'identifier si deux décompositions successives sont équivalentes. Les opérateurs de décomposition des tâches T1, T2 et *Envoyer email* sont respectivement *alternatif*, *séquentiel* et *séquentiel*. Il n'est donc possible d'appliquer l'une des règles RNH1 à RNH4 (présenté dans la section 6.2.1) que pour les tâches T2 et *Envoyer email*. Ces deux tâches ayant pour opérateur de décomposition *séquentiel*, c'est la règle RNH1 qui peut être appliquée.

La tâche intermédiaire qui est alors supprimée est la tâche T2. Cette tâche a été ajoutée lors de la suppression des tâches équivalentes (étape présentée dans 6.4.2.1). Cette tâche n'a aucune caractéristique qui empêche sa suppression ni qui doive être reportée aux caractéristiques des tâches T1 et *Editer email*.

Le modèle obtenu après l'application de cette règle est alors le modèle présenté sur la Figure 6.4.5b. Ce modèle contient l'ensemble des informations contenues dans le modèle présenté sur la Figure 6.4.2 mais ne contient plus de tâches équivalentes et plus aucune règle ne peut être appliquée pour limiter le nombre de niveaux de hiérarchisation des tâches. Il n'est donc pas nécessaire de réitérer les étapes visant à limiter le nombre de tâches équivalentes et le nombre de niveaux hiérarchiques du modèle.

Après cette étape de transformation du modèle de tâches, nous renommons la tâche T1 en *Selectionner email*.

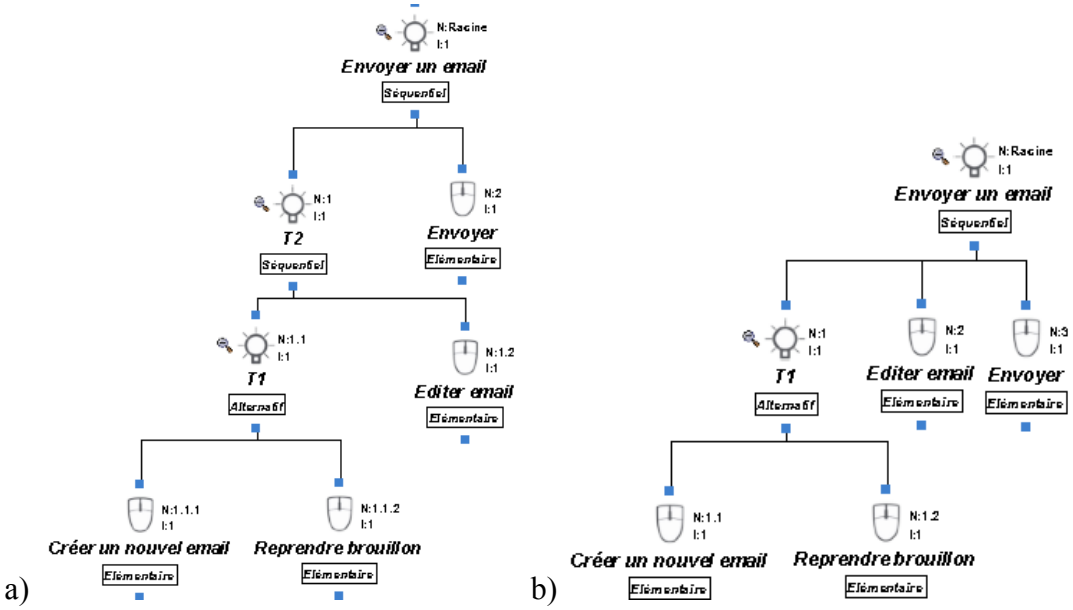


Figure 6.4.5 : modèle K-MAD avant (a) et après (b) l'application des règles de limitation du nombre de niveaux hiérarchiques

6.4.3. Générer un premier squelette

Dans le cas où aucun dialogue n'a été conçu, nous proposons la génération d'un premier squelette de modèle d'IH reposant sur l'application des règles d'association entre modèle de tâche et modèle de dialogue.

Nous partons pour cela d'un modèle de tâche K-MAD (v2) ayant subi la première étape (section 6.4.2). Pour notre cas d'étude, ce modèle est présenté sur la Figure 6.4.6.

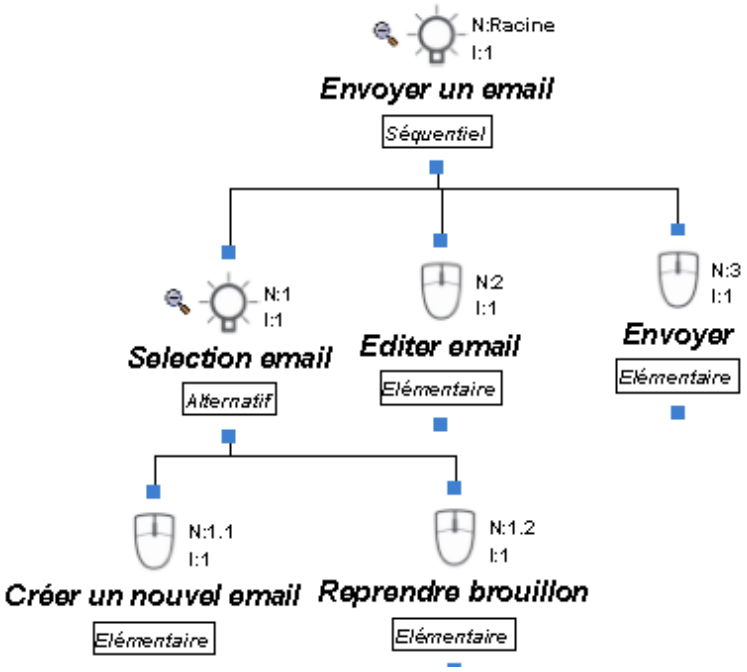


Figure 6.4.6 : Modèle K-MAD(v2) réécrit

Les niveaux hiérarchiques

Application de RC1. La règle RC1 fait associer aux niveaux hiérarchiques du modèle de tâche et du modèle de dialogue. Dans le cas de la génération du squelette, la règle RC1a est appliquée. Trois niveaux d'abstraction du dialogue sont donc conçus :

- A1 : le niveau correspondant au dialogue de l'application globale
- A2 : le niveau de dialogue de gestion des emails
- A3 : le niveau de dialogue d'un email

Les tâches décomposées et les automates

Application de RC2. La règle RC2 implique qu'il existe pour toute tâche décomposée du modèle de tâche au moins un automate de niveau hiérarchique inférieur dans le modèle de dialogue.

Dans le modèle de tâche de notre étude, seules les tâches *Selectionner email* et *Envoyer un email* sont composées et donc concernées par cette règle. La tâche *Sélectionner email* est au second niveau d'abstraction du modèle de tâche. Le niveau d'abstraction A3 du modèle de dialogue est donc composé d'au moins un automate permettant d'exécuter la tâche *sélectionner email* (produisant un jeton *SélectionnerEmail* consommé au niveau A2).

La tâche *Envoyer un email* est au premier niveau d'abstraction du modèle de tâche. Le niveau d'abstraction A2 du modèle de dialogue est donc composé d'un automate produisant un jeton *EnvoyerEmail* consommé par le niveau d'abstraction A1. La Figure 6.4.7 présente la communication des jetons entre les niveaux déduits des règles RC2.

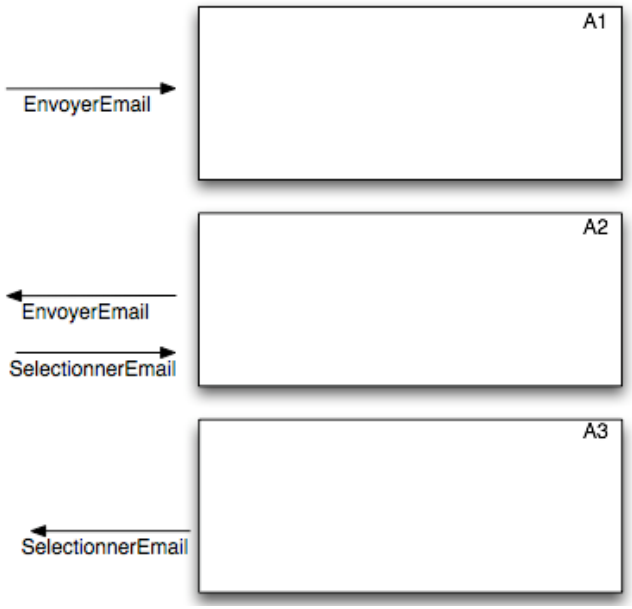


Figure 6.4.7 : Squelette généré après application de RC2

Application de RC3. La tâche *Envoyer un email* décompose ces sous-tâches par l'opérateur de décomposition *séquentiel*. La règle RC3 s'applique donc à l'automate d'A2. L'état de départ de chacune des transitions du dialogue représentant l'exécution des tâches *Editer email* et *Envoyer* est l'état d'arrivée de la précédente. La Figure 6.4.8 présente le squelette de l'automate A2 obtenu après l'utilisation de la règle RC3.

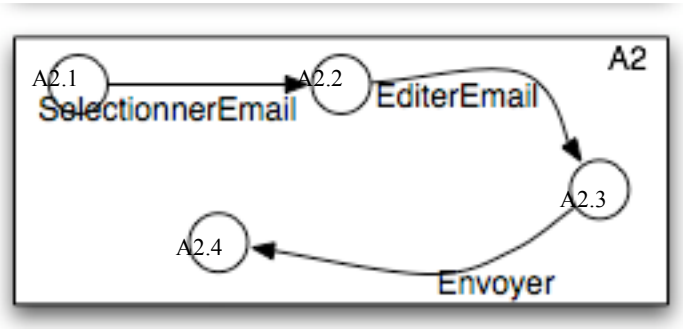


Figure 6.4.8 : Squelette généré après application de RC3

Application de RC4. La tâche *Sélectionner Email* décompose ces sous-tâches par l'opérateur de décomposition *alternatif*. La règle RC4 s'applique donc à l'automate d'A3. Les transitions représentant l'exécution des tâches *créer un nouvel email* et *Reprendre brouillon* ont donc même état de départ. La Figure 6.4.9 présente le squelette de l'automate généré après application de RC4.

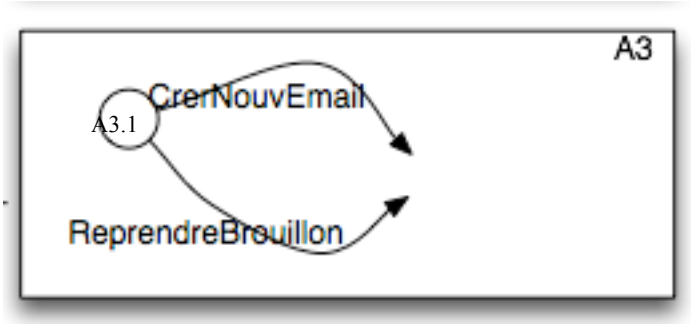


Figure 6.4.9 : Squelette généré après application de RC4

Dans le modèle présenté sur la Figure 6.4.6, aucune tâche n'est itérative, la règle RC6 ne s'applique donc pas.

Les transitions et les tâches

Application de RC7. La règle RC7 implique sur toutes les tâches interactives du modèle de tâche ont une représentation sous forme de transitions dans le modèle de dialogue. Dans notre cas d'illustration, les tâches *Créer un nouvel email*, *Reprendre brouillon*, *Editer email* et *Envoyer* sont les tâches interactives. Toutes sont représentées par des transitions dans le dialogue.

Application de RC8. La règle RC8 implique qu'aucune tâche *utilisateur* n'est représentée par une transition dans le dialogue. Le modèle de tâche représenté sur la Figure 6.4.6 ne contient aucune tâche dont l'exécutant est de type *utilisateur*.

Application de RC9. La règle RC9 impose que le dialogue ne concerne qu'une seule machine exprimée dans le modèle de tâche, cette règle est donc naturellement respectée.

Les objets et les variables

Application de RC10 et RC11. Les règles RC11 et RC12 reposent sur la définition préalable des variables du dialogue et des objets du modèle de tâche. Elle ne peuvent donc être exploitées lors de la génération d'un squelette de modèle de dialogue.

Conclusion sur la génération du squelette

L'application des règles définies dans la section 6.3 permet de générer un premier squelette de modèle de dialogue. Ce premier squelette peut servir de base à la suite de la conception du dialogue, mais ne peut nullement être suffisant.

Comme nous l'avons illustré dans cette section, le dialogue obtenu n'est pas complet. La Figure 6.4.10 présente le modèle partiel de dialogue sous forme d'IH obtenu après application des règles à partir du modèle de tâche illustré sur la Figure 6.4.6. Ce modèle nécessite donc d'être modifié pour être complété.

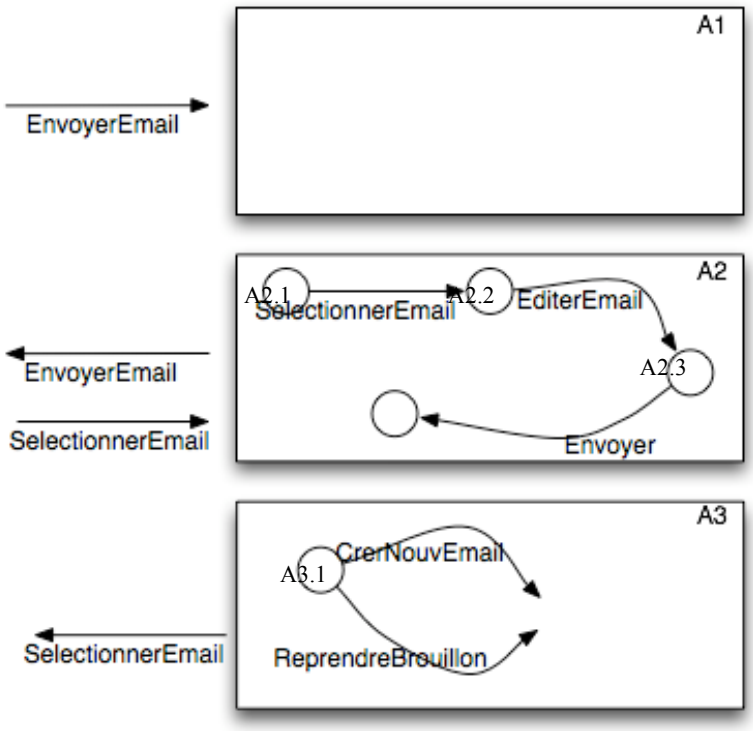
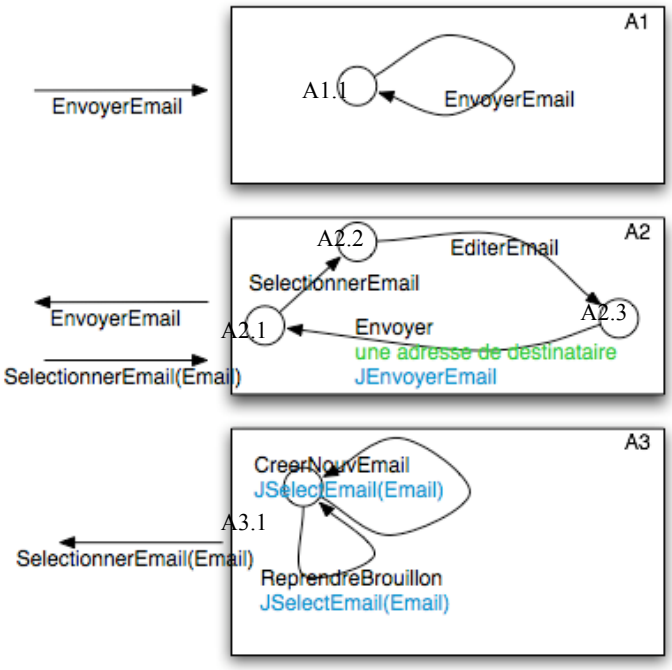


Figure 6.4.10 : Squelette du modèle de dialogue généré par l'application des règles

6.4.4. Vérification d'un modèle de dialogue existant

Des modifications du modèle de dialogue sont apportées pour compléter le dialogue obtenu et présenté sur la Figure 6.4.10. Les modifications sont présentées sur la Figure 6.4.11.

Ces modifications ont pour but d'associer les jetons produits avec les transitions qui les produisent et à permettre la réitération de l'envoi d'emails.



A1.1 : ProduireEmail
A2.1 : Pas d'email A2.2 : En Edition A2.3 : Envoyable
A3.1 : Selection

Figure 6.4.11 : Modèle de dialogue modifié

6.4.4.1. Associer les modèles

Les niveaux hiérarchiques

Cette association lie les niveaux hiérarchiques des deux modèles. Nous procédons à cette liaison comme illustré sur la Figure 6.4.12.

A chaque niveau hiérarchique du modèle de tâches correspond un et un seul niveau hiérarchique du modèle de dialogue.

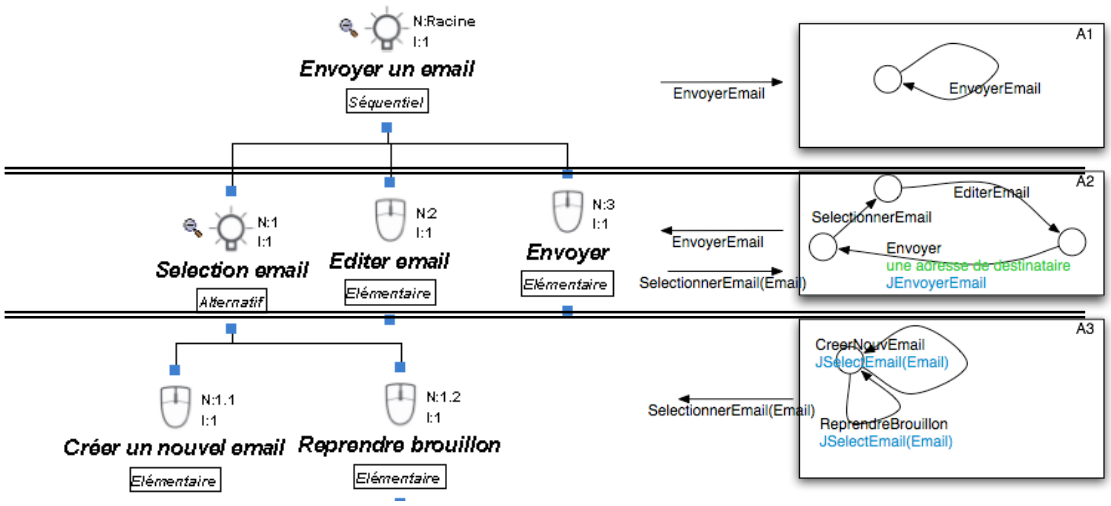


Figure 6.4.12 : Liaison par niveaux hiérarchiques des deux modèles

Les tâches décomposées et les automates

Les tâches décomposées du modèle de tâches repris sur la Figure 6.4.12 sont *Envoyer un email* et *Sélectionner Email*. Ces deux tâches sont liées à un automate du dialogue de la Figure 6.4.11 (repris sur la Figure 6.4.12). *Envoyer un email* est associé à l'automate composant le niveau d'abstraction A2 et *Sélectionner Email* est associé à l'automate composant le niveau d'abstraction A3.

Les tâches et les transitions

Dans notre exemple (illustré sur la Figure 6.4.12), toutes les tâches du modèle de tâches ont pour correspondance une et une seule transition du modèle de dialogue. Cette relation de correspondance entre les tâches et les transitions est représentée dans le Tableau 6-2.

Tâches du modèle de tâches	**Transitions du modèle de dialogue**	**Niveau hiérarchique**
Créer un nouvel email	CreerNouvEmail	A3
Reprendre brouillon	ReprendreBrouillon	A3
Selection email	SelectionnerEmail	A2
Editer email	EditerEmail	A2
Envoyer	Envoyer	A2
Envoyer un email	EnvoyerEmail	A1

Tableau 6-2 : Relation de correspondance entre tâches et transitions de notre exemple

Les objets et les variables

L'objet principalement utilisé dans le modèle de tâche K-MAD(v2) pour l'envoi d'un email est l'*email*. Cet objet est un objet abstrait dont les instances peuvent être regroupées dans un ensemble de brouillons et d'emails envoyés.

La seule variable utilisées dans le dialogue défini sur la Figure 6.4.11 est la variable *email* qui est utilisée dans la garde de la transition *Envoyer* (qui n'autorise l'envoi d'un email que si un destinataire a été spécifié).

Nous associons donc l'objet abstrait *email* du modèle de tâches avec la variable *email* du modèle de dialogue.

6.4.4.2. Vérification des règles de cohérence

Une fois les associations définies, les règles de cohérence peuvent être vérifiées (le modèle de tâches n'ayant pas été modifié, il est inutile de le retranscrire).

Vérification de RC1

L'ordre de hiérarchie des ensembles de tâches est respecté dans l'ordre hiérarchique des automates qui leur sont associés. L'ensemble des automates associés à un ensemble de tâches (ensemble composant un niveau hiérarchique de l'arbre de tâche) est défini comme étant l'ensemble des automates composés des transitions associées aux tâches (au vu des associations définies dans 6.4.4.1).

$n_{TEnvoyerEmail} > n_{TSelectionnerEmail}, n_{TEditerEmail}, n_{TEnvoyer} > n_{TCreerUnNouvelEmail}, n_{TReprendreBrouillon}$
et
$n_{DEnvoyerEmail} > n_{DSelectionnerEmail}, n_{DEditerEmail}, n_{DEnvoyer} > n_{DCreerNouvelEmail}, n_{DReprendreBrouillon}$
avec n_{Ti} et n_{Di} les niveaux hiérarchiques de la tâche T_i et de la transition D_i

Vérification de RC2

Aux deux tâches décomposées du modèle de tâche *Envoyer un email* et *Sélectionner email* correspond un automate au niveau d'abstraction inférieur.

Vérification de RC3

La tâche *Envoyer un email* a pour opérateur de décomposition l'opérateur *séquentiel*. Due à l'utilisation de cet opérateur, les transitions associées aux sous-tâches d'*Envoyer un email* se succèdent (les états de départ des unes sont les états de fin des autres). Cette règle est vérifiée dans notre cas d'étude.

$D_{SelectionnerEmail}.end = D_{EditerEmail}.beg$
et
$D_{EditerEmail}.end = D_{Envoyer}.beg$

Vérification de RC4

La tâche *Sélectionner Email* a pour opérateur de décomposition l'opérateur *alternatif*. Les transitions associées aux tâches *Créer un nouvel email* et *Reprendre brouillon* (les transitions *CreerNouvEmail* et *ReprendreBrouillon* du niveau A3) ont même état de départ dans l'automate composant A3. La règle RC4 est donc respectée.

$D_{CreerNouvEmail}.beg = D_{ReprendreBrouillon}.beg$

Vérification de RC6

Aucune tâche du modèle de tâche n'est spécifiée comme étant itérative, cependant les transitions *creerNouvEmail, ReprendreBrouillon* et *EnvEmail* ont même état de départ et d'arrivée. D'après RC6, les tâches qui leur sont associées doivent donc être itératives. La règles RC6 n'est donc pas vérifiée pour les deux modèles présentés sur la Figure 6.4.12.

Vérification de RC7 et RC8

Les tâches sont toutes associées à des transitions donc, en particulier, les tâches interactives sont toutes représentées par une transition dans le modèle de dialogue (RC8 est respectée). De plus, aucune tâche n'a *Utilisateur* comme type de tâche, donc RC7 est respectée.

Vérification de RC9

Le modèle de tâche ne fait intervenir qu'une seule machine, les transitions qui représentent ces tâches sont donc toutes réalisées sur une même machine (RC9 est respectée).

Vérification de RC10

Dans le modèle de tâche, aucune action n'a été définie, aussi, cette règle ne peut être vérifiée avec les modèles tels qu'ils sont définis.

Vérificaiton de RC11

La tâche *Envoyer* dans le modèle de tâche a une pré condition spécifiant que l'email en cours d'édition doit posséder au moins un destinataire (une adresse de destinataire). Dans l'automate A2 représenté sur la Figure 6.4.12, la transition *Envoyer* a une garde qui rend nécessaire la présence d'une adresse de destinataire de l'email pour permettre son envoi. La transition *Envoyer* et la tâche *Envoyer* sont associées, leur conditions d'exécution doivent donc utiliser des objets et variables associés. Pour la pré condition de la tâche *Envoyer* et la garde de la transition *Envoyer*, cette règle est appliquée puisque la pré condition utilise l'objet *Email* alors que la garde utilise la variable *Email* qui sont tous deux associés (voir la section 6.4.4.1).

6.4.4.3. Bilan de la vérification

La vérification des règles de cohérence entre les modèles de tâche et de dialogue conçus est résumée dans le Tableau 6-3. Ce rapport d'analyse de cohérence montre que les deux modèles présentés sur la Figure 6.4.12 ne sont pas cohérents l'un par rapport à l'autre.

La violation de la règle RC6, par exemple, montre que des tâches qui ne sont pas spécifiées comme étant itératives peuvent être réalisées itérativement par le dialogue conçu. La conception ne peut donc être considérée comme étant terminée et nécessite la modification d'un ou des deux modèles mis en relation.

RC1	OK
RC2	OK
RC3	OK
RC4	OK
RC5	Sans objet
RC6	Problème de cohérence : Transitions ayant même état de départ et d'arrivée associées à des tâches non itératives
RC7	OK
RC8	OK
RC9	OK
RC10	Sans objet
RC11	OK

Tableau 6-3 : Bilan de la vérification des règles

6.5. Conclusion sur la mise en relation des modèles par l'utilisation des méta-modèles

De par la sémantique de leurs composants, les modèles de tâches et les modèles de dialogue sont deux points de vue distincts mais très liés. Nous avons proposé dans ce chapitre d'associer certains de leurs composants en utilisant leur méta-modèles (définis dans les chapitres Chapitre 4 et Chapitre 5). Grâce à la richesse en expressivité des deux formalismes de modèles que nous avons choisis, et en nous appuyant sur leur structure hiérarchique, nous avons pu définir quatre associations différentes. Ces associations reposent permettent d'établir un lien formel entre des composants de chacun des modèles. Les quatre associations définies sont :

- association des tâches et des transitions
- association des tâches décomposées et des automates
- association des objets et des variables
- association des expressions

Ces associations définissent un lien fort entre un modèle de tâches K-MAD et un modèle de dialogue IH, composé de quatre associations (à titre de comparaison, les

approches de génération comme TERESA n'utilisent que l'association entre les tâches et les transitions).

Une fois ce lien établi, nous proposons des règles de cohérences (les RC) devant être vérifiées pour assurer au concepteur que ces deux modèles ont été conçus de manière cohérente l'un par rapport à l'autre. Ces règles se basent sur les quatre relations d'association que le concepteur a préalablement définies. Ces règles ouvrent la voie à de nombreux outils et méthodes pour permettre de concevoir et valider les applications interactives.

Au travers d'un scénario de conception, dans la dernière section de ce chapitre (§ 6.4), nous avons illustré deux cas d'utilisation de la liaison entre le méta-modèle K-MAD(v2) et le méta-modèle des IH. Elle a été utilisée pour la génération d'un squelette de dialogue et pour la vérification de la cohérence d'un modèle de tâches et d'un modèle de dialogue. Bien que relativement simple, l'illustration par ce processus de conception permet de souligner deux aspects fondamentaux de notre approche :

- la **généricité** : dès lors où les modèles respectent les méta-modèles définis, l'approche peut être appliquée
- l'**adaptation au processus de conception** d'une part en laissant de la liberté au concepteur qui apporte ses connaissances dans le processus de conception et d'autre part en permettant l'itération de modifications qui n'engendre pas la nécessité de recommencer l'ensemble des étapes préalablement réalisées

De plus, ce scénario est un des scénarios de conception possible mais nos travaux peuvent être utilisés dans d'autres scénarios. Pas exemple dans le cas où un modèle de tâches et un modèle de dialogue sont conçus séparément (éventuellement par deux concepteurs différents) et pour lesquels notre démarche peut être utilisée à des fins de validations. Un autre cas est l'utilisation de notre approche en rétro-conception dans le cas où un modèle de dialogue a été conçu et que le modèle de tâches est nécessaire (pour une re-conception par exemple). Les règles peuvent alors être utilisées pour générer un squelette de modèle de tâches au lieu du squelette de modèle de dialogue comme il a été présenté dans 6.4.3.

Chapitre 7. Conclusion et Perspectives

La conception d'applications interactives est une tâche de plus en plus complexe, en raison de la multiplicité des paramètres à prendre en compte (plateforme, interaction, dispositifs d'entrée et de sortie, environnement d'utilisation, utilisateur…) mais aussi face à la difficulté de compréhension mutuelle des intervenants. Chacun d'eux apporte en effet ses connaissances dans le processus de conception… mais aussi sa méconnaissance des autres domaines d'expertise.

Pour permettre de palier à ce problème, des modèles ont été développés. Ils sont utilisés dans les différentes étapes du processus de conception, chacun représentant un point de vue sur le système à concevoir. Le génie logiciel a depuis longtemps vu l'intérêt de l'utilisation de ces modèles pour la conception des systèmes informatiques, et a vu l'émergence d'un courant de travaux, l'ingénierie dirigée par les modèles (IDM), qui fait la part belle à ces modèles. Cependant, en plus des considérations purement fonctionnelles pour lesquelles les premiers modèles ont été développés, la conception des systèmes interactifs doit prendre en compte l'utilisateur.

Dans une conception centrée utilisateur, ce dernier est considéré comme l'intervenant le plus important mais aussi, bien souvent, le moins proche des considérations techniques mises en œuvre. Pour s'adapter à la conception centrée utilisateur (et donc prendre en compte l'utilisateur dans le processus de conception) des modèles ont à nouveau été développés, et appliqués dans le cadre d'une conception itérative faisant la part belle au prototypage et au maquettage afin de recueillir des retours de l'utilisateur. Quelques-uns des modèles intervenant pour la conception de la partie IHM d'une application interactive sont présentés sur la Figure 7.1.

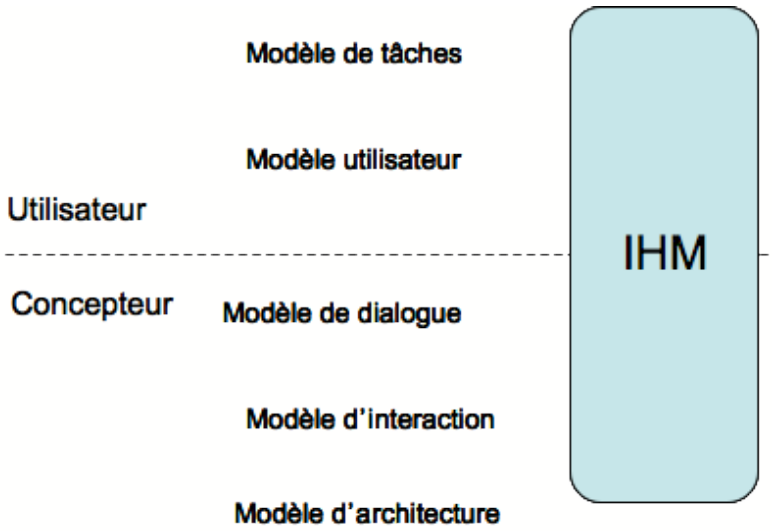


Figure 7.1 : Quelques modèles intervenant dans la conception centrée utilisateur des IHM

Parmi les modèles créés côté humain, les modèles de tâches sont un moyen pour exprimer les activités pour lesquelles l'utilisateur a besoin de l'application interactive (en référence à UML, c'est le modèle du POURQUOI manquant). Ils permettent de modéliser l'activité humaine, et se sont avérés un excellent outil pour recueillir les besoins des utilisateurs en terme d'activités à accomplir à l'aide du système à concevoir. Afin d'augmenter la compréhension des utilisateurs de ces modèles des outils des simulation ont été développés, permettant de rendre explicite la dynamique du modèle.

Du point de vue du système, le modèle de dialogue est celui qui exprime la dynamique de l'application. À première vue, le modèle de tâches d'un côté, et le modèle de dialogue de l'autre, semblent très proches. Bien que ce lien entre les deux modèles ait été souvent exploité, aucune approche ne permet de les associer tout au long du processus de conception pour permettre une conception conjointe des modèles de tâche et de dialogue.

Notre approche vise à explorer cette voie. Pour cela, nous avons commencé par étudier ces deux catégories de modèles, en nous restreignant à ceux qui présentaient le plus fort degré d'opérationnalité (existence d'outils d'édition et de simulation, existence de boîtes à outils d'implémentation). En combinant études théoriques et empiriques, nous avons pu choisir un modèle de chaque catégorie, et vérifier qu'ils pouvaient convenir à nos besoins. Ceci nous a amené à choisir le modèle de tâches K-MAD, et le modèle de dialogue des Interacteurs Hiérarchisés (IH). Afin d'être complet, nous avons également étudié les limites des approches existantes faisant intervenir ces deux catégories de modèles.

Notre étude s'inscrit pleinement dans le cadre de l'IDM. Pour pouvoir raisonner sur les modèles, nous avons défini leur méta-modèle. Ces méta-modèles ont été conçus après une étude de l'utilisation des deux formalismes, qui nous a amenée à proposer des modifications tant dans le formalisme K-MAD (débouchant sur un modèle nommé K-MAD(v2)), que dans celui des IH. Des travaux sont actuellement en cours pour permettre leur intégration dans le modèle implémenté de K-MADe, et une bibliothèque implémentant le modèle des IH est en développement.

Dns un deuxième temps, nous avons étudié le lien qui pouvait être formellement établi entre ces deux méta-modèles. Pour cela nous nous sommes basé sur la nature hiérarchique des deux formalismes pour définir une philosophie générale. Nous avons ensuite identifié quatre associations entre des éléments du méta-modèle de K-MAD et le méta-modèle des IH, puis réalisé un méta-modèle exprimant ces associations entre modèles. Ces associations impliquent des règles de cohérence pouvant être vérifiées entre les composants liés. Ces associations et règles sont indépendantes de leur utilisation, et permettent d'unir les deux modèles y compris lorsque des modifications sont apportées. C'est la première fois, à notre connaissance, que de telles règles sont établies formellement au niveau de ces deux catégories de modèles, par l'usage de leur méta-modèle. Ceci ouvre la voie sur de nombreux outils et méthodes, tant pour aider à la conception du dialogue des applications, que pour aider à la validation des applications par rapport aux tâches qu'elles sont censées supporter.

Enfin, nous avons proposé une méthode utilisant les règles définies précédemment pour effectuer la co-conception des modèles de tâches et des modèles de dialogue, que

nous avons expérimentée sur un exemple. La Figure 7.2 représente schématiquement notre approche globale.

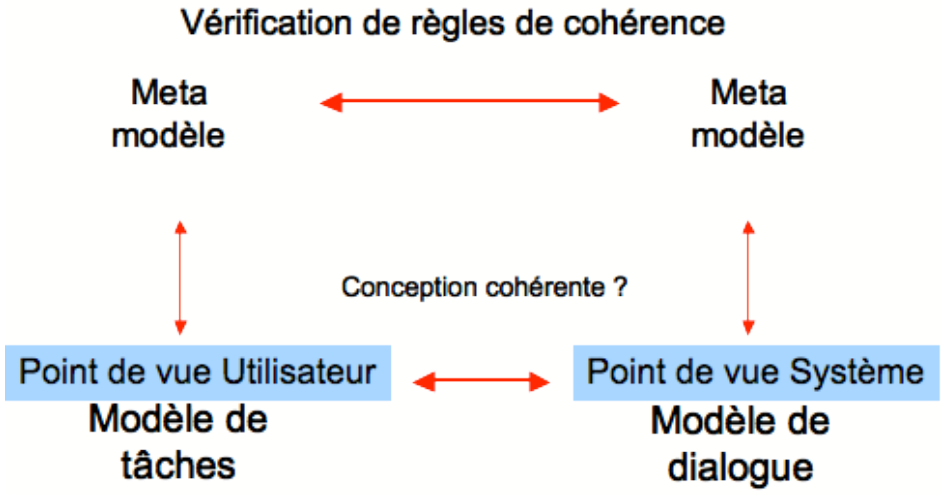


Figure 7.2 : Représentation schématique de notre approche

Dans le domaine des perspectives, nos travaux demandent dans un premier temps à être étendus. Sur le plan de l'expérimentation, notre étude est loin d'être exhaustive. Il conviendrait d'utiliser largement la méthode que nous préconisons pour en étudier la portée. Une validation par une étude empirique auprès d'utilisateurs divers est également nécessaire. Sur le plan formel, les extensions de travaux sont également possibles. Nous n'avons ainsi pas exploité totalement le lien que nous avons mis en évidence sur les expressions. À n'en pas douter, il y a dans l'usage de ces expressions, et dans celui des objets associés, de nombreuses possibilités pour étendre les possibilités de validation.

Dans un deuxième temps, un outil supportant la co-conception des modèles de tâches et de dialogue, par exemple dans l'environnement Eclipse, pourrait être conçu. Cet outil permettrait de valider notre approche de conception concurrente des modèles, dans une conception itérative telle que préconisée par la conception centrée utilisateur. Le développement d'outils de prototypage basés sur les modèles de tâches pourrait également être envisagé.

Notre approche se veut générique tant du point de vue des applications développées que du processus de conception utilisé. Le principe d'association entre les méta-modèles et de vérification de cohérence pendant la conception pourrait donc être appliqué à d'autres modèles pour compléter la conception basée sur modèle. Par exemple, des travaux récents proposent de combiner le modèle de tâches K-MAD avec le modèle d'interaction ASUR pour la conception de systèmes mixtes [Charfi 2009]. Comme nous l'avons dit dans ce rapport, la validation de la cohérence entre modèles de tâches et modèles de dialogue que nous proposons fait abstraction des techniques d'interaction utilisées bien qu'elles permettent d'animer le dialogue. Dans le cas de la conception de systèmes mixtes, ces deux approches ayant pour point commun le modèle de tâches K-MAD, pourraient être combinées pour se compléter avantageusement (Figure 7.3).

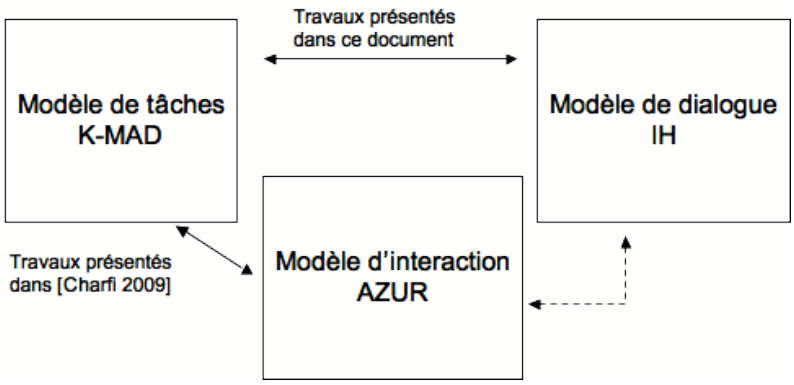


Figure 7.3 : Organisation des modèles de tâches, de dialogue et d'interaction pour une approche de conception des systèmes mixtes

Enfin, actuellement nous proposons d'établir une vérification de la cohérence en donnant pour résultat : « *oui, les deux modèles sont conçus de manière cohérente* » ou « *non les deux modèles ne sont pas conçus de manière cohérente* ». Dans le cas où la cohérence entre les deux modèles n'est pas vérifiée, le concepteur peut souhaiter une aide pour identifier précisément la raison de l'incohérence et pour la modifier.

De même, l'utilisateur peut lui aussi vouloir valider le dialogue (pour la partie non vérifiée par les règles de cohérence). Dans ce but, des travaux ont été menés comme avec l'outil PetShop [Bastide 2000] qui permet d'animer le dialogue exprimé sous forme d'ICO, ou l'approche présentée dans [Navarre, et al. 2001] qui propose d'utiliser les scénarios pour la vérification de la prise en compte d'un modèle de tâches CTT dans un dialogue ICO. Les travaux présentés dans [Sanou 2008] proposent également d'utiliser les scénarios issus des modèles de tâches en vérifiant leur réalisation effective sur l'interface développée. Combiner ces approches à celle présentée dans ce document pourrait permettre de franchir un nouveau pas dans la vérification et la validation des applications interactives.

Chapitre 8. Références

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

Aït-Ameur, Y. Développements Contrôlés de Programmes par Modélisations et Vérifications de Propriétés. 2000. Université de Poitiers.

Aït-Ameur, Y., Aït-Sadoune, I. et Baron, M. Modélisation et Validation formelles d'IHM : LOT 1 (LISI/ENSMA). 2005. University.

AMBOSS. http://wwwcs.uni-paderborn.de/cs/ag-szwillus/lehre/ws05_06/PG/PGAMBOSS.

Appert, C. et Beaudouin-Lafon, M. SMCanvas: augmenter la boîte à outils Java Swing pour prototyper des techniques d'interaction avancées. 2006. IHM. Montréal. pp. 99-106

Appert, C. et Beaudouin-Lafon, M. SwingStates: Adding State Machines to the Swing Toolkit. 2006. UIST. pp. 319-322

Appert, C., Huot, S., Dragicevic, P. et Beaudoin-Lafon, M. FlowStates : Prototypage d'applications interactives avec des flots de données et des machines à états. 2009. IHM'09. Grenoble (France). ACM. pp. 119-128

Balbo, S., Ozkan, N. et Paris, C. Choosing the Right Task-modeling Notation: A Taxonomy. The Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. A. Stanton, 2004, pp. 445-466

Baron, M. et Girard, P. SUIDT : A task model based GUI-Builder. 2002. C. Pribeanu and J. Vanderdonckt. TAMODIA : Task MOdels and DIAgrams for user interface design. Romania, Bucharest. Inforec Printing House. pp. 64-71

Baron, M. et Scapin, D. http://www-rocq.inria.fr/merlin/kmade/kmadedocfr.pdf

Barthet, M.-F. Logiciels Interactifs et Ergonomie, Modèles et méthodes de conception. Dunod Informatique, 1988.

Bass, L., Hardy, E., Hoyt, K., Little, R. et Seacord, R. The ARCH Model : SEEHEIM Revisited, The Serpent run time architecture and dialog model. 1988. University.

Bastide, R. Objets coopératifs : Un formalisme pour la modélisation des systèmes concurrents. 1992. Université de Toulouse 1.

Bastide, R. PetShop : a tool for the formal specification of CORBA systems. 2000. Conference on object Oriented Programming Systems Languages and Applications. Minneapolis, United States. ACM. pp. 167-169

Bastien, C. et Scapin, D. Ergonomic Criteria for the Evaluation of Human-Computer Interfaces. 1993. University.

Beaudouin-Lafon, M. Interaction instrumentale: de la manipulation directe à la réalité augmentée. 1997.

Beaudouin-Lafon, M. Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. 2000. CHI. The Hague, Netherlands. pp. 446-453

Blanch, R. Architecture logicielle et outils pour les interfaces hommes-machines graphiques avancées. 2005. Paris XI, Orsay.

Blanch, R. et Beaudouin-Lafon, M. Programming Rich Interactions using the Hierarchical State Machine Toolkit. 2006. AVI 2006. Venice, Italy. pp. 51-58

Boukhalfa, K., Bellatreche, L. et Caffiau, S. ParAdmin : Un Outil d'Assistance à l'Administration et Tuning d'un Entrepôt de Données. 2008.

Caffiau, S., Girard, P., Scapin, D. L., Guittet, L. et Sanou, L. Assessment of Object Use for Task Modeling. 2008a. P. Forbrig and F. Paternò. Engineering Interactive Systems (HCSE 2008 and TAMODIA 2008). Pisa, Italy. Springer (LNCS 5247). pp. 14-28

Caffiau, S., Guittet, L., Scapin, D. L. et Sanou, L. Utiliser les outils de simulation des modèles de tâches pour la validation des besoins utilisateur : une revue des problèmes (Poster). 2008b. ERGO'IA. Biarritz, France. pp. 257-258

Calvary, G., Coutaz, J. et Thevenin, D. A unifying reference framework for the development of Plastic User Interfaces. 2001. R. M. Little and L. Nigay. Engineering for Human-Computer Interaction (8th IFIP International Conference, EHCI'01, Toronto, Canada, May 2001). Canada. Springer. pp. 173-192

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L. et Vanderdonckt, J. A unifying reference framework for multi-target user interfaces. Interacting With Computers, 2003, 15/3, pp. 289-308

Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Souchon, N., Bouillon, L., Florins, M. et Vanderdonckt, J. Platicity of User Interfaces: A Revised Reference Framework. 2002.

Card, S., Moran, T. et Newell, A. The Psychology of Human-Computer Interaction. Lawrence Erlbaum Associates, 1983.

Charfi, S. Conception et Evaluation des Systèmes Interactifs Mixtes selon une Approche Centrée Utilisateur. 2009. Université Paul Sabatier.

Couix, S. Usages et construction des modèles de tâches dans la pratique de l'ergonomie : une étude exploratoire. 2007. University.

Coutaz, J. PAC, an Implementation Model for the User Interface. 1987. IFIP TC13 Human-Computer Interaction (INTERACT'87). Stuttgart. North-Holland. pp. 431-436

Coutaz, J. Interfaces Homme-Ordinateur, Conception et Réalisation. Dunod Informatique, 1990.

Coutaz, J. et Nigay, L. Seeheim et Architecture Multi-agent. 1991. Interfaces Homme-Machine. Dourdan. pp.

Depaulis, F., Jambon, F., Girard, P. et Guittet, L. Le modèle d'architecture logicielle H4 : Principes, usages, outils et retours d'expérience dans les applications de conception technique. Revue d'Interaction Homme-Machine, 2006, 7(1), pp. 93-129

Dix, A. Tasks = Data + Action + Context: Automated Task Assistance through Data-Oriented Analysis (invited paper). 2008. P. Forbrig and F. Paternò. Engineering Interactive Systems 2008 (HCSE 2008 and TAMODIA 2008). Pisa, Italy. Springer (LNCS 5247). pp. 1-13

Dix, A., Finlay, J., Abowd, G. et Beale, R. Human-Computer Interaction. Prentice Hall, 1993.

Dix, A. J. Formal Methods for Interactive Systems. Academic Press, 1991.

Dragicevic, P. et Fekete, D. Support for input adaptability in the ICON toolkit. 2004. ICMI'04. ACM. pp. 212-219

Eisenstein, J., Puerta, A. et Vanderdonckt, J. XIML: Towards a Universal User Interface Markup Language. 2001, pp.

EXPRESS. The EXPRESS language reference manual. 1994. University.

Falzon, P. nature, objectifs et connaissances de l'ergonomie. Ergonomie, P. Falzon, 2004, pp. 17-35

Fekete, J.-D. Les trois services du noyau sémantique indispensables à l'IHM. 1996. J. Coutaz. Journées Francophones sur l'Ingénierie de l'Interaction Homme-Machine (IHM'96). Grenoble. Cépaduès. pp. 45-50

Gamboa, R. F. Spécification et implémentation d'ALACIE : Atelier Logiciel d'Aide à la Conception d4interfaces Ergonomiques. 1998. Paris XI.

Gamboa, R. F. et Scapin, D. L. Editing MAD* task description for specifying user interfaces, at both semantic and presentation levels. 1997. M. D. Harrison and J. C. Torres. Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'97). Granada, Spain. Springer-Verlag. pp. 193-208

Gauffre, G. Couplage de la Modélisation et Développement des Systèmes interactifs Mixtes. 2009. Université de Toulouse.

Girard, P. Ingénierie des systèmes interactifs : vers des méthodes formelles intégrant l'utilisateur. 2000. Université de Poitiers.

Girard, P., Pierra, G. et Guittet, L. Les interacteurs hiérarchisés : une architecture orientée tâches pour la conception des dialogues. Revue d'Automatique et de Productique Appliquée (RAPA), 1995, 8, 2-3. pp. 235-240

Graham, I. Object oriented methods. Addison-Wesley Publishing Compagny, 1991.

Gray, P., England, D. et McGowan, S. XUAN: Enhancing the UAN to capture temporal relation among actions. 1994. University.

Green, M. W. A Survey of three Dialogue Models. ACM Transactions on Graphics, 1986, 5, 3. pp. 244-275

Guittet, L. Contribution à l'Ingénierie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO. 1995. Université de Poitiers.

Halbwachs, N., Caspi, P., Raymond, P. et Pilaud, D. The synchronous dataflow programming language lustre. 1991. Proceedings de IEEE. pp. 1305-1320

Harel, D. Statecharts: a visual formalism for complex systems. Science of Computer Programming, 1987, 8, 3. pp. 231-274

Hartson, H. R. et Gray, P. Temporal aspects of Tasks in the User Action Notation. Human-Computer Interaction, 1992, 7, 1. pp. 1-45

Hix, D. et Hartson, H. R. Developping user interfaces: Ensuring usability through product & process. John Wiley & Sons, inc., 1993.

Jacob, R. J. K. Using Formal Specification in the Design of Human-Computer Interface. 1982. Human Factors in computing systems. pp. 315-321

Jambon, F. Error Recovery Representations in Interactive System Development. 1997. C. Stephanidis and N. Carbonell. Third Annual ERCIM Workshop on "User Interfaces for All". Obernai, France. INRIA Lorraine. pp. 177-182

Jaulent, P. Génie logiciel : les méthodes. Armand Colin, 1992.

Johnson, P., Johnson, H., Waddington, R. et Shouls, A. Task-Related Knowledge Structures: Analysis, Modelling and Application. People and Computers: from research to implementations, 1988, pp. 35-62

Lewis, C. et Norman, D. A. Designing for Error. User Centered System Design / New Perspectives on Human-Computer Interaction, D. A. Norman and S. W. Draper, 1986, pp. 411-432

Limbourg, Q., Pribeanu, C. et Vanderdonckt, J. Towards Uniformed Task Models in a Model-Based Approch. 2001a. Proceedings DSVIS. pp.

Limbourg, Q., Pribeanu, C. et Vanderdonckt, J. Uniformation of Task Models in Model-Based Approaches. 2001b. University.

Limbourg, Q. et Vanderdonckt, J. Comparing Task Models for User Interface Design (Chapter 6). The Handbook of Task Analysis for Human-Computer Interaction, D. Diaper and N. Stanton, 2003, pp.

Lucquiaud, V. Proposition d'un noyau et d'une structure pour les modèles de tâches orientés utilisateurs. 2005a. 17th French-speacking conference on Human-computer interaction. Toulouse. pp. 83-90

Lucquiaud, V. Sémantique et Outil pour la Modélisation des Tâches Utilisateur: N-MDA. 2005b. Poitiers.

Luyten, K. Dynamic User Interface Generation for Mobile and Embedded Systems with Model-Based User Interface Development. 2004. University Limburg.

Luyten, K., Clerckx, T., Coninx, K. et Vanderdonckt, J. Derivation of a Dialog Model from a Task Model by Activity Chain Extraction. 2003. DSV-IS'2003. Funchal, Madeira Island, Portugal. Springr-Verlag. pp.

Marca, D. et McGowan, C. SADT: structured analysis and design technique. McGraw-Hill, 1987.

mde, P. http://planet-mde.org/.

Minsky, M. The society of Mind. Touchstone. 1988.

Morfeo, P. http://forge.morfeo-project.org/wiki_en/index.php/Domain_Model.

Mori, G., Paternò, F. et Santoro, C. Tool Support for Designing Nomadic Applications. 2003. Intelligent User Interfaces (IUI'2003). Miami, Floride. pp. 141-148

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

Myers, A. B. A brief history of human-computer interaction technology. Interactions, 1998, 5, 2. pp. 44-54

Navarre, D., Palanque, P., Paterno, F., Santoro, C. et Bastide, R. A Tool Suite for Integrating Task and System Models through Scenarios. Design, Specification, and Verification -Interactive Systems (DSV-IS'01), C. Johnson, 2001, pp. 88-113

Nielsen, J. Usability Engineering. ISBN 0-12-518405-0. Academic Press, 1993.

Norman, D. A. Stages and levels in human-machine interaction. International Journal of Man-machine studies, 1984, 21, pp. 365-375

Norman, D. A. The Psychology of Everyday Things. Harper & Collins, 1988.

Norman, D. A. et Draper, S. W. User Centered System Design. Lawrence Erlbaum Associates, 1986.

Normand, V. Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation. 1992. Université Joseph Fourier.

Object Management Group. www.uml.org/.

Olsen, D. R. User Interface Management Systems: Models and Algorithms. Morgan Kaufmann Publisher, 1992.

Palanque, P. Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur. 1992. Université de Toulouse I.

Palanque, P. et Bastide, R. Petri-Net Based Design of User Interfaces using the Interactive Cooperative Objects Formalism. 1994. F. Paternò. Interactive Systems: Design, Specification, and Verification (DSV-IS'94). Bocca di Magra, Italy. Springer. pp. 383-400

Palanque, P., Bastide, R. et Winckler, M. Automatic Generation of Interactive Systems: Why A Task Model is not Enough. Human - Computer Interaction, C. S. a. D. H. Julie Jacko, 2003, Theory and Practive, Vol. 1, pp. 198-202

Palanque, P., Paternò, F. et Wright, P. Case Study: User Interfaces for Air-Traffic Control. 1998. Workshop on Designing User Interfaces for Safety Critical Systems (CHI'98). Los Angeles, USA. ACM/SIGCHI. pp.

Paterno, F. ConcurTaskTrees: An Engineered Notation for Task Models. The Handbook of tTask Analysis for Human-Computer Interaction, D. Diaper and N. A. Stanton, 2004, pp. 483-501

Paternò, F. Model-Based Design and Evaluation of Interactive Applications. Springer, 2001.

Paternò, F. et Faconti, G. P. On the LOTOS use to describe graphical interaction. 1992, pp. 155-173

Paternò, F., Mancini, C. et Meniconi, S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. 1997. IFIP TC13 human-computer interaction conference (INTERACT'97). Sydney, Australia. pp. 362-369

Pfaff, G. E. User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim. 1985. Springer-Verlag.

Pinheiro da Silva, P. User Interface Declarative Models and Development Environments : A survey. 2000. P. Palanque and F. Paterno. 7th Eurographics workshop on Design, Specification and Verification of Interactive Systems DSVIS 2000. Limerick, Ireland. Springer. pp. 207-226

Pribeanu, C. et Vanderdonckt, J. A methodological approach to tack-based design of user interfaces. Studies in Informatics and Control, 2002, 11, N°2, pp. 145-158

Puerta, A. The MECANO project : comprehensive and integrated support for Model-Based Interface development. 1996. J. Vanderdonckt. Computer-Aided Design of User interface (CADUI'96). Namur, Belgium. Presse Universitaire de Namur. pp. 19-35

Puerta, A. R. A Model-Based Interface Development Environment. IEEE Software, 1997, 14, 4. pp. 40-47

Rasmussen, J., Pejtersen, A. M. et Goodstein, L. P. Cognitive Systems Engineering. John Wiley & Sons, 1994.

Ricard, E. et Pollet, Y. Du modèle de tâches à l'interface homme/machine GLADIS/ALADIN : une approche industrielle. 1994. R. Patesson. ERGO-IA. Bayonne. pp. 445-457

Samaan, K. Prise en compte du modèle d'interaction dans le processus de construction et d'adaptation d'applications interactives. 2006.

Sanou, L. Définition et réalisation d'une boîte à outils générique dédiée à la Programmation sur Exemple. 2008. Ecole Nationale Supérieure de Mécanique et d'Aérotechnique.

Scapin, D. Guide ergonomique de conception des interfaces homme-machine. 1986. University.

Scapin, D. et Bastien, J.-M. C. Analyse des tâches et aide ergonomique à la conception : l'approche MAD* (chapitre 3). Analyse et conception de l'I.H.M. / Interaction Homme-Machine pour les S.I. vol.1, C. Kolski, 2001, 1, pp.

Scapin, D. L. et Pierret-Golbreich, C. MAD : Une méthode analytique de description des tâches. 1989. Colloque sur l'Ingénierie des Interfaces Homme-Machine (IHM'89). Sophia-Antipolis, France. pp. 131-148

Schmidt, D. C. Model-Driven Engineering. IEEE Computer, 2006, pp.

Sebillotte, S. Décrire des tâches selon les objectifs des opérateurs, de l'interview à la formalisation. Le Travail Humain 1991, 54, 3. pp. 193-223

Sebillotte, S. Méthodologie pratique d'analyse de la tâche en vue d'extraction de caractéristiques pertinentes pour la conception d'interfaces. 1994. University.

Sebillotte, S. et Scapin, D. L. From Users' Task Knowledge to High-Level Interface Specification. International Journal of Human-Computer Interaction, 1994, 6, 1. pp. 1-15

Shneiderman, B. Direct Manipulation: a Step beyond Programming Languages. IEEE Computer, 1983, 16, 8. pp. 57-69

Shneiderman, B. Designing the User Interface. 3. Addison-Wesley, 1998.

Sibertin-Blanc, C. High Level Petri Nets with Data Structure. 1985. 6th European Workshop on Petri Nets and Applications. Espoo, Finland. pp.

Sinning, D., Wurdel, M., Forbrig, P., Chalin, P. et Khendek, F. Practical Extensions for Task Models. 2007. S. LNCS. TAMODIA 2007. Toulouse. pp. 42-55

Souchon, N. et Vanderdonckt, J. A Review of XML-compliant User Interface Description Languages. DSV-IS 2003, J. A. Jorge, N. J. Numes and J. F. e. Cunha, 2003, LNCS 2844, pp. 377-391

Systems, I. I. P. Definition of the Temporal Ordering Specification Language LOTOS. 1984. University.

Szekely, P. Retrospective and challenge for Model Based Interface Development. Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96), F. Bodart and J. Vanderdonckt, 1996, pp. 1-27

Szekely, P., Luo, P. et Neches, R. Beyond Interface Builders: Model-Based Interface Tools. 1993. InterCHI93. pp. 383-390

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J. et E. Salcher. Declarative interface models for user interface construction tools : the MASTERMIND approach. 1995. L. J. Bass and C. Unger. IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction (EHCI'95). Grand Targhee Resort (Yellowstone Park), USA. Chapman & Hall. pp. 120-150

Tarby, J.-C. Gestion Automatique de Dialogue Homme-Machine à partir de spécification conceptuelles. 1993. Université de Toulouse I.

Tarby, J.-C. et Barthet, M.-F. Analyse et modélisation des tâches dans la conception des systèmes d'information. Analyse et conception de l'IHM, Interaction Homme-Machine pour les systèmes d'information, K. C., 2001, 1, pp.

Tardieu, H., Rochfeld, A. et Colletti, R. La méthode Merise - Tome 1 Principes et outils. Editions d'organisation, 1983.

Texier, G. Contribution à l'ingéniérie des systèmes interactifs : Un environnement de conception graphique d'applications spécialisées de conception. 2000. Université de Poitiers.

Thévenin, D. Adaptation en Interaction Homme-Machine : le cas de la plasticité. 2001. Université Joseph Fourier.

Thimbleby, H. Press On: principles of interaction programming. The MIT Press, 2007.

van der Veer, G. C. GTA: Groupware Task Analysis - Modeling Complexity. Acta Psychologica, 1996, pp. 297-322

van der Veer, G. C., Hoeve, M. et Lenting, B. F. Modeling Complex Work Systems - Method meets Reality. 1996. T. R. G. Green, J. J. Canas and C. P. Warren. 8th

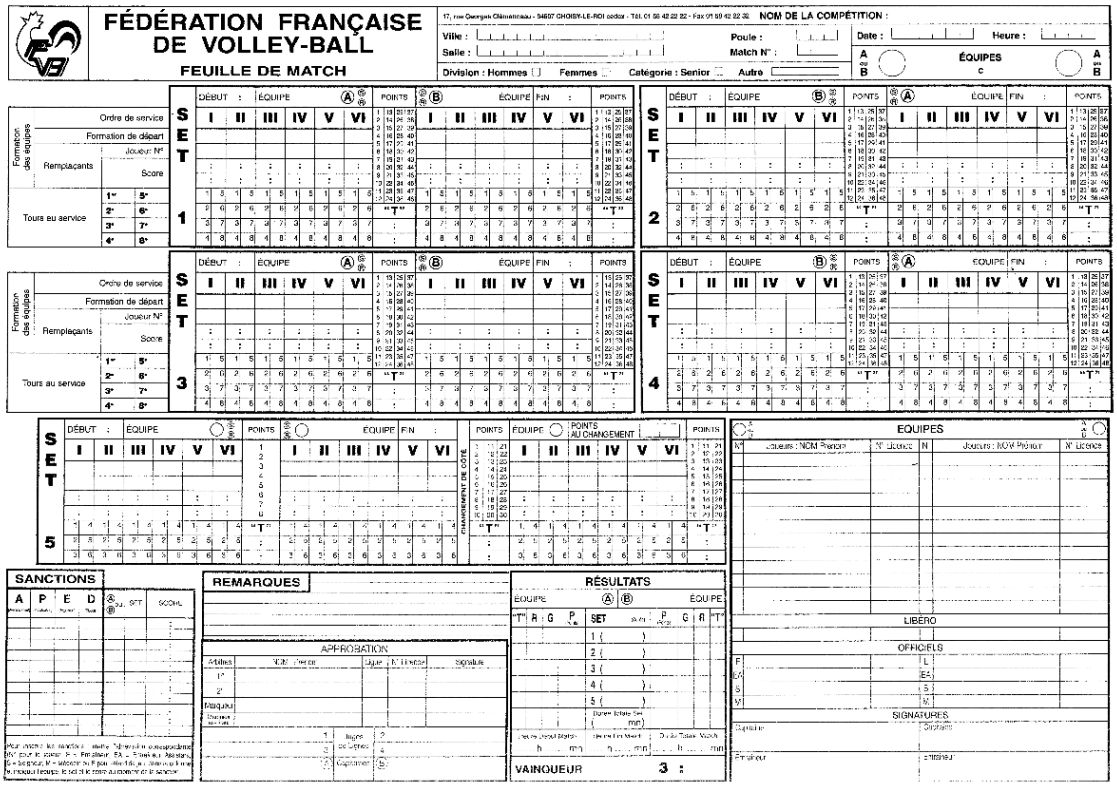
European Conference on Cognitive Ergonomics (EACE). INRIA, Le Chesnay. Cognition and the worksystem. pp. 115-120

van Welie, M., van der Veer, G. C. et Eliëns, A. An Ontology for Task World Models. DSV-IS, 1998, pp.

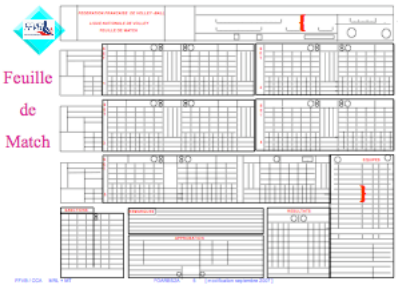
Chapitre 9. Annexes

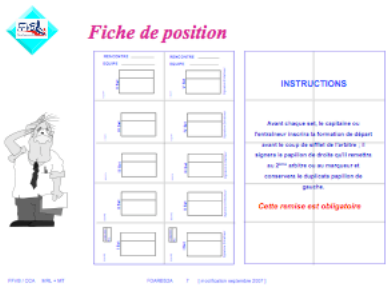
9.1. Annexe 1 : Les documents de l'évaluation

9.1.1. Un exemple de feuille de match



9.1.2. Instructions de la Fédération Française de Volley-ball pour le marquage de la feuille de match (FFVB)





PT1 - Avant le début de la rencontre : *(1aBFI)

((1a) = point de la directive pour remplir la feuille de match du BFI)

* Compléter les lignes ou les cases numérotées de 1 à 9 de la façon suivante (en MAJUSCULES D'IMPRIMERIE) :

- N° 1 : nom de la compétition
- N° 2 : ville de la rencontre
- N° 3 : salle de la rencontre
- N° 4 : indiquer le niveau de compétition
- N° 5 : numéro du match
- N° 6 : catégorie : mettre une croix dans le carré "Hommes" ou "Femmes"
- N° 7 : date de la rencontre

NOTE Laisser en blanc les ◯ A ou B précédents ou suivants les noms des équipes.

- N° 8 : heure prévue de la rencontre
- N° 9 : nom des équipes (*inscrire à gauche, l'équipe qui reçoit, ou première citée par le calendrier officiel, si terrain neutre*)

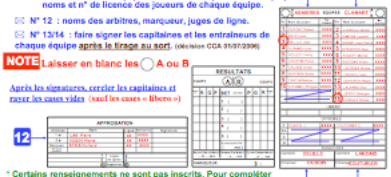
PT1 - Avant le début de la rencontre : *(1b)

* Compléter les lignes ou les cases numérotées de 10 à 14 de la façon suivante :

- N° 10/11 : noms des équipes - n° de maillot (ordre numérique) noms et n° de licence des joueurs de chaque équipe.
- N° 12 : noms des arbitres, marqueur, juges de ligne.
- N° 13/14 : faire signer les capitaines et les entraîneurs de chaque équipe après le tirage au sort. (décision CCA 31/07/2006)

NOTE Laisser en blanc les ◯ A ou B

Après les signatures, cercler les capitaines et rayer les cases vides (sauf les cases « libero »)

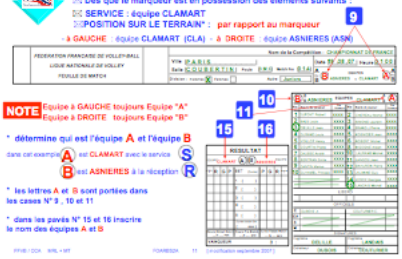


* Certains renseignements ne sont pas inscrits. Pour compléter ces pavés, il est nécessaire de connaître la position des équipes sur le terrain, élément connu après tirage au sort.

PT2 - Après le tirage au sort : *(2)

- Dès que le marqueur est en possession des éléments suivants :
- SERVICE : équipe CLAMART
- POSITION SUR LE TERRAIN* : par rapport au marqueur

- à GAUCHE : équipe CLAMART (CLA) - à DROITE : équipe ASNIERES (ASN)



NOTE Equipe à GAUCHE toujours Equipe "A"
Equipe à DROITE toujours Equipe "B"

* détermine qui est l'équipe A et l'équipe B
dans cet exemple : A est CLAMART avec le service S
B est ASNIERES à la réception R

* les lettres A et B sont portées dans les cases N° 9 , 10 et 11

* dans les pavés N° 15 et 16 inscrire le nom des équipes A et B

PT2 - Après le tirage au sort :*(2)

* 1er SET : compléter :

- EQUIPE A : CLAMART - Cocher la lettre S car cette équipe détient le service
- EQUIPE B : ASNIERES - Cocher la lettre R car cette équipe est en réception

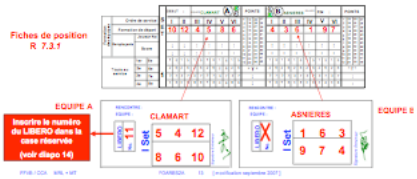
NOTE *Dans un but de simplification et pour éviter les risques d'erreur, il est logique de remplir les SETS 2 et 3 en suivant bien les lettres B - A et A - B en conservant toujours le R pour l'équipe située à droite et le S pour celle située à gauche.* (Pour cet exemple bien entendu.)

PT2 - Après le tirage au sort : *(2)

* Inscrire la position des joueurs dans les pavés "Formation de départ" dans les cases se trouvant sous les numéros I à VI : suivant l'ordre indiqué sur les fiches de position; le N° I étant le joueur arrière droit,
conformément à la *Règle R 7.4.1*

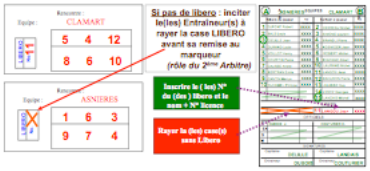
Vérifier Systématiquement que ces numéros figurent dans la composition de l'équipe.

Fiches de position R 7.3.1



PT2 - Après la remise des fiches de position : *(2)

- Dès que le marqueur a fini d'inscrire les formations de départ,
- Dans les cases spécifiques « LIBERO »,
- Enregistrer le(s) libero(s) ou rayer la (les) case(s) sans libero



PT3 - Pendant la rencontre *(3 a)

- Inscrire dans la case "DEBUT", l'heure officielle de début de Set (en heures et minutes).
si retard, le justifier dans la case "REMARQUES"
- Contrôler la conformité du numéro du joueur au service (CLAMART N° 10) avec celui de la case correspondante du pavé "Tours au service".
- Cocher d'un petit trait oblique, le numéro de cette case.
en cas de faute de rotation, *laisser le service s'effectuer*, puis prévenir le second arbitre.
- Dans la case du joueur N° 4 de l'équipe B "ASNIERES", qui est en réception R porter une croix. En effet, ce joueur, au premier tour, n'effectue pas de service.

PT3 - Pendant la rencontre *(3a 4)

RAPPEL : les 4 premiers Sets se jouent en marque continue en 25 pts, avec 2 pts d'écart, sans limite.

- Les points acquis sont cochés au fur et à mesure dans la colonne "POINTS".
- Au moment où le service est perdu, on porte dans la case "TOURS AU SERVICE" le nombre de points acquis par l'équipe et ainsi de suite : exemple équipe A "CLA".

- Lorsque les joueurs II-III.... VI sont au service, au fur et à mesure, les scores acquis par l'équipe sont portés dans la case 1 *au moment où ils perdent le service.*

Avec la marque continue, IL NE PEUT JAMAIS Y AVOIR DEUX FOIS DE SUITE LE MEME SCORE

PT4 - Changement de joueur *(3 d)

Equipe CLAMART le joueur N° 12 est remplacé par le N°3 sur le score de 7 /10 en sa défaveur.

☞ Lors de la demande, vérifier systématiquement l'existence du remplaçant dans la CASE composition de l'équipe et la possibilité du changement. R.15.6

☞ Inscrire

✎ le numéro du remplaçant sous celui du remplacé,

✎ le score lors de cette opération (celui de l'équipe qui a demandé, en premier).

12
3
7 :10

PT4 - Remplacement de joueur *(3 d)

❖ CONSEIL d'amélioration. ©

1°) Lors de la demande, vérifier systématiquement l'existence du remplaçant dans la case composition de l'équipe et la possibilité du changement.

2°) Dès le contrôle effectué, lever un avant-bras et la main : cela signifie que le remplacement est autorisé.

3°) Inscrire le numéro du remplaçant et le score de l'équipe qui a demandé.

4°) puis poser le stylo et lever les deux bras vers le deuxième arbitre : cela signifie que le jeu peut reprendre.

PT4 - Changement de joueur *(3d)

☞ Lorsque le titulaire reprend sa place :

✎ inscrire dans la seconde case, sous la précédente, le score de l'équipe au moment où le changement est effectué, et entourer le numéro du joueur sorti.

Equipe CLAMART : le joueur N° 12 reprend sa place de titulaire sur le score de 9/14 en sa défaveur.

ATTENTION : si remplacement du "LIBERO", cela ne peut être qu'un remplacement exceptionnel, donc inscrit dans la case "REMARQUES"

12
3
7 :10
9 :14

PT5 - Temps mort *(3c)

Au bas de la colonne "POINTS" de l'équipe qui a demandé, inscrire le score au temps mort dans l'une puis dans l'autre des deux cases affectées à cet usage,
*(celui de l'équipe qui en fait la demande, en premier).

❖ EXEMPLE :

☞ si A CLAMART, demande le temps mort, inscrire **09 : 14**

☞ si B ASNIERES, demande le temps mort, inscrire **14 : 09**

PT6 - Avertissement / Pénalisation *(3e)

* Utiliser le pavé "SANCTIONS" : ©

✎ pour inscrire les sanctions => mettre l'abréviation correspondante (No pour le joueur, E pour Entraîneur, EA pour Entraîneur Assistant, S pour Soigneur, M pour Médecin ou R pour Retard de jeu) dans la colonne et indiquer l'équipe (A ou B), puis le set (1 à 5) et enfin le score au moment de la sanction.

Colonne A : il ne peut figurer que la lettre : R

Colonne P : il peut figurer R ou N° /E/AE/S/M

Colonnes E et D : il ne peut figurer que des N° /E/AE/S/M

SANCTIONS

"A"	P	E	D	A/B	Set	Score
R				B	1	4
5				A	3	7

EXEMPLES :

1) Avertissement pour Retard de jeu à l'équipe d'ASNIERES score 4/3 au cours du 1er Set (geste 25 : R 16.2.2)

2) Pénalisation pour le joueur No 5 de Clamart, score 7/13 au 3ème Set.

✎ pour une Pénalisation entourer le Point de pénalité marqué par l'équipe adverse dans la colonne "POINTS"

PT6 -Tableau des sanctions *(3.e.1/2)

❖ CONSEIL d'amélioration. ©

Attention :

1) L'Avertissement pour retard de jeu = R dans la 1ère colonne A
- Matérialisé par l'arbitre en 1er avec le geste 25 sans carton jaune Règle 16.2.2
- C'est une sanction collective donc pas de N° de joueur ni de perte de point.

2) La Pénalisation : le P de pénalisation doit vous faire penser à : Plus un Point à l'équipe adverse
Matérialisée par Ex : (2)
- Soit pour "retard de jeu ", (sanction collective) matérialisée par l'arbitre en 1er avec le geste 25 et le carton jaune : Règle 16.2.3
- Soit pour un joueur (sanction individuelle) donc il faut un N° ou E/AE/S/M, matérialisée avec le geste 6 avec le carton jaune et inscrite pour les 2 cas dans la 2ème colonne. Règle 21.3.1

3) L'Expulsion (sanction individuelle) donc il faut un N° ou E/AE/S/M, matérialisée par l'arbitre en 1er avec le geste 7 avec le carton rouge et inscrite dans la 3ème colonne.
Le joueur reste sous le contrôle des arbitres : Règle 21.3.2.1

4) La Disqualification (sanction individuelle) donc il faut un N° de joueur, matérialisée par l'arbitre en 1er avec le geste 8 avec les cartons jaune/rouge et inscrite dans la 4ème colonne.
Le D de disqualification = Définitif = Dehors : donc plus sous le contrôle des arbitres.
Règles 21.3.3.1

PT7 - 5ème SET *(3b)

* Le tirage au sort effectué par le premier arbitre détermine :

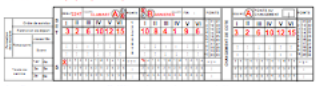
☞ la position des équipes A et B sur le terrain, l'équipe avec le service.

* Dans notre Exemple : A reste l'équipe "CLAMART", en Réception,
B reste l'équipe "ASNIERES", avec le Service.

Cette appellation ne peut être modifiée, ceci dans le seul but de présenter une récapitulation correcte du tableau "TOTAL et RESULTATS".

1) Inscrire la composition de l'équipe placée à gauche du marqueur simultanément dans les pavés gauche et droit du pavé "SET 5", l'équipe de droite dans le pavé du milieu.

2) Comme dans les sets précédents, cocher le "S" et le "R", barrer d'un X la case 1 de l'équipe qui réceptionne et noter l'heure de début.



PT7 - 5ème SET *(3b)

RAPPEL : 1) se joue en 15 pts, avec 2 pts d'écart sans limite.
2) il n'y a pas de TM "Technique" au cours du 5ème Set.

✎ Dès que le huitième point est marqué, il y a changement de camp, inscrire dans le rectangle "POINTS AU CHANGEMENT" le nombre de points marqués par l'équipe qui passe de gauche à droite (uniquement).

✎ Reporter les informations (temps morts et changements de joueurs) du pavé gauche au pavé droit. Ne pas reporter les points marqués. La partie droite étant la suite de la partie gauche, continuer la marque dans cette section.

❖ Lors du changement de camp, mentionner le nombre de points marqués dans le pavé "POINTS AU CHANGEMENT", et dans la case du serveur (à gauche) s'il a fini de servir.

PT8 - Fin de SET *(3a 5)

- A la fin du set inscrire dans la case "FIN", l'heure de fin de set (en heures et minutes),
- cercler le score de chaque équipe dans la dernière case "TOURS AU SERVICE" utilisée, SAUF :
- Si le dernier point est marqué sur service adverse, il est inscrit et cerclé dans la case du joueur qui aurait du prendre le service, sans cocher le numéro de cette case.
- Souligner le score de fin de set de chaque équipe dans la colonne "POINTS" et annuler les points suivants.

PT8 - Fin de Match *(4)

Le pavé "RESULTATS" est une récapitulation avec totalisation par équipe des éléments suivants :

- T : nombre de Temps- morts
- R : nombre de Remplacements
- G :1 pour set gagné : 0 pour set perdu
- POINTS = score de chaque Set
- Durée : durée en minutes

- Effectuer les totaux
- Porter les informations sur la durée du match
- Dans la case "VAINQUEUR" :
  - porter le nom de l'équipe gagnante,
  - après le "3" inscrire le nombre de Sets de l'équipe perdante.

RESULTATS								
ÉQUIPE CLAMART (A)					(B) ASNIERES ÉQUIPE			
T	R	G	P	SET (Durée)	P	G	R	T
2	2	0	17	1 (25)	25	1	0	0
1	0	1	25	2 (20)	10	0	4	2
2	0	1	25	3 (28)	13	0	6	2
2	0	0	08	4 (22)	25	1	1	0
1	0	1	15	5 (18)	09	0	2	1
8	2	3	90	Durée totale Set 113 min	82	2	13	5
Heure Début Match 21 h 00 mn				Heure Fin Match 23h 05 mn	Durée totale Match 2 h 05 mn			
VAINQUEUR				CLAMART	3 : 2			

PT8 - Fin de Match *(4)

La feuille totalement remplie, doivent signer dans l'ordre suivant :

- le marqueur,
- les capitaines,
- le deuxième arbitre,
- le premier arbitre, qui doit remettre un exemplaire de la feuille de match (double) à chaque entraîneur.
- l'original est remis ou conservé par l'organisateur

APPROBATION				
Arbitres	Nom	Pays	N° licence	Signature
1 er	LAB Pierre		XXXX	
2 e	VOSDE Pierre		XXXX	
Marqueur	SCRIB Ouillard		XXXX	
Marqueur Assistant				
1 / 3	Juges de lignes	2 / 4		
A	Capitaines	B		

En cas de réclamation (Règle 5.1.2.1), et avec l'autorisation préalable du 1er arbitre, le marqueur écrit ou permet au capitaine d'écrire sur la feuille de match une déclaration sur l'incident contesté. (Règle 25.2.3.2)

Marqueur

REGLE 25

25. 1 Emplacement

Le marqueur remplit ses fonctions assis à la table de marque située du côté opposé et en face du premier arbitre.

25. 2 Responsabilités

Il tient la feuille de match conformément aux Règles, en coopérant avec le second arbitre.
Il utilise un vibreur ou tout autre signal sonore pour donner des signaux aux arbitres sur la base de ses responsabilités.

25. 2. 1 Avant le match et le set, le marqueur :

25.2.1.1 inscrit les données du match et des équipes selon les procédures en vigueur et recueille les signatures des capitaines et des entraineurs ;

25.2.1.2 enregistre les formations de départ de chaque équipe à partir des fiches de position ;

S'il ne reçoit pas à temps les fiches de position, il doit en informer immédiatement le second arbitre.

25.2.1.3 enregistre le nom et le numéro du joueur Libéro.

Les fiches de position ne sont montrées à personne, excepté aux arbitres.

Marqueur "suite et fin" REGLE 25

R 25. 2. 2 Pendant le match, le marqueur :

25.2.2.1 enregistre les points marqués et en surveille la concordance avec le tableau d'affichage ;

25.2.2.2 contrôle l'ordre de rotation au service et signale à l'arbitre toute erreur immédiatement après la frappe de service ;

25.2.2.3 enregistre les temps-morts et les remplacements, en contrôle le nombre et en informe le second arbitre ;

25.2.2.4 signale aux arbitres une demande d'interruption qui ne serait pas fondée ;

25.2.2.5 annonce aux arbitres les fins de set, le début et la fin de chaque temps- mort technique et la marque du 8e point dans le set décisif ;

25.2.2.6 enregistre les sanctions,

25.2.2.7 enregistre tout événement selon les instructions du second arbitre, ex. les remplacements exceptionnels *(Règle 15.7)*, les temps de récupération *(Règle 17.1.2)*, les interruptions prolongées *(Règle 17.3)*, les ingérences extérieures *(Règle 17.2)*, etc...

25. 2. 3 A la fin du match, le marqueur :

25.2.3.1 inscrit le résultat final ;

25.2.3.2. en cas de réclamation, et avec l'autorisation préalable du premier arbitre, écrit ou permet au capitaine d'écrire sur la feuille de match une déclaration sur l'incident contesté ;

25.2.3.3 après avoir signé lui-même la feuille de match, recueille les signatures des capitaines, puis celles des arbitres.

POUR INFORMATION

I.- TENUE DE LA FEUILLE DE MATCH

DIRECTIVES POUR REMPLIR LA FEUILLE DE MATCH DE VOLLEY-BALL

SYSTEME DE LA « MARQUE CONTINUE »

1- AVANT LA RENCONTRE

Vérifier que les lignes ou cases identifiant la rencontre ont été dûment remplies et éventuellement les compléter comme suit, le tout :

EN MAJUSCULES D'IMPRIMERIE :

a) Dans la partie supérieure de la feuille :

1.a.1 Nom de la compétition.
1.a.2 Site (ville) et code du pays.
1.a.3 Salle (nom du stade ou du gymnase).
1.a.4 Poule (ex : 2MA).
1.a.5 Numéro de la rencontre.
1.a.6 Division et catégorie : mettre un X dans la case.

1.a.7 Date : jour, mois, année.
1.a.8 Heure : celle fixée par la CCS ou la CRS.
1.a.9 Nom des équipes (équipe recevante ou première citée par le calendrier officiel si terrain neutre A GAUCHE). Laisser en blanc les cercles A et B qui seront remplis après le tirage au sort (voir 2.3).

b) Dans la partie inférieure droite :

1.b.1 Equipes : dans le même ordre que 1.a.9 ci-dessus, laisser en blanc les cercles A et B.
1.b.2 **EN ORDRE NUMERIQUE CROISSANT** Numéro de maillot de TOUS les joueurs participant à la rencontre. NOM, Prénom, numéros de licences des joueurs de la rencontre. ***(sauf FIVB).***
Pour le **LIBERO** se reporter à 1.b.5.
1.b.3 NOM, Prénom et Numéros de licences des (E) entraîneurs, (EA) entraîneurs-adjoints, (S) soigneurs, (M) médecins ; ceux-ci peuvent présenter un certificat d'accréditation.
1.b.4 Signature des Capitaines (cercler leurs numéros de maillots en 1.b.2, aucun autre repère) qui doivent vérifier 1.b.2 et 1.b.3 de l'équipe adverse et de leur équipe.

1.b.5 Signatures des Entraîneurs qui doivent effectuer les mêmes vérifications pour leur équipe. Ils peuvent éventuellement vous informer de la présence du LIBERO qui devra être noté dans la case prévue à cet effet ***(sauf FIVB).***

ANNULER LES CASES NON UTILISEES en 1.b.2, 1.b.3., 1.b.5. TOUTE RATURE, RECTIFICATION, MODIFICATION DOIT ETRE APPROUVEE PAR L'ARBITRE DANS LE PAVE « REMARQUES ».
Sauf FIVB et CEV

c) Dans la partie « APPROBATION »

1.c.1 Nom, prénom et numéro de licence du premier arbitre. Ligue d'appartenance.
1.c.2 Nom, prénom et numéro de licence du second arbitre. Ligue d'appartenance.
1.c.3 Nom, prénom et numéro de licence du marqueur. Ligue d'appartenance.
1.c.4 Noms des juges de ligne.

2- APRES LE TIRAGE AU SORT

Obtenir les informations suivantes :

2.1 Du premier arbitre : le côté du terrain où chaque équipe débute le jeu ; l'équipe qui sert en premier.
2.2 Du second arbitre : les fiches de position.

Puis,

2.3 Dans la partie intitulée « SET 1 », inscrire le nom des équipes en fonction du côté du terrain (équipe A à gauche du marqueur).
Où elles vont débuter la rencontre. Cocher d'un X le « S » cerclé de l'équipe qui sert et le « R » cerclé de celle qui reçoit. Barrer d'un X la case 1 de l'équipe qui réceptionne.
2.4 Compléter (A ou B) les cercles vides des parties « EQUIPES » (1.a.9 et 1.b.1).
2.5 Dans la partie intitulée « SET 2 », procéder comme en 2.3 en inversant l'ensemble des données.
2.6 Dans la partie intitulée « SET 3 », procéder comme en 2.3.

2.7 Pour la partie intitulée « SET 4 » si elle doit être utilisée, procéder comme en 2.5.
2.8 APRES AVOIR VERIFIE QUE LES NUMEROS DES JOUEURS FIGURANT SUR LA FICHE DE POSITION EXISTENT DANS LA COMPOSITION (1.b.2) DE L'EQUIPE, les enregistrer selon l'ordre déterminé sur la ligne « Formation de départ », et ce, pour chaque set.

Si un LIBERO doit être utilisé au cours du match, son numéro doit figurer sur la 1ère fiche de position. Vous devez l'inscrire dans la case prévue à cet effet. C'est le dernier moment pour le faire. Si pas de LIBERO annuler la case (sauf FIVB).

9.2. Annexe 2 : Les études de cas

Nous présentons, dans cette annexe, les modèles de tâches des études de cas obtenus une fois la conception terminée. Nous présentons ces modèles selon K-MAD tel que le formalisme est présenté dans [Lucquiaud 2005b]. Pour chacun des cas (le *mailer* et le *Mastermind*), nous présentons la **décomposition des tâches**, l'**état du monde** (objets manipulés, utilisateur et événement) et l'**utilisation des objets** (les liens entre les tâches et les objets).

9.2.1. Mailer

9.2.1.1. Décomposition des tâches

La première étape de la conception du modèle de tâches de notre étude de cas, la *gestion des emails*, est la détermination des différentes tâches et des opérateurs d'ordonnancement des sous-tâches. Afin de ne pas surcharger nos figures, nous présenterons le modèle réalisé en plusieurs fractions. L'activité *gestion des emails* est composée de trois activités principales : *Envoyer email*, *Gérer emails reçus* et *gérer carnet d'adresses*. L'organisation des différentes compositions de l'activité est présentée Figure 9.2.1.

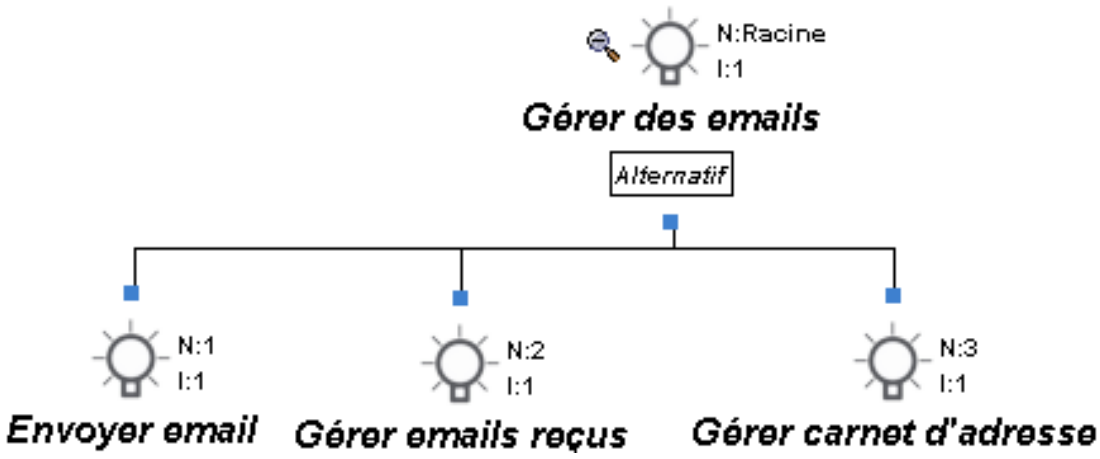


Figure 9.2.1 : Décomposition de haut niveau de l'activité *Gérer les emails*

Envoyer Email. *Envoyer un email* peut suivre deux procédés différents : soit l'email est un nouvel email (*envoyer nouvel email*), soit il est enregistré dans les brouillons (*envoyer email-brouillon*). Dans l'activité que nous avons modélise, ce sont les deux seuls moyens de produire un email, d'autres moyens pourraient être envisagés comme le production d'un email réponse par exemple. Lorsque l'utilisateur choisit d'envoyer un

nouvel email, la première tâche à accomplir est la création d'un nouvel email (*créer email*), puis l'éditer (*éditer email*) et enfin l'envoyer (*envoyer email*).

Chaque tâche doit avoir terminé son exécution pour que la suivante soit exécutable. Donc, elles sont liées par l'opérateur de *séquence*. La tâche *Créer un email* est une tâche élémentaire interactive : l'utilisateur est à l'origine de la création par le système. Au contraire, les tâches *Editer email* et *Envoyer email* sont des tâches composées.

Pour éditer un email, l'utilisateur complète l'en-tête (*Remplir l'en-tête*), éventuellement (noté *OPT* sur la tâche concernée) le message (*Remplir message*) et attache éventuellement un document (*Attacher un document*). Ces tâches peuvent être exécutées de manière concurrente (utilisation de l'opérateur *parallèle*).

Afin de *Remplir l'en-tête*, l'utilisateur doit entrer l'adresse du destinataire (*Remplir destinataire*), éventuellement le sujet (*Remplir sujet*) et les destinataires des copies *Remplir CC* et *Remplir CCC*.

Les tâches *Remplir destinataire*, *Remplir sujet*, *Remplir le message*, *Remplir CC*, *Remplir CCC* et *Attacher un document* sont composées de deux tâches mises en séquence. La première est une tâche utilisateur, dont le but est de déterminer mentalement les destinataires (et/ou de les rechercher, par exemple dans son agenda ou dans les archives des emails), le sujet, le message et le document. La seconde est une tâche interactive qui présente comment l'utilisateur donne les informations déterminées mentalement au système. La tâche *Attacher un document* a pour particularité d'avoir pour tâche interactive non pas *Entrer document* mais *Choisir le document* parmi l'ensemble des fichiers que l'utilisateur possède. La Figure 9.2.2 présente uniquement une de ces tâches décomposées (*Remplir destinataire*), les autres (colorées sur les Figures) respectent le même schéma.

De plus, les tâches *Remplir destinataire*, *Remplir CC*, *Remplir CCC* et *Attacher document* peuvent être réalisées plusieurs fois (afin de permettre la spécification de plusieurs destinataires et d'attacher plusieurs documents).

Envoyer email s'exécute en deux parties : tout d'abord, l'utilisateur lance l'envoi de l'email (*Lancer l'envoi*) et le système réalise alors l'expédition et indique si l'email a été envoyé ou pas (*Envoyer et Afficher*). Ces deux tâches sont élémentaires (non décomposées) et liées par l'opérateur de séquence (la première doit être terminée pour permettre l'exécution de la seconde).

Le modèle de tâches correspondant à la décomposition des tâches pour l'envoi d'un nouvel email est présenté Figure 9.2.2.

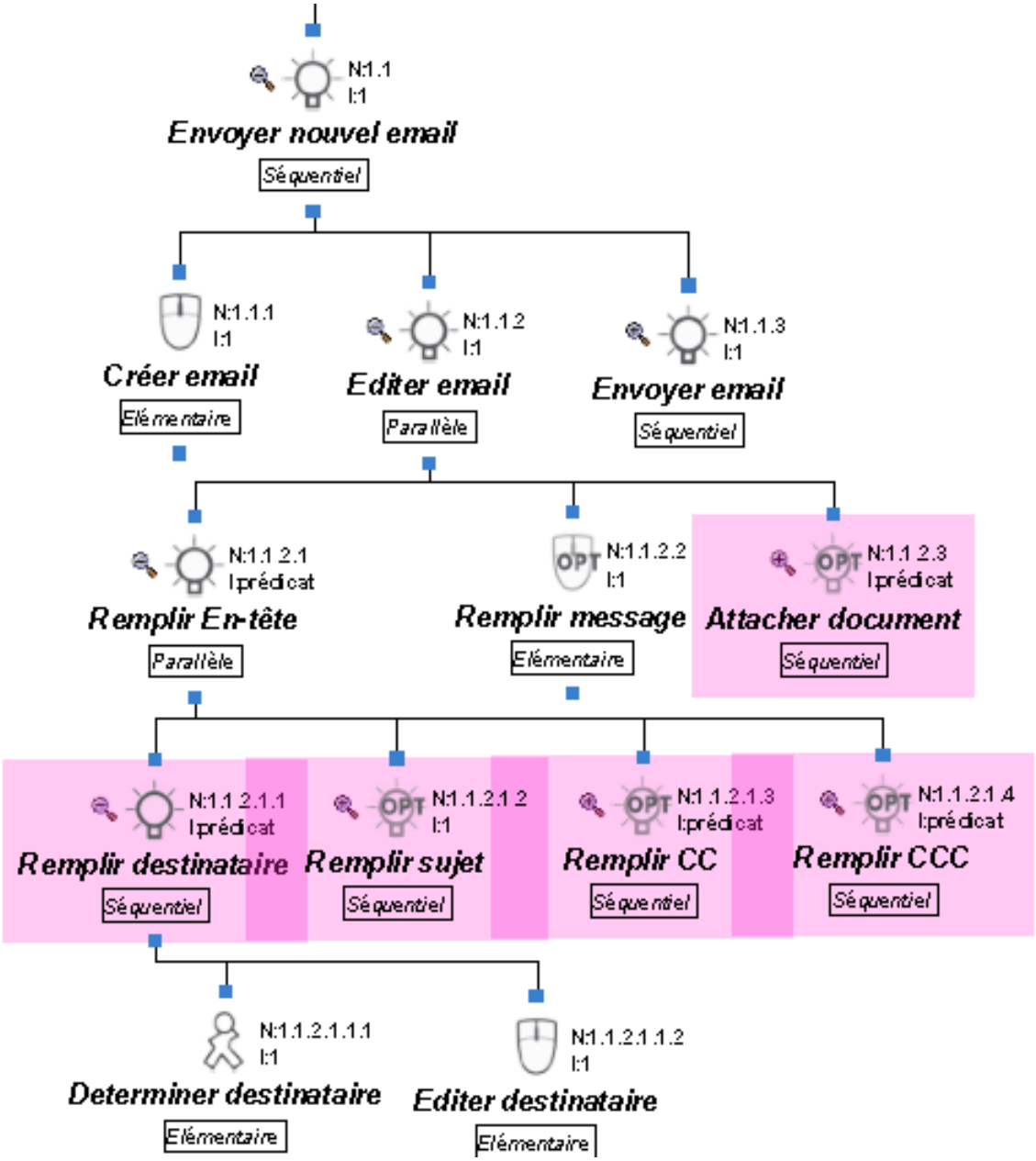


Figure 9.2.2 : Modèle pour l'envoi d'un nouvel email

Dans le cas où l'utilisateur ne crée pas un nouvel email mais reprend un email préalablement enregistré dans les brouillons, la première tâche à réaliser a pour but la sélection d'un email issu des brouillons (*Reprendre email-Brouillon*). Une fois cet email identifié, la tâche à réaliser est la modification (*Modifier email*). Les modifications peuvent être apportées à tous les champs éditables (destinataires, sujet, message, pièce jointe). Enfin, une fois les modifications apportées, l'email peut être envoyé (*Envoyer email*). La Figure 9.2.3 présente la décomposition des tâches pour l'envoi d'un email-brouillon.

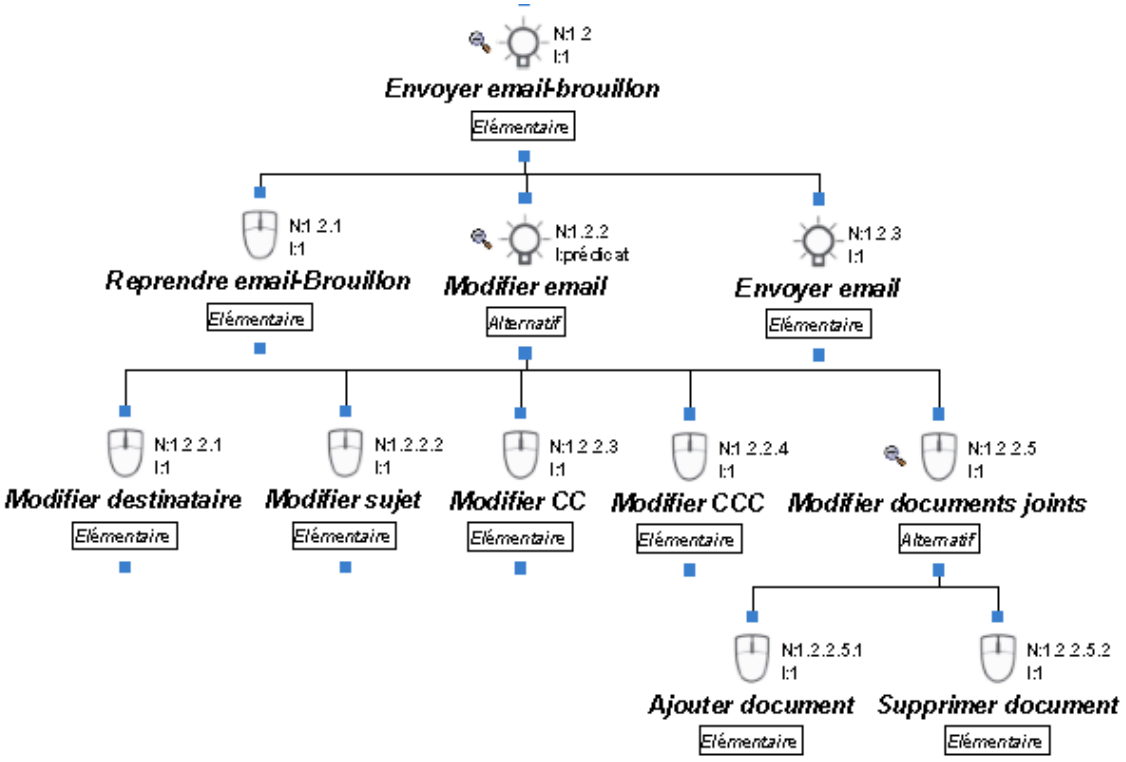


Figure 9.2.3 : Modèle de tâches pour l'envoi d'un email-brouillon

Gérer email reçu. Gérer les emails reçus (Figure 9.2.4) signifie alternativement (*alternatif*) deux tâches. La première est l'information de la réception d'emails par le système (tâche système *indiquer email reçu*). La seconde est le traitement des emails reçus. Pour réaliser cette tâche, trois tâches sont séquentiellement réalisées. Tout d'abord, l'utilisateur choisit interactivement l'email reçu qui va être traité (*choisir un email*), puis l'utilisateur consulte cet email (*lire*) et peut éventuellement (*OPT*) modifier l'emplacement de l'email (*Ranger*).

Choisir l'email qui va être traité (*Choisir email*) est réalisé en deux temps. D'abord, la sélection d'un groupe (i.e d'un dossier) d'emails reçus (*Choisir dossier emails reçus*), puis dans ce dossier d'emails, la sélection de l'email (*Sélectionner email*).

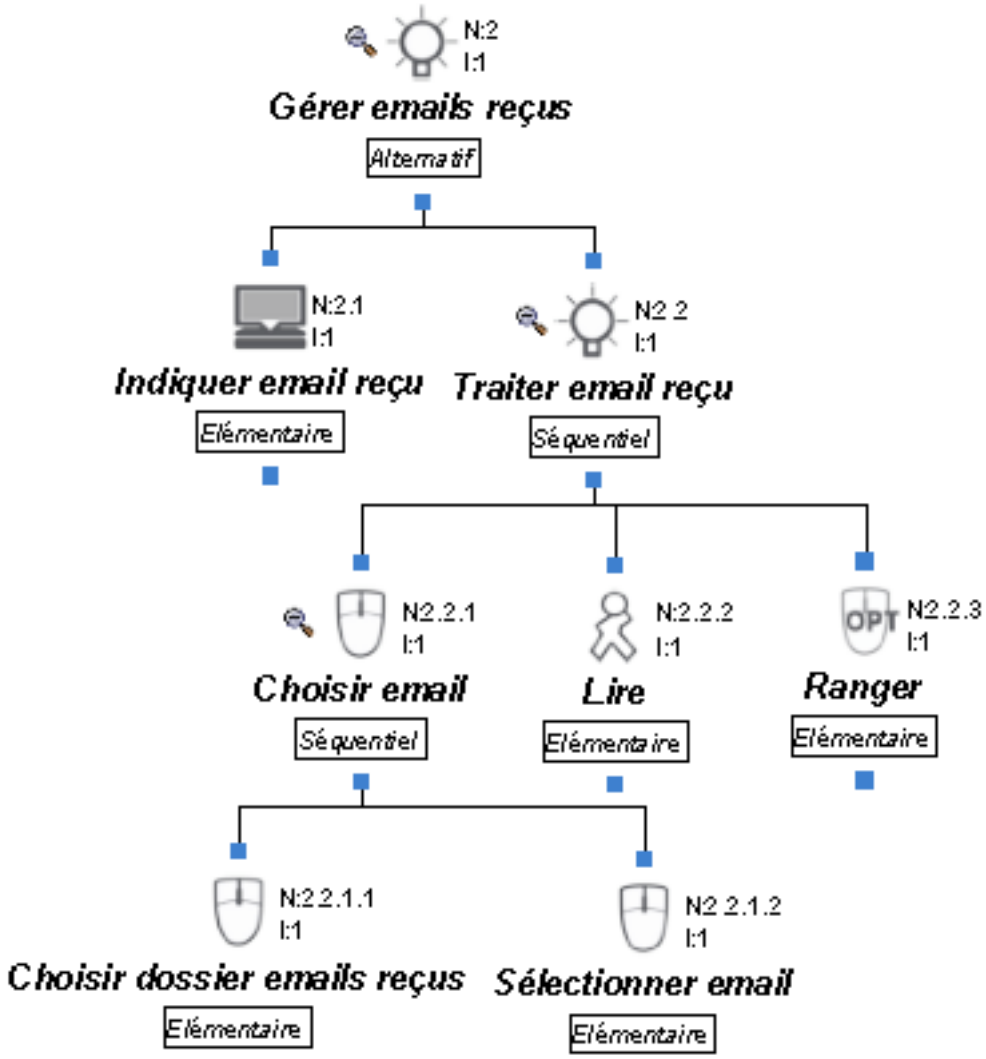


Figure 9.2.4 : Modèle de tâches pour la gestion des emails reçus

Gérer carnet d'adresse. La gestion du carnet d'adresse comprend trois tâches principales : *ajouter un contact*, *supprimer un contact* ou *modifier un contact*. *Ajouter un contact* est réalisée en deux temps : le renseignement des données du contact (*Renseigner nom*, *Renseigner prénom*, *Renseigner adresse*) et l'enregistrement des données (*Enregistrer*). Nous considérons ici, un seul carnet d'adresse.

Pour *Supprimer un contact*, la première tâche à accomplir est la sélection du contact (*Sélectionner email*) puis de le supprimer.

Enfin, pour modifier un contact, l'utilisateur sélectionne le contact (*Sélectionner contact*) et modifie une ou plusieurs des données de ce contact (*Modifier données*). La Figure 9.2.5 présente la décomposition de la tâche *Gérer carnet d'adresse*.

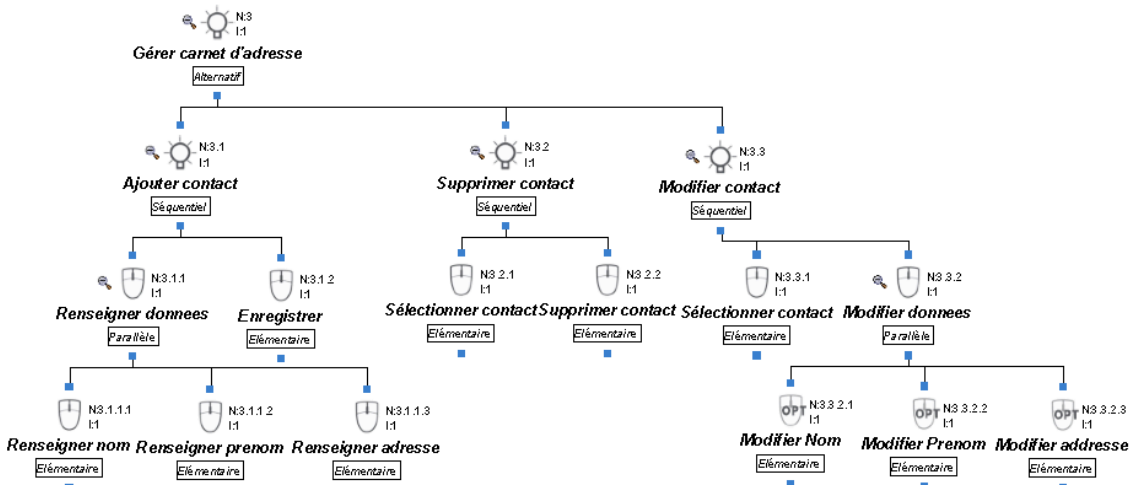


Figure 9.2.5 : Modèle de tâches pour la gestion du carnet d'adresse

9.2.1.2. L'état du monde modélisé

Une fois les tâches et leurs décompositions exprimées, la seconde étape est la définition des objets. Nous allons dans cette partie, présenter l'application des concepts modélisant l'état du monde dans l'activité de notre étude de cas.

Un seul événement est intégré dans cette activité, cet événement est la réception par le système d'un email (*Réception*).

Dans notre étude de cas, le modèle possède trois objets abstraits : les emails, les personnes et les documents. Les attributs de l'email sont les destinataires (principaux, de copie conforme et de copie conforme cachée), le sujet, le message et les documents. Tous ces attributs sont de type *Texte*. Afin de représenter l'email en cours d'édition, nous utilisons un ensemble d'emails : l'ensemble des emails en cours (*EmailEnCours*). En plus de ce groupe modélisant l'état des emails, trois groupes sont définis pour contenir les emails reçus. Le groupe des emails reçus non rangés (*EmailReçuNonTrié*) est le premier groupe dans lequel les emails reçus sont stockés. Les deux derniers groupes (*EmailGroupe1* et *EmailGroupe2*) sont des groupes contenant des emails reçus triés dans les groupes 1 et 2. Enfin, l'ensemble des emails enregistrés dans les brouillons (*Brouillons*) est ajouté.

Les personnes sont identifiées par un nom, un prénom et une adresse (*identifiant@nomDuDomaine*), également de type texte. Un groupe de personnes contenant le destinataire est nécessaire. Un second groupe est défini pour modéliser le carnet d'adresse.

Le dernier objet abstrait est le document. Un document est seulement caractérisé par son nom, que nous avons choisi comme identifiant (nous n'avons pas pris en compte les autres caractéristiques telles que la taille…). Nous définissons deux groupes de documents : un contenant tous les documents de l'utilisateur et un second contenant le

document attaché. Le Tableau 9-1 résume les objets abstraits avec leurs attributs et les groupes.

Objet	Attributs	Groupe	
Email	Destinataire CC CCC Sujet Message Document	EmailEnCours	Groupe des emails en cours sans destinataire
		EmailRecuNonTrie	Groupe des emails reçus et non triés
		EmailGroupe1	Groupe des emails reçus et rangés dans le groupe 1
		EmailGroupe2	Groupe des emails reçus et rangés dans le groupe 2
		Brouillons	Groupe des emails enregistrés dans les brouillons
Personne	Nom Prenom Adresse	PersonneEdition	Groupe de personnes qui sont en édition
		CarnetAdresse	Groupe des personnes enregistrées dans le carnet d'adresses
Document	Nom	Fichiers	Groupe de tous les fichiers de l'utilisateur
		FichierSelectionne	Groupe de fichiers attachés à l'email

Tableau 9-1 : Les objets abstraits, leurs attributs et leurs groupes pour le mailer

9.2.1.3. Utilisation de l'état du monde défini

Nous avons défini un événement, *Reception*, représentant la réception d'un (ou plusieurs) email(s). Cet événement est le seul qui permet le déclenchement de la tâche Système *Indiquer email reçu*. Il est indispensable pour l'exécution de cette tâche, il est donc défini comme étant l'événement déclenchant de la tâche *Indiquer email reçu*.

Les objets de K-MAD sont utilisés pour permettre l'expression de pré-conditions sur l'exécution des tâches, d'itération des tâches et d'expression de manipulation des objets. Ces dernières permettent de simuler les changements d'état des objets dus à l'exécution des tâches. Notre étude de cas (telle que nous l'avons modélisée) comporte six tâches itératives (*Remplir En-tête*, *Remplir destinataire*, *Remplir CC*, *Remplir CCC*, *Attacher document*, *Modifier email*), en plus des pré-conditions et des expressions de manipulation (*post-condition* dans K-MAD).

Les six tâches itératives ont pour condition d'itération les souhaits de l'utilisateur. Cette condition n'est donc pas exprimable à partir de l'état du monde modélisé.

Envoyer email. La création d'un nouvel email ne requiert pas de pré-condition et son exécution ajoute un nouvel objet *email* à l'ensemble des emails en cours mais n'ayant

pas de destinataire (*EmailEnCours*) (`create($EmailEnCours, 'nouveau') ; set($EmailEnCours, $destinataire := 'null')`) avec *nouveau* comme nom de l'objet concret). Afin de compléter un email, il est nécessaire qu'au moins un objet *email* existe dans le groupe *EmailEnCours* et l'utilisateur ne peut envoyer un email que s'il existe au moins un mail prêt à être envoyé (un email dans le groupe des emails en cours dont le destinataire n'est pas 'null'). Une fois l'email envoyé, l'objet correspondant quitte l'ensemble *EmailEnCours*. C'est la conséquence de l'exécution de la tâche *Envoyer et afficher* exprimée par l'expression : `remove($EmailEnCours)`. Il n'y a pas de pré-condition pour définir le destinataire de l'email, le sujet et le message. Le résultat de la tâche *Determiner destinataire* est le déplacement d'une personne du carnet d'adresse dans le groupe des *destinataires* (`replace($CarnetAdresse, $Destinataire)`). La tâche permet de mettre l'adresse de la personne contenue dans le groupe des destinataires dans l'ensemble des adresses de destinataire de l'email en cours (`set($EmailEnCours, $Destinataire := getValue($PersonneEdition, $Adresse))`).

L'ajout des autres destinataires (*Remplir CC* et *Remplir CCC*) suit le même procédé.

Au contraire des autres tâches pour compléter l'email, la tâche *attacher document* respecte une pré-condition : le groupe *Fichiers* doit contenir au moins un fichier. L'exécution des tâches *Remplir sujet*, *Remplir le message* et *Attacher document* complètent respectivement les attributs *sujet*, *message* et *document*.

Les Tableaux 9-2 et 9-3 décrivent chaque type de pré-condition et d'expression de manipulation des objets (formelle et informelle) de ces tâches.

Pour pouvoir envoyer un email-brouillon, la pré-condition est qu'au moins un email soit enregistré comme brouillon. Lorsqu'un email est repris (*Reprendre email-Brouillon*), l'un des emails enregistrés (désigné par l'utilisateur) comme brouillon devient l'email en cours d'édition (`replace($Brouillon,$EmailEnCours)`). Les tâches de modification (*Modifier destinataire*, *Modifier sujet*, *Modifier CC*, *Modifier CCC*, *Modifier documents joints*) suivent le même procédé que dans le cas de l'édition d'un nouvel email.

Tâche	**Pré-condition textuelle**	**Pré-condition formelle**
Editer email	Il existe au moins un email	`card($EmailEnCours)>0`
Envoyer email	Il existe au moins un email avec un destinataire	`card($EmailEnCours, $Destinataire!='null')>0`
Envoyer email-brouillon	Au moins un email est enregistré comme brouillon	`card($Brouillon)>0`

Tableau 9-2 : Description des pré-conditions (formelles et informelles) des tâches pour l'envoi d'email.

Tâche	**Manipulation textuelle des objets**	**Manipulation formelle des objets**
Créer un	Un nouvel email est	`create($EmailEnCours, 'nouveau') ; set($EmailEnCours,$Destinataire :='nu`

email	créé	ll')
Envoyer email	L'email est envoyé	remove($EmailEnCours)
Déterminer destinataire	Une personne du carnet d'adresse devient destinataire désigné	Replace($CarnetAdresse, $Destinataire)
Editer destinataire	Le destinataire de l'email en cours est initialisé avec l'adresse du destinataire auquel l'utilisateur a pensé	set($EmailEnCours, $Destinataire:=getValue($Destinataire ,$Adresse) ; replace($Destinataire, $CarnetAdresse)
Editer sujet	Le sujet est précisé dans l'email par l'utilisateur	set($EmailEnCours,$Sujet := ?STR)
Reprendre email-Brouillon	Changement d'état d'un email-brouillon pour devenir l'email en édition	replace($Brouillon,$EmailEnCours)

Tableau 9-3 : Description des expressions de manipulation des objets (formelles et informelles) des tâches pour l'envoi d'email.

Gérer emails reçus. Dans cette activité, la seule tâche nécessitant l'expression d'une pré-condition est *Traiter email reçu*. Pour pouvoir traiter un email reçu, alors la condition est le fait qu'il y ait au moins un email reçu (trié ou non). Les post-conditions des autres tâches ne sont pas exprimables avec le formalisme exprimé.

Tâche	**Pré-condition textuelle**	**Pré-condition formelle**
Gérer emails reçus	Il y a au moins un email reçu	card($EmailRecuNonTrie) >0 OR card($EmailGroupe1)>0 OR card($EmailGroupe2)>0
Tâche	**Manipulation des objets textuelle**	**Manipulation des objets formelle**
Indiquer email reçu	Un email reçu est ajouté dans l'ensemble des emails non triés	create($EmailRecuNonTrie,'recu')

Tableau 9-4 : Description des pré-conditions et des expressions de manipulation des objets (formelles et informelles) des tâches pour la gestion des emails reçus

Gérer carnet d'adresse. Aucune condition nécessaire (pré-condition) n'est à exprimer pour l'ajout d'un contact. Au contraire, *Supprimer contact* et *Modifier contact* nécessitent qu'il y ait au moins un contact dans le carnet d'adresse. Lorsque la tâche *Créer nouveau contact* est exécutée, alors une personne est ajoutée dans l'ensemble des personnes en édition (`PersonneEdition`). C'est de ce groupe que le contact est édité.

L'édition de chacun des attributs du nouveau contact (*Renseigner Nom*, *Renseigner Prenom* et *Renseigner adresse*) est exprimée par des post-conditions sur l'objet créé. L'exécution de la tâche *Enregistrer* a pour conséquence le déplacement de la personne nouvellement créée du groupe des personnes en édition au groupe des personnes du carnet d'adresse.

Tâche	**Pré-condition textuelle**	**Pré-condition formelle**
Supprimer contact	Il y a au moins un contact dans le carnet d'adresse	`card($CarnetAdresse) >0`
Modifier contact	Il y a au moins un contact dans le carnet d'adresse	`card($CarnetAdresse) >0`
Tâche	**Manipulation des objets textuelle**	**Manipulation des objets formelle**
Créer nouveau contact	Ajoute une personne dans l'ensemble des personnes en cours d'édition	`create($PersonneEdition,'pers')`
Renseigner nom	Donne une valeur à l'attribut *nom* d'un objet du groupe *PersonneEdition*	`set($PersonneEdition, $Nom := ?STR)`
Enregistrer	fait passer le nouveau contact dans le groupe des contacts du carnet d'adresse	`replace($PersonneEdition, $CarnetAdresse)`

Tableau 9-5 : Description des pré-conditions et des expressions de manipulation des objets (formelles et informelles) des tâches pour la gestion du carnet d'adresse

9.2.2. Mastermind

9.2.2.1. Décomposition des tâches

Pour réaliser le modèle de tâche du Mastermind, nous considérons une partie qui oppose l'ordinateur à un joueur. Deux tâches de haut niveau composent alors le jeu. Dans un premier temps, l'ordinateur génère une combinaison sécrète (*générer combinaison secrète*) et ensuite l'utilisateur joue à la trouver (*trouver la combinaison*). Pour cela, il compose une combinaison (*composer une combinaison*) qu'il fait évaluer (*soumettre combinaison*). Pour chaque proposition, l'ordinateur évalue la proposition et présente au joueur le nombre de pions qui sont bien placés dans sa combinaison et le nombre de pions qui sont bien choisis mais mal placés dans la combinaison (*afficher le résultat*). Dans ce

jeu, l'activité de l'adversaire qui cherche à trouver la combinaison secrète est complexe et peut être décrite comme suit : le joueur, détermine d'abord le pion qu'il va choisir, il choisit ce pion parmi les six (6) ensembles de pions (*déterminer un pion*) et le désigne par la suite (*prendre pion*). Une fois le choix opéré il détermine la position dans laquelle le pion désigné sera placé dans la combinaison (*déterminer une position*) et le place (*placer pion*).

Le modèle de tâches K-MAD de ce jeu avec une décomposition jusqu'aux tâches élémentaires du joueur est présenté Figure 9.2.6.

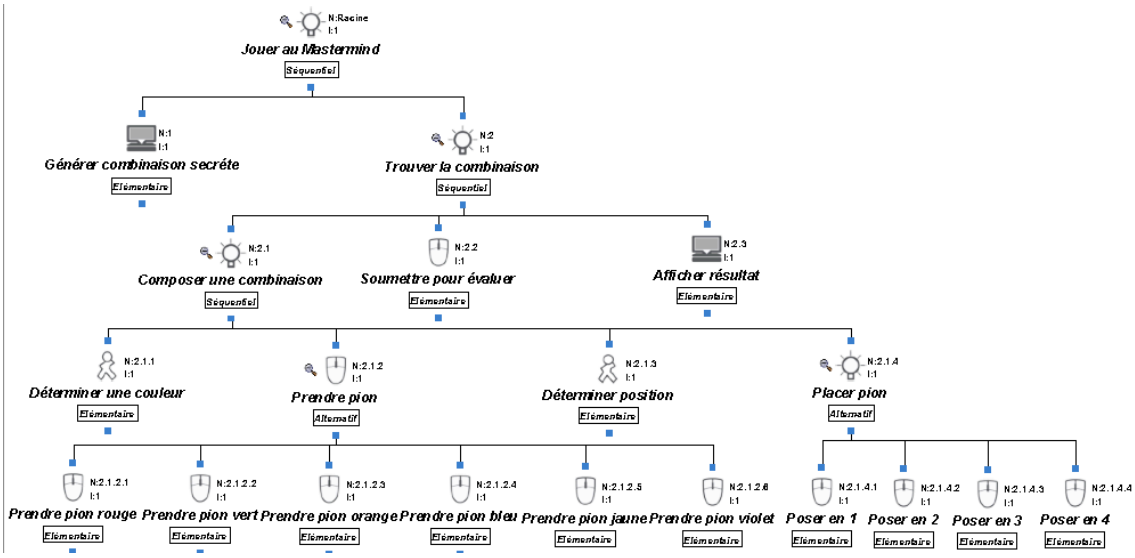


Figure 9.2.6 : Décomposition des tâches du jeu de Mastermind

9.2.2.2. Les objets de l'état du monde

La modélisation que nous proposons du jeu de Mastermind ne fait intervenir qu'un seul utilisateur (le *joueur*) et aucun événement. Deux types d'objets sont manipulés dans cette activité : les pions et les essais.

Au cours d'une partie de *Mastermind*, l'utilisateur manipule les objets *pions* pour composer les combinaisons. Un pion est caractérisé par sa couleur (*Texte*). Plusieurs groupes de pions sont manipulés :

- Le code secret (*CodeSecret*). C'est une liste de quatre (4) pions qui représentent la combinaison générée aléatoirement par le système

- La combinaison en cours de composition (*combinaisonEnCours*). C'est une combinaison de quatre (4) pions réalisée par l'utilisateur. Pour cette combinaison, la position des pions est importante. La tâche de l'utilisateur sera donc d'associer à chacune des positions (une à quatre) un pion qu'il aura choisi.

- Les six ensembles de pions de couleur (*Jaune*, *Violet*, *Rouge*, *Vert*, *Bleu* et *Orange*)

- Le pion en cours de manipulation (*pionChoisi*) : l'utilisateur choisit un seul pion à la fois et le place dans sa combinaison. Ce pion choisi est un objet particulier. Il est désigné et connu. C'est pourquoi nous l'identifions à travers un groupe qui permet de le manipuler

Les essais représentent les combinaisons ayant été évaluées. Leurs attributs sont une représentation d'une combinaison et son évaluation (nombre de pions biens et mal placés). Ces objets, leurs attributs et leurs groupes de leurs objets concrets, sont explicités dans le Tableau 9-6.

Objet	Attribut	Groupe
Pion	Couleur	CodeSecret
		CombinaisonEnCours
		PionChoisi
		Jaune
		Orange
		Rouge
		Vert
		Bleu
		Violet
Essai	Combinaison NbreBienPlace NbreMalPlace	EssaiJoue

Tableau 9-6 : Les objets, attributs et groupes du jeu de Mastermind

9.2.2.3. Utilisation de l'état du monde défini

D'abord nous avons deux tâches de haut niveau *Générer combinaison secrète* et *Trouver combinaison* dont les pré-conditions et les post-conditions sont décrites dans le Tableau 9-7. Pour être exécutée, la tâche *Générer combinaison secrète* doit vérifier en pré-condition que la liste des pions de couleur d'où sont tirés les pions n'est pas vide. Sa post-condition à l'exécution consiste à déplacer un pion généré aléatoirement vers une position définie par l'ordre de génération dans la combinaison de jeu.

La tâche *trouver combinaison* englobe les tâches de composition de combinaison et de leur évaluation. Pour être exécuté, il doit être vérifié en pré-condition que la combinaison secrète a déjà été généré. C'est une tâche itérative qui peut s'exécuter tant que le joueur n'a pas trouvé la combinaison gagnante et que le nombre d'essais autorisés n'est pas expiré.

Tâches	**Pré-condition textuelle**	**Pré-condition formelle**
Trouver la combinaison	la combinaison secrète doit être déjà composée.	`card($combinaisonSecret)==4`
Tâches	**Post-condition textuelle**	**Post-condition formelle**
Générer	La couleur des pions du	`set($CodeSecret,$couleur := ?STR);`

combinaison secrète	code secret est définie	`set($CodeSecret,$couleur := ?STR);` `set($CodeSecret,$couleur := ?STR);` `set($CodeSecret,$couleur := ?STR);`

Tableau 9-7 : Les conditions exprimées pour les tâches de haut niveau du jeu de Mastermind

La tâche *trouver combinaison* se décompose en sous-tâches pour lesquelles des expressions peuvent être décrites. Nous les décrivons dans le Tableau 9-8. *Composer une combinaison* n'a pas de pré-condition, ni de post-condition, mais est itérée 4 fois pour placer les 4 pions d'une combinaison.

Tâche	**Pré-condition textuelle**	**Pré-condition formelle**
Soumettre pour évaluer	La combinaison de jeu a été composée.	`Card($combinaisonDeJeu)==4`
Tâche	**Post-condition textuelle**	**Post-condition formelle**

Soumettre pour évaluer	La combinaison n'est plus en cours d'édition et un résultat de l'évaluation est disponible	`set($essaiJoue, $combinaison := ?STR);`
		`set($essaiJoue, $nbp := ?STR) ;`
		`set($essaiJoue, $nmp := ?STR) ;`

Prendre pion rouge	Le pion choisi est rouge	`replace($Rouge,$PionChoisi)`
Placer pion	Le pion choisi est ajouté à la combinaison en cours d'édition	`replace($pionChoisi, $CombinaisonEnCours)`

Tableau 9-8 : Les conditions exprimées pour *trouver la combinaison*

La tâche *Soumettre pour évaluer* est exécutée suite à la composition de la combinaison de jeu et doit vérifier au préalable (précondition) que la combinaison de jeu à été effectivement composée. Elle permet de déterminer le nombre de pions de la combinaison bien et mal placés.

Lorsqu'un pion est choisi *Prendre pion rouge, Prendre pion bleu*…, le pion devient le pion choisi est change donc de groupe pour passer de l'ensemble des pions disponibles à l'ensemble contenant le pion choisi.

La tâche *Placer pion* a pour post-condition l'ajout du pion choisi dans la combinaison en cours d'édition.

9.3. Annexe 3 : Présentations du formalisme EXPRESS et d'EXPRESS-G

Nous allons, dans cette annexe, décrire sommairement le formalisme de modélisation de données EXPRESS et la représentation graphique EXPRESS-G. C'est ce formalisme que nous avons utilisé pour les modélisations de notre étude.

La présentation que nous allons faire étant une présentation rapide du formalisme, nous conseillons au lecteur voulant plus d'information sur ce sujet de se tourner vers [EXPRESS 1994] ou http://en.wikipedia.org/wiki/EXPRESS_(data_modeling_language) d'où sont issues nos figures.

9.3.1. EXPRESS

EXPRESS est issu d'études menées dans le cadre du projet STEP (STandard for Exchange Product model data). Ce formalisme de modélisation permet l'expression des données et est standardisé par la norme ISO 10303-21 depuis 1994.

EXPRESS est un langage de description de données qui permet de définir des données sur les objets et les relations entre eux.

Les composants principaux de ce langage sont les entités (ENTITY). Ces entités peuvent être vues comme des classes en UML [Object Management Group 1997] sans les méthodes. Les attributs des entités peuvent être des attributs simple (STRING, REAL, BINARY, INTEGER, LOGICAL, BOOLEAN), nommés (définis par une ENUMERATION ou une union de type (SELECT)).

Les entités sont liées les unes aux autres par différentes relations. Ces relations sont les relations d'héritage (simple, multiple ou répétée) ou des relations de composition (différents types d'agrégat sont disponibles : LIST, SET, BAG ou ARRAY).

Les entités peuvent être contraintes, soit par des contraintes locales (clause WHERE) sur une entité particulière, soit par des contraintes globales (clauses UNIQUE, INVERSE et RULE). Pour compléter ces descriptions de données, EXPRESS dispose d'un langage procédural. Ce langage est utilisé pour définir des procédures et des fonctions utilisées dans le modèle (pour définir une valeur calculée par exemple).

Les modèles définis avec le langage EXPRESS sont nommés SCHEMA et se composent comme sur l'exemple de la Figure 9.3.1.

a)

```
SCHEMA Exemple

TYPE T_t=ENUMERATION OF(T1,T2,T3);
END_TYPE;

ENTITY E1;
            a1 : T_t;
            a2 : STRING;
            a3 : INTEGER;
DERIVE
            a4 : INTEGER := f(SELF);
UNIQUE ur1: a3;
END_ENTITY;

FUNCTION f(p: E1) : INTEGER;
RETURN (a3 +3);
END_FUNCTION;

END_SCHEMA;
```

b)

```
SCHEMA Family;

ENTITY Person
   ABSTRACT SUPERTYPE OF (ONEOF (Male, Female));
     name: STRING;
     mother: OPTIONAL Female;
     father: OPTIONAL Male;
END_ENTITY;

ENTITY Female
   SUBTYPE OF (Person);
END_ENTITY;

ENTITY Male
   SUBTYPE of (Person);
END_ENTITY;

END_SCHEMA;
```

Figure 9.3.1 : Structures d'un schéma EXPRESS a) générique et b) d'un schéma *Family*

Les objets d'EXPRESS sont en relation. Trois relations différentes sont définis :

- la relation de composition obligatoire
- la relation de composition facultative
- la relation d'héritage

Des instances conforment à un schéma EXPRESS sont représentées dans un fichier. Chaque instance est identifiée par un OID (#i) et contient une valeur pour chaque attribut de l'entité.

9.3.2. EXPRESS-G

Une représentation graphique du formalisme EXPRESS a été définie : EXPRESS-G. La représentation graphique des principaux composants d'EXPRESS est présentée Figure 9.3.2.

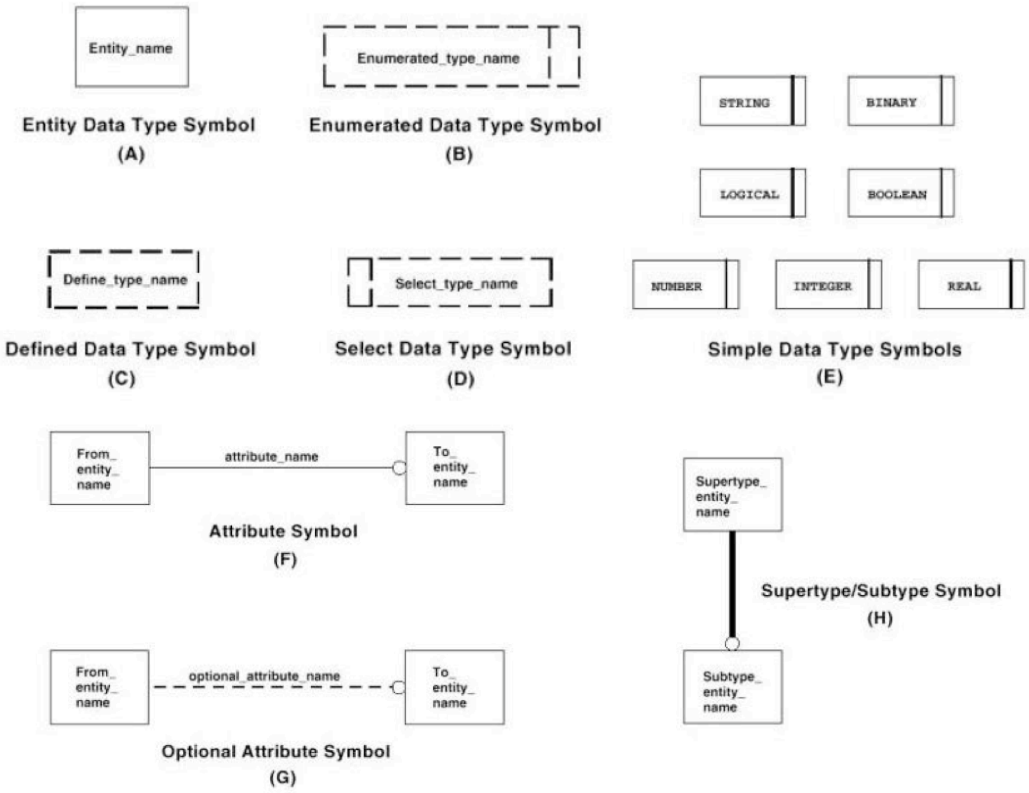


Figure 9.3.2 : Représentation en EXPRESS-G des principaux composants d'EXPRESS

Un exemple de représentation graphique EXPRESS-G du schéma présenté sous sa forme textuel sur la Figure 9.3.1b est présenté Figure 9.3.3.

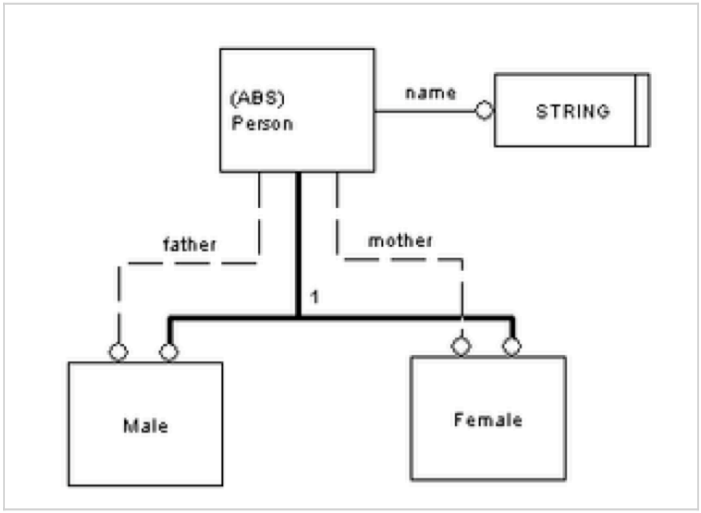
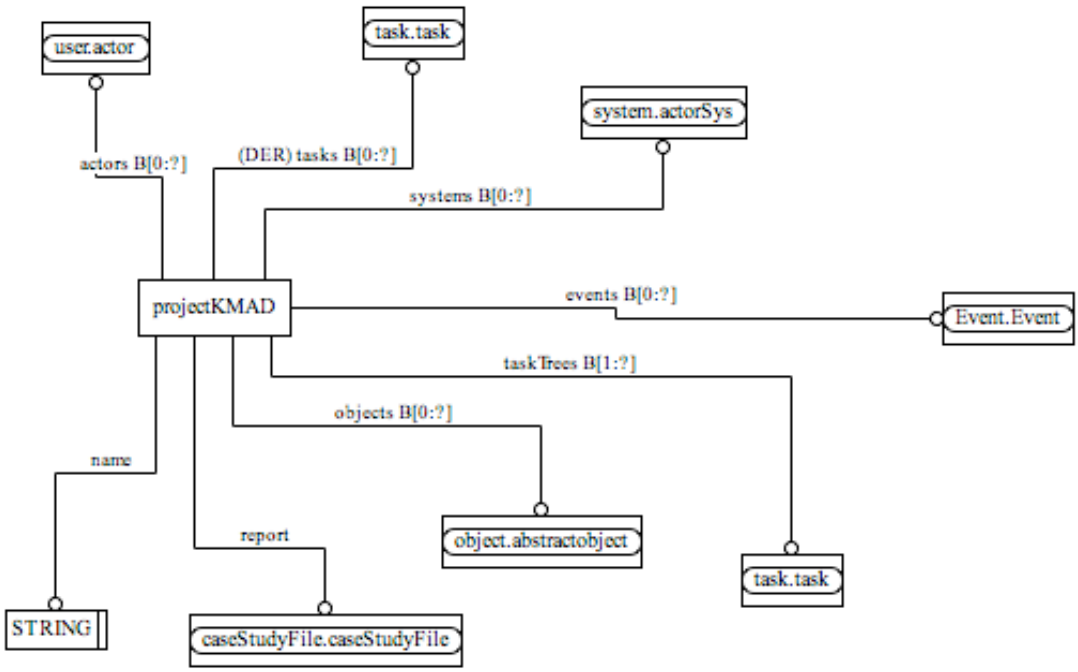


Figure 9.3.3 : Représentation en EXPRESS-G du schéma *Family*

9.4. Annexe 4 : Les méta-modèles de K-MAD(v2)

Pour chacun des schémas de la méta-modélisation de K-MAD(v2), une représentation graphique (EXPRESS-G) et une représentation textuelle du méta-modèle et de ses contraintes sont présentés.

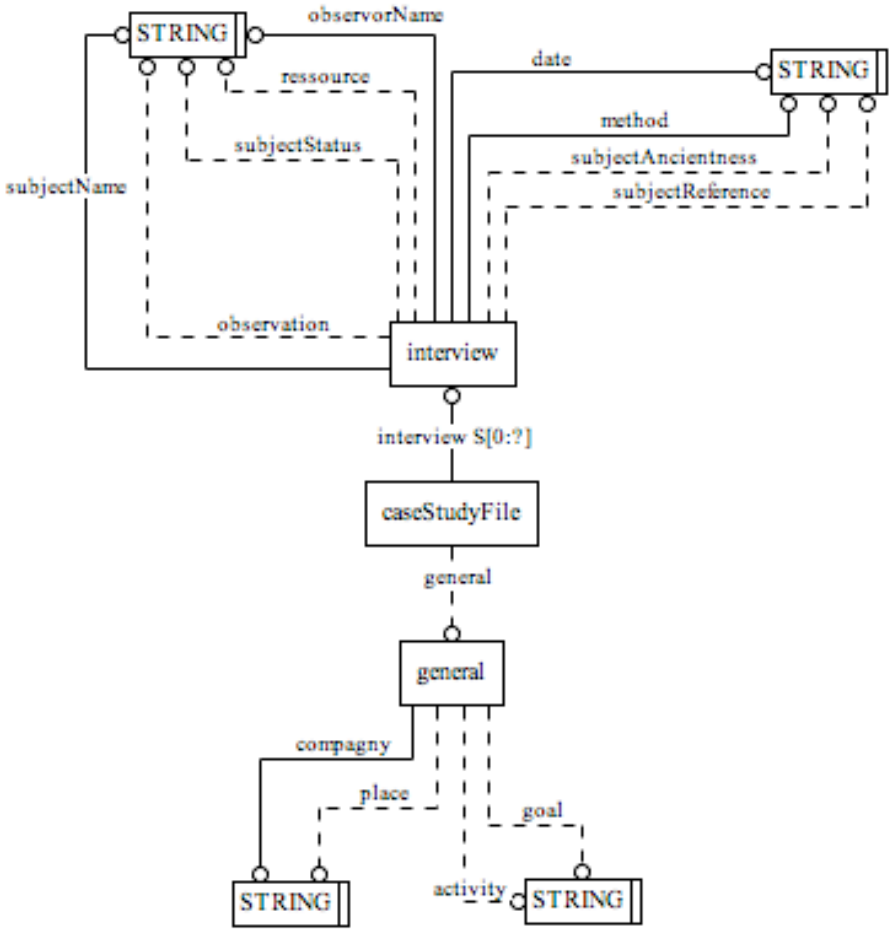
9.4.1. Un projet




```
SCHEMA projectKMAD;
use from caseStudyFile;
use from task;
use from object;
use from Event;
ENTITY projectKMAD;
       name:String;
       report:caseStudyFile;
       taskTrees:BAG[1:?] of task;
       objects:BAG[0:?] of abstractobject;
       events:BAG[0:?] of event;
       actors:BAG[0:?] of actor;
       systems:BAG[0:?] of actorSys;
       DERIVE tasks: BAG[0:?] of task :=taskOfProject(taskTrees);
END_ENTITY;
FUNCTION taskOfProject(trees:Bag of task): BAG of task;
LOCAL
       resTot:BAG of task:=[];
```

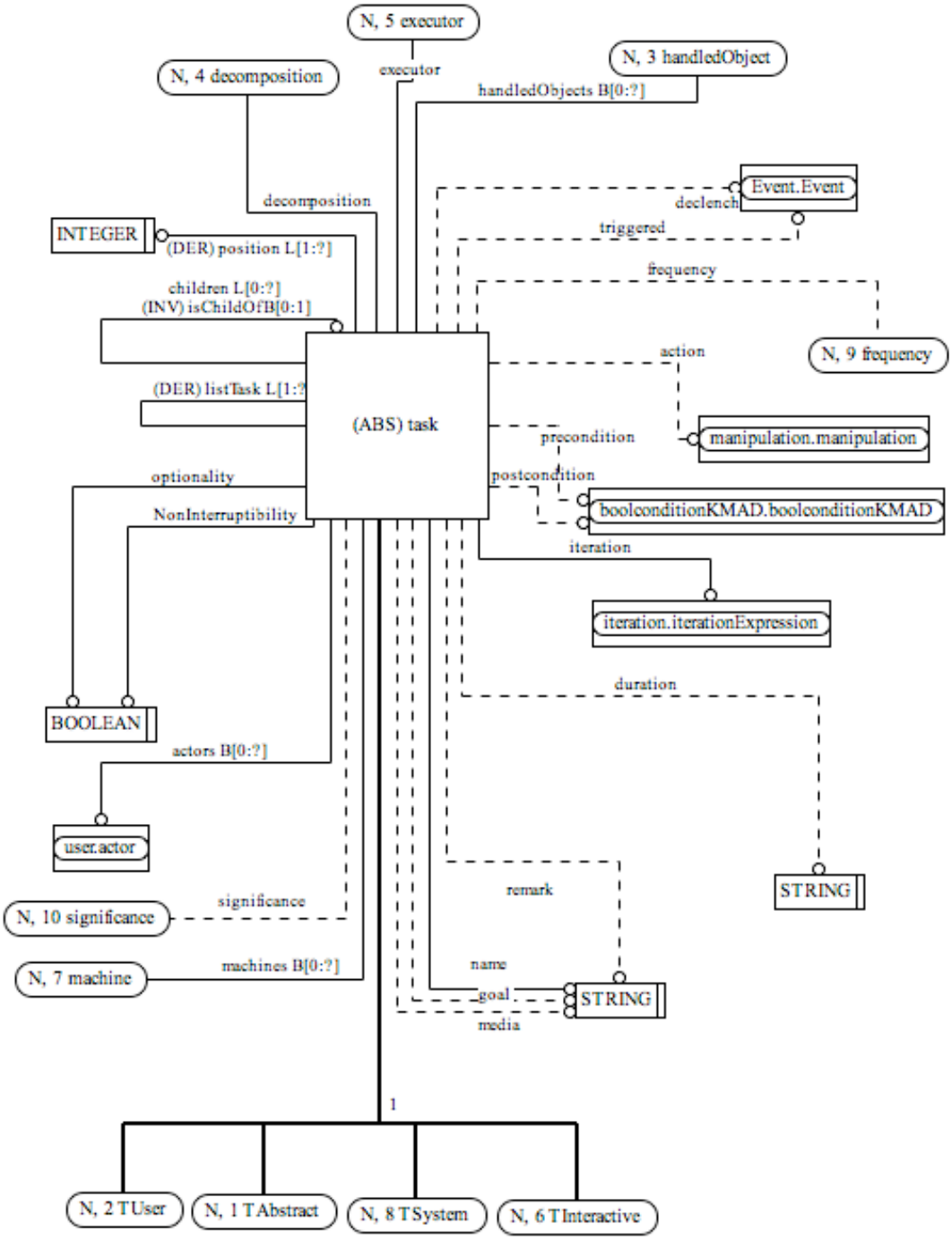
```
        res1:LIST of task;
        it:INTEGER:=0;
        it2:INTEGER:=0;
        it3:INTEGER:=0;
END_LOCAL;
REPEAT it:=LOINDEX(trees) TO HIINDEX(trees);
res1:=trees[it].listTasks;
        REPEAT it2:=LOINDEX(res1) TO HIINDEX(res1);
                it3:=HIINDEX(resTot)+it2+1;
                resTot[it3]:=res1[it2];
        END_REPEAT;
END_REPEAT;
RETURN(resTot);
END_FUNCTION;
END_SCHEMA;
```

9.4.2. Un cas d'étude



```
SCHEMA caseStudyFile;

ENTITY general;
	compagny:String;
	place: OPTIONAL String;
	--description de l'activité de l'entreprise
	activity:OPTIONAL String;

	--justification de l'étude de cas
	goal:OPTIONAL String;
END_ENTITY;

ENTITY interview;
	subjectName:String;
	subjectStatus:OPTIONAL String;
	subjectReference:OPTIONAL String;
	--ancienneté du sujet
	subjectAncientness:OPTIONAL String;
	--à propos du recueil lui meme
	date:String;
	method:String;
	observorName:String;
	--informations complementaires
	observation:OPTIONAL String;
	ressource:OPTIONAL String;
END_ENTITY;

ENTITY caseStudyFile;
	general:OPTIONAL general;
	interview:SET [0:?] OF interview;
END_ENTITY;

END_SCHEMA;
```

9.4.3. Une tâche



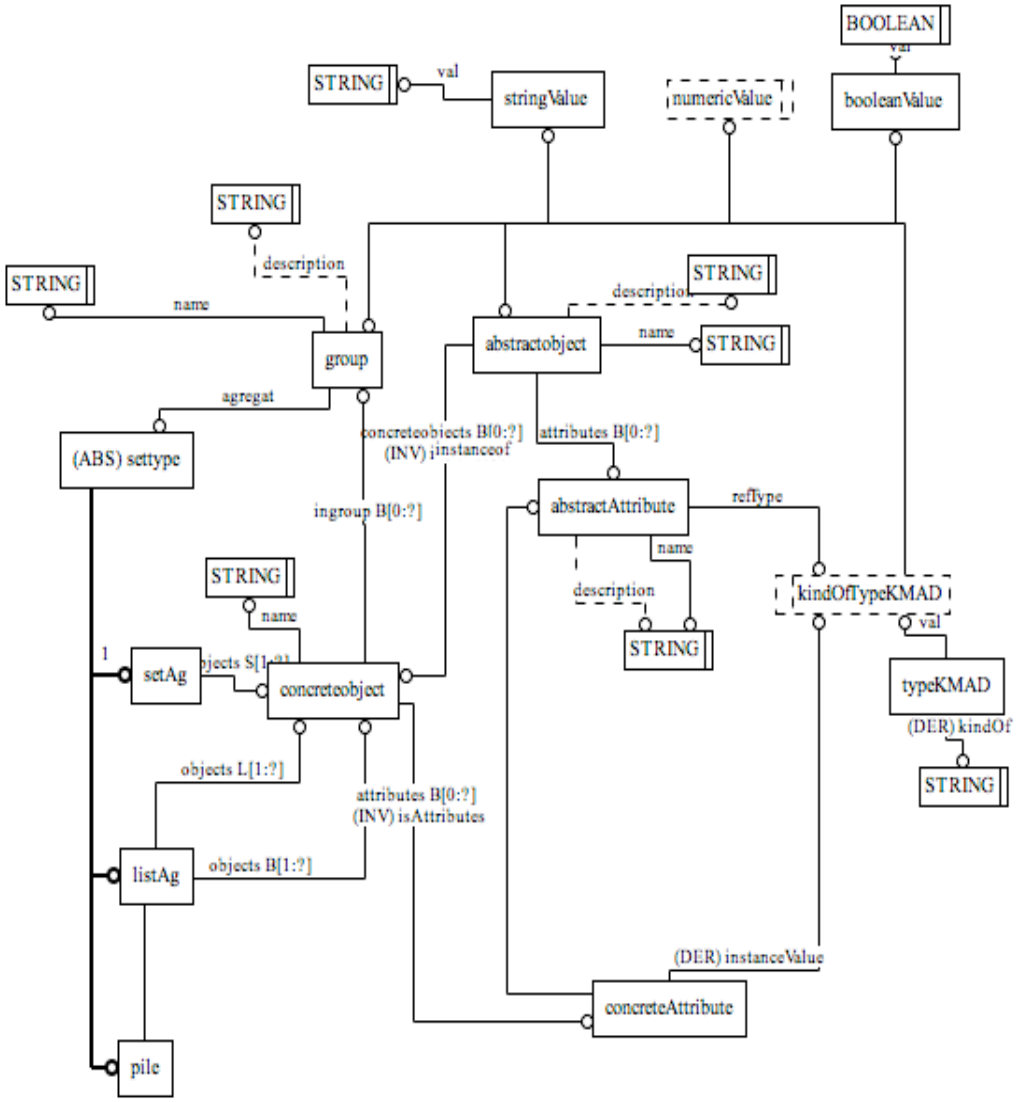

```
SCHEMA task;

use from Event;
use from iteration;
use from boolconditionKMAD;
use from manipulation;
use from user;
use from system;
```

```
--les types d'executants
TYPE executor = ENUMERATION OF (Tuser, Tsystem, Tinteractive, Tabstract);
END_TYPE;

--niveau de frequences
TYPE frequency = ENUMERATION OF (high, medium,low);
END_TYPE;

--niveau d'importance de la tache
TYPE significance = ENUMERATION OF (high, medium,low);
END_TYPE;

--modalité
TYPE modal = ENUMERATION OF (cognitive, sensoriMotrice);
END_TYPE;

--ordonnancement (operateurs de décomposition)
TYPE decomposition=ENUMERATION OF (choice, concurrent, sequence, noOrder, elementary);
END_TYPE;

TYPE source=ENUMERATION OF(systemSource, userSource, interactiveSource, unknownSource);
END_TYPE;

ENTITY handledObject;
       objectType:abstractObject;
       source:source;
END_ENTITY;

ENTITY machine;
       system:actorSys;
END_ENTITY;

ENTITY task
       Abstract supertype of (oneof(TAbstract, TUser, TInteractive, TSystem));
       name:String;
       remark:OPTIONAL String;
       goal:OPTIONAL String;
       media:OPTIONAL String;
       significance: OPTIONAL significance;
       frequency: OPTIONAL frequency;
       duration: OPTIONAL String;
       optionality:boolean;
       NonInterruptibility:boolean;
       actors:BAG[0:?] OF actor;
       machines:BAG[0:?] OF machine;
       iteration: iterationExpression;
       precondition: OPTIONAL boolconditionKMAD;
       postcondition: OPTIONAL boolconditionKMAD;
       action: OPTIONAL manipulation;
       triggered:OPTIONAL event;
       declenchant:OPTIONAL event;
       decomposition: decomposition;
       children:LIST[0:?] OF task;
       handledObjects:BAG[0:?] of handledObject;
       executor:executor;
DERIVE position: LIST[1:?] of integer:=indice(SELF);
       listTask: LIST[1:?] of task:=arbre(SELF);
INVERSE isChildOf:BAG[0:1]of task for children;
END_ENTITY;


--tous les executor sont des attributs DERIVE
```

```
ENTITY TAbstract
       subtype of (task);
       WHERE wra: (SELF.executor=Tabstract);
END_ENTITY;

ENTITY TInteractive
       subtype of (task);
       feedback:OPTIONAL String;
       modal:modal;
       WHERE wri: (SELF.executor=Tinteractive);
END_ENTITY;

ENTITY TUser
       subtype of (task);
       modal:modal;
       WHERE wru: (SELF.executor=Tuser);
END_ENTITY;

ENTITY TSystem
       subtype of (task);
       feedback:OPTIONAL String;
       WHERE wrs: (SELF.executor=Tsystem);
END_ENTITY;

FUNCTION indice ( t : task) : LIST of integer ;
LOCAL
       i:INTEGER;
       filles:LIST of task;
END_LOCAL;
       if(t.isChildOf<>[])then
              return([1]);
       else
              i:=place(t, t.isChildOf);
              return(concat(indice(t.isChildOf),i));
       END_IF;
END_FUNCTION ;

FUNCTION place(m:task; f:LIST of task):integer;
LOCAL
       it:INTEGER:=0;
END_LOCAL;
       REPEAT it:=LOINDEX(f) TO HIINDEX(f);
              IF f[it]=m THEN
              return (it);
              END_IF;
       END_REPEAT;
END_FUNCTION;

FUNCTION concat (linit: LIST of integer; p:integer):LIST of integer;
LOCAL
       res:LIST of integer;
       it:INTEGER:=0;
END_LOCAL;
 REPEAT it:=LOINDEX(linit) TO HIINDEX(linit);
 res[it]:=linit[it];
 END_REPEAT;
 it:=it+1;
 res[it]:=p;
 return (res);
END_FUNCTION;
```

```
--recupere la liste des taches du sous-arbre de racine t
FUNCTION arbre(t:task):LIST of task;
LOCAL
       it:INTEGER:=0;
       it2:INTEGER:=0;
       num:INTEGER;
       res:LIST of task:=[];
       int:LIST of task;
END_LOCAl;
   IF (t.children=[]) THEN
     RETURN([t]);
   ELSE
     REPEAT it:=LOINDEX(t.children) TO HIINDEX(t.children);
       int:=t.children[it].listTask;
       num:=HIINDEX(res)+1;
       REPEAT it2:=LOINDEX(int) TO HIINDEX(int);
         res[it2+num]:=int[it2];
       END_REPEAT;
     END_REPEAT;
     RETURN(res);
   END_IF;
END_FUNCTION;

END_SCHEMA;
```

9.4.4. Un objet

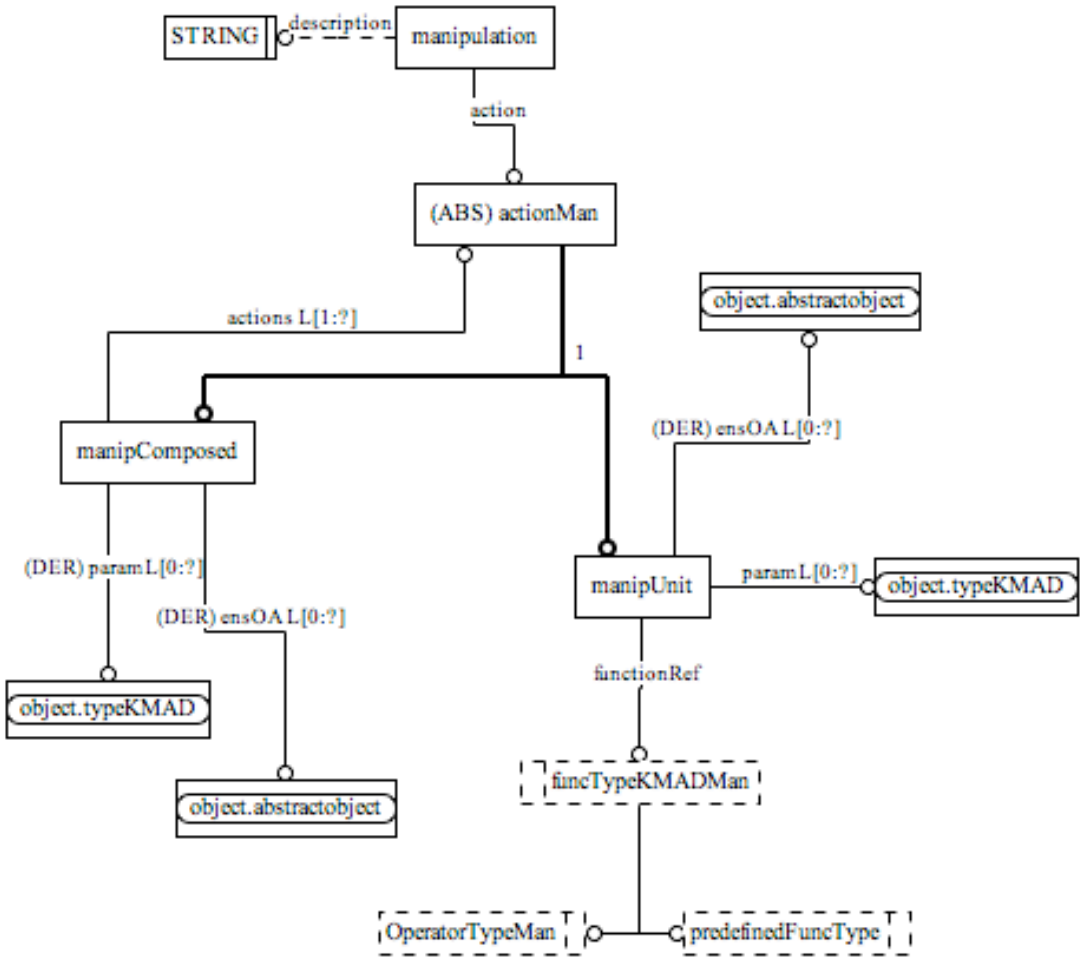



```
SCHEMA object;

TYPE numericValue = ENUMERATION OF (intValue,realValue);
END_TYPE;
TYPE kindOfTypeKMAD = SELECT (stringValue, numericValue, group, booleanValue,
abstractobject);
END_TYPE;
ENTITY typeKMAD;
       val:kindOfTypeKMAD;
       DERIVE kindOf:string:=kindOfName(SELF.val);
END_ENTITY;
ENTITY intValue;
       val:integer;
END_ENTITY;
ENTITY booleanValue;
       val:boolean;
END_ENTITY;
ENTITY realValue;
       val:real;
END_ENTITY;
```

```
ENTITY stringValue;
       val:string;
END_ENTITY;

ENTITY abstractAttribute;
       name:string;
       description:OPTIONAL string;
       refType:kindOfTypeKMAD;
END_ENTITY;

ENTITY abstractobject;
       name:string;
       description:OPTIONAL string;
       attributes:BAG[0:?] of abstractAttribute;
       concreteobjects:BAG[0:?] of concreteobject;
UNIQUE name;
END_ENTITY;

ENTITY concreteAttribute;
       instanceof:abstractAttribute;
DERIVE instanceValue:KindOfTypeKMAD:=instanceof.kindOfTypeKMAD;
INVERSE isAttributes:concreteobject for attributes;
END_ENTITY;

ENTITY concreteobject;
       name:string;
       attributes: BAG[0:?]of concreteAttribute;
       ingroup:BAG[0:?] of group;
INVERSE instanceof: abstractobject for concreteobjects;
END_ENTITY;

ENTITY settype ABSTRACT supertype of (oneof(setAg, listAg, pile));
END_ENTITY;

ENTITY setAg
       SUBTYPE OF (settype);
       objects: SET[1:?] of concreteobject;
END_ENTITY;

ENTITY listAg
       SUBTYPE OF (settype);
       objects: LIST[1:?] of concreteobject;
END_ENTITY;

ENTITY pile
       SUBTYPE OF (settype);
       objects: BAG[1:?] of concreteobject;
END_ENTITY;

ENTITY group;
       name:string;
       description:OPTIONAL string;
       agregat: settype;
UNIQUE
       name;
END_ENTITY;
```

```
FUNCTION kindOfName(varKMAD:kindOftypeKMAD):string;
	if(varKMAD=stringValue) then
	return ('string');
	else
		if(varKMAD=booleanValue) then
		return ('boolean');
		else
			if(varKMAD=abstractobject) then
			return ('abstractobject');
			END_IF;
		END_IF;
	END_IF;
END_FUNCTION;

END_SCHEMA;
```

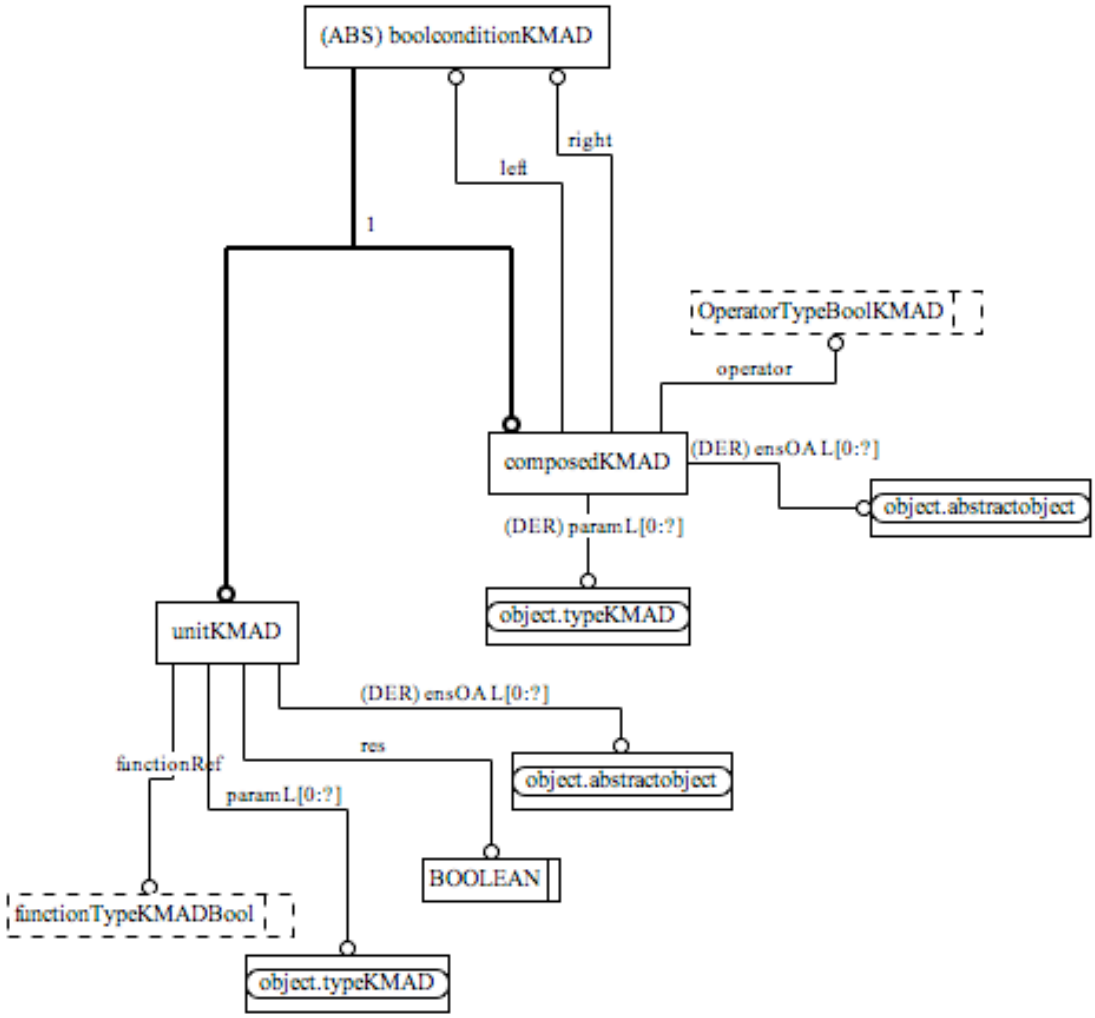
9.4.5. Une expression de manipulation



```
SCHEMA manipulation;
use from object;

TYPE predefinedFuncType=ENUMERATION OF (fadd, fcreate, fgetValue, fremove, freplace,
fset);
END_TYPE;

TYPE OperatorTypeMan = ENUMERATION OF(plus, moins, mult, divis,sup, inf, seq, plusEgal,
moinsEgal, affect, ET, OU, NON);
END_TYPE;

TYPE funcTypeKMADMan= SELECT (predefinedFuncType, OperatorTypeMan);
END_TYPE;

ENTITY manipulation;
       description:optional string;
       action:actionMan;
END_ENTITY;

ENTITY actionMan abstract supertype of (oneof(manipUnit,manipComposed));
END_ENTITY;

ENTITY manipUnit
subtype of(actionMan);
       functionRef:funcTypeKMADMan;
       param:list[0:?] of typeKMAD;
       DERIVE ensOA:LIST[0:?] of abstractobject:=selectOA(SELF.param);
END_ENTITY;

ENTITY manipComposed subtype of(actionMan);
       actions:LIST[1:?] of actionMan;
       DERIVE param:LIST[0:?] of typeKMAD:=typeKMADOf(SELF);
              ensOA:LIST[0:?] of abstractobject:=varOA(SELF.actions);
END_ENTITY;

FUNCTION typeKMADOf(man:manipComposed):LIST of typeKMAD;
LOCAL
       res:LIST[0:?] of typeKMAD:=[];
END_LOCAL;
       REPEAT i:=LOINDEX(man.actions) TO HIINDEX(man.actions);
              res:=res+man.actions[i].param;
       END_REPEAT;
       return (res);
END_FUNCTION;

FUNCTION varOA(p:LIST of actionMan):LIST of abstractobject;
LOCAL
       res:LIST[0:?] of abstractobject:=[];
END_LOCAL;
       REPEAT i:=LOINDEX(p) TO HIINDEX(p);
              res:=res+p[i].ensOA;
       END_REPEAT;
       return (res);
END_FUNCTION;

FUNCTION selectOA(p:LIST of typeKMAD):LIST of abstractobject;
LOCAL
       res:LIST[0:?] of abstractobject:=[];
END_LOCAL;
       REPEAT i:=LOINDEX(p) TO HIINDEX(p);
              if(p[i].kindOf='abstractobject') then
```

```
            res:=res+p[i].val;
            END_IF;
      END_REPEAT;
      return (res);
END_FUNCTION;
END_SCHEMA;
```

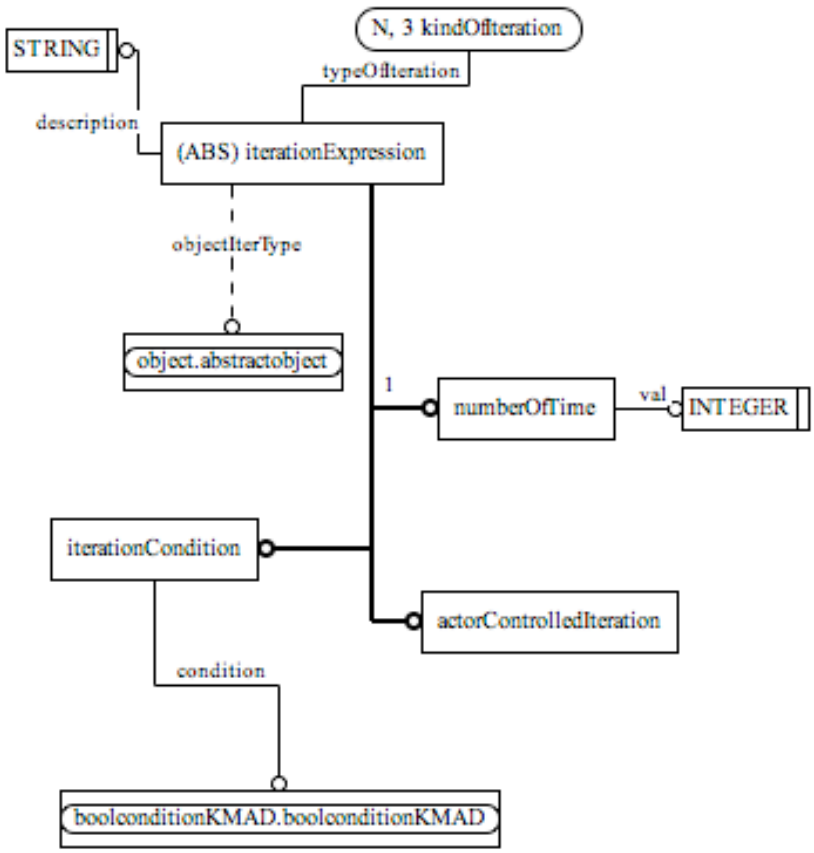
9.4.6. Une expression booléenne




```
SCHEMA boolconditionKMAD;
use from object;

TYPE OperatorTypeBoolKMAD = ENUMERATION OF(plus, moins, mult, divis, sup, inf, egal,
diff, plusEgal, moinsEgal, affect, ET, OU, NON);
END_TYPE;

TYPE functionTypeKMADBool = ENUMERATION OF(card, getValue, isExist, isEmpty, isExistAt);
END_TYPE;
```

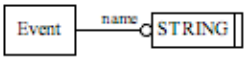
```
ENTITY boolconditionKMAD abstract supertype of (oneof(unitKMAD, composedKMAD));
END_ENTITY;

ENTITY composedKMAD
       subtype of(boolconditionKMAD);
       left:boolconditionKMAD;
       right:boolconditionKMAD;
       operator:OperatorTypeBoolKMAD;
       DERIVE param:LIST[0:?] of typeKMAD:=typeKMADOf(SELF);
              ensOA:LIST[0:?] of abstractobject:=varOA(SELF);
END_ENTITY;

ENTITY unitKMAD
subtype of(boolconditionKMAD);
       functionRef:functionTypeKMADBool;
       res:boolean;
       param:LIST[0:?] of typeKMAD;
       DERIVE ensOA:LIST[0:?] of abstractobject:=selectOA(SELF.param);
END_ENTITY;

FUNCTION typeKMADOf(condComp:composedKMAD):LIST of typeKMAD;
LOCAL
       res:LIST[0:?] of typeKMAD:=[];
END_LOCAL;
       res:=condComp.left.param+condComp.right.param;
       return (res);
END_FUNCTION;

FUNCTION varOA(p:composedKMAD):LIST of abstractobject;
LOCAL
       res:LIST[0:?] of abstractobject:=[];
END_LOCAL;
       res:=p.left.ensOA+p.right.ensOA;
       return (res);
END_FUNCTION;

FUNCTION selectOA(p:LIST of typeKMAD):LIST of abstractobject;
LOCAL
       res:LIST[0:?] of abstractobject:=[];
END_LOCAL;
       REPEAT i:=LOINDEX(p) TO HIINDEX(p);
              if(p[i].kindOf='abstractobject') then
              res:=res+p[i].val;
              END_IF;
       END_REPEAT;
       return (res);
END_FUNCTION;

END_SCHEMA;
```

9.4.7. Une expression d'itération



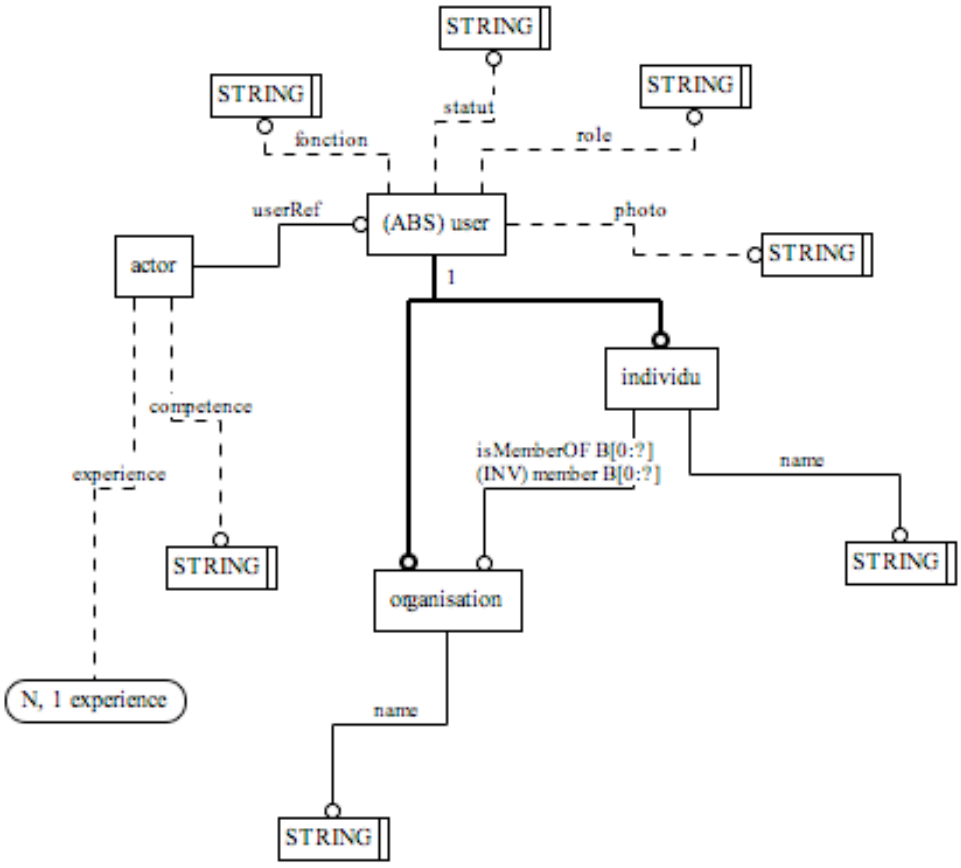

```
SCHEMA iteration;
use from boolconditionKMAD;

TYPE kindOfIteration = ENUMERATION OF (num, iterCond, actorControl);
END_TYPE;

ENTITY iterationExpression abstract supertype of (oneof(numberOfTime,
iterationCondition, actorControlledIteration));
       description:string;
       typeOfIteration:kindOfIteration;
       objectIterType:OPTIONAL abstractobject;
END_ENTITY;

ENTITY numberOfTime subtype of (iterationExpression);
       val:integer;
       WHERE wrn:(SELF.typeOfIteration=num);
END_ENTITY;

ENTITY iterationCondition subtype of (iterationExpression);
       condition:boolconditionKMAD;
       WHERE wric:(SELF.typeOfIteration=iterCond);
END_ENTITY;

ENTITY actorControlledIteration subtype of (iterationExpression);
WHERE wrac:(SELF.typeOfIteration=actorControl);
--cas où l'iteration est faite tant que l'utilisateur veut
END_ENTITY;
END_SCHEMA;
```

9.4.8. Un événement




```
SCHEMA Event;

ENTITY Event;
	name:String;
UNIQUE
	name;
END_ENTITY;
END_SCHEMA;
```

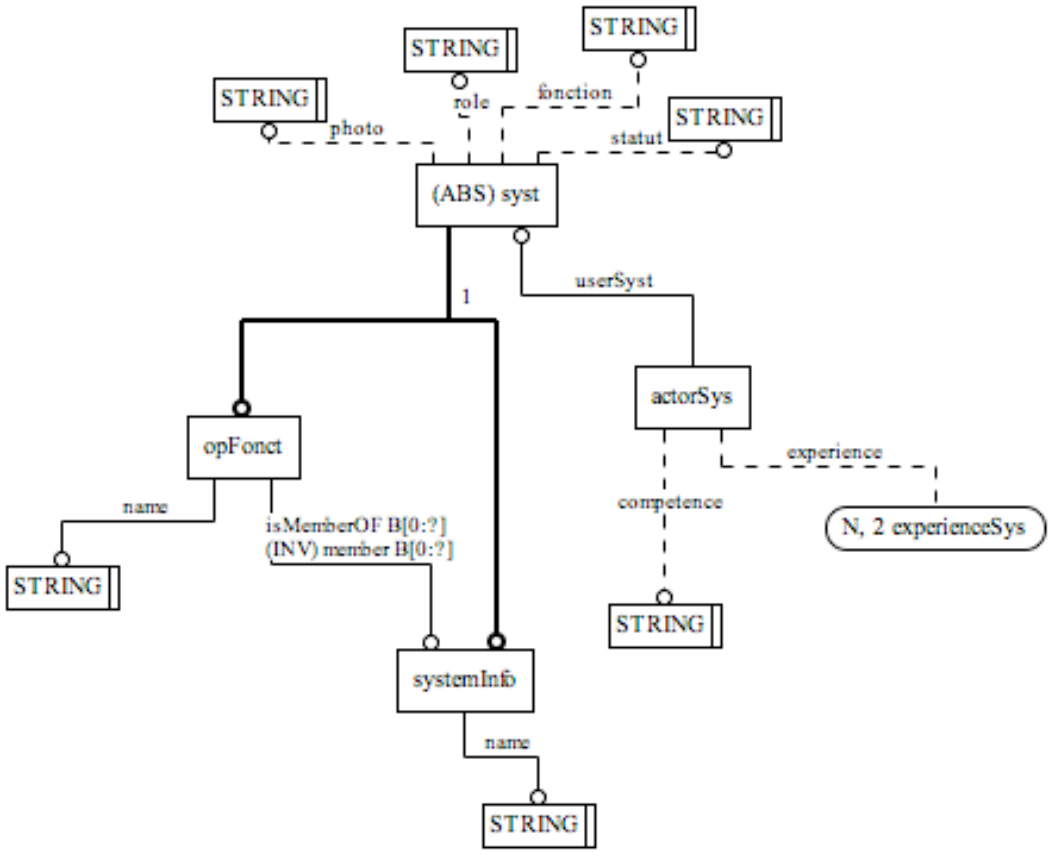
9.4.9. Un utilisateur




```
SCHEMA user;

TYPE experience = ENUMERATION OF (expert, medium, novice);
END_TYPE;
```

```
ENTITY actor;
       experience : OPTIONAL experience;
       competence : OPTIONAL STRING;
       userRef:user;
END_ENTITY;

ENTITY user abstract supertype of(oneof(individu,organisation));
       statut: OPTIONAL STRING;
       fonction:OPTIONAL STRING;
       role:OPTIONAL String;
       photo:OPTIONAL string;
END_ENTITY;

ENTITY individu subtype of (user);
       name:STRING;
       isMemberOF:BAG[0:?] of organisation;
       UNIQUE name;
END_ENTITY;

ENTITY organisation subtype of (user);
       name:string;
INVERSE       member: BAG[0:?] of individu for isMemberOf;
UNIQUE name;
END_ENTITY;

END_SCHEMA;
```

9.4.10. Un système



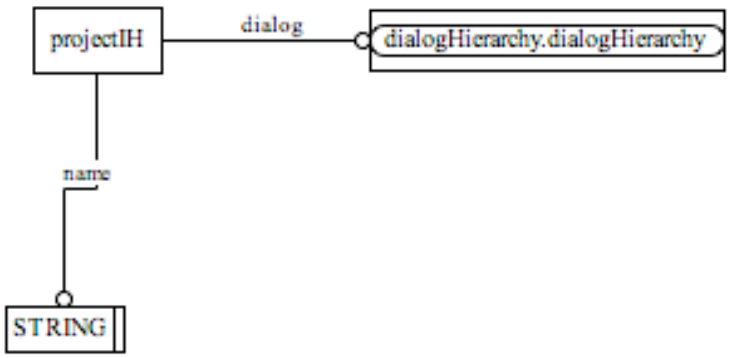

```
SCHEMA system;

TYPE experienceSys = ENUMERATION OF (expert, medium, novice);
END_TYPE;

ENTITY actorSys;
       experience : OPTIONAL experienceSys;
       competence : OPTIONAL STRING;
       userSyst:syst;
END_ENTITY;

ENTITY syst abstract supertype of(oneof(systemInfo,opFonct));
       statut: OPTIONAL STRING;
       fonction:OPTIONAL STRING;
       role:OPTIONAL String;
       photo:OPTIONAL string;
END_ENTITY;

ENTITY opFonct subtype of (syst);
       name:STRING;
       isMemberOF:BAG[0:?] of systemInfo;
       UNIQUE name;
END_ENTITY;

ENTITY systemInfo subtype of (syst);
       name:string;
```

```
INVERSE       member: BAG[0:?] of opFonct for isMemberOf;
UNIQUE name;
END_ENTITY;

END_SCHEMA;
```

9.5. Annexe 5 : La méta-modèlisation des IH

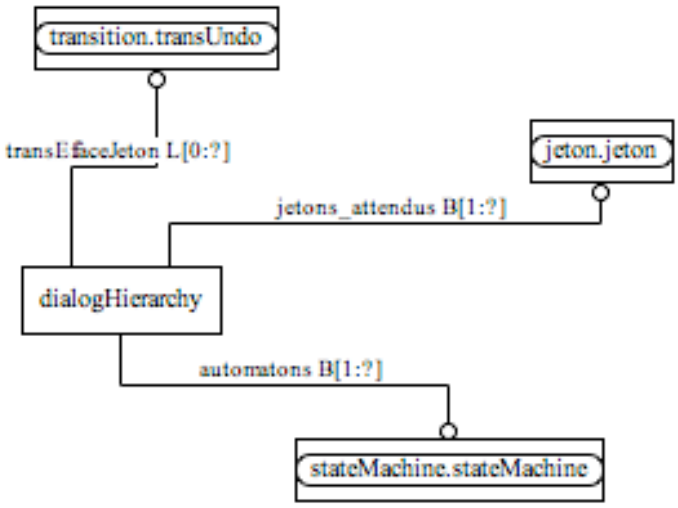
Pour chacun des schémas de la méta-modélisation des IH, une représentation graphique (EXPRESS-G) et une représentation textuelle du méta-modèle et de ses contraintes sont présentés.

9.5.1. Un projet IH



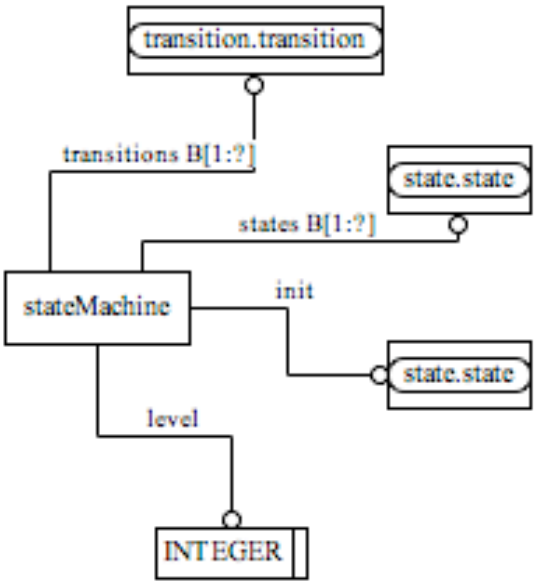

```
SCHEMA projectIH;
use from dialogHierarchy;

ENTITY projectIH;
       name:String;
       dialog:dialogHierarchy ;
END_ENTITY;

END_SCHEMA;
```

9.5.2. Un dialogue hiérarchique



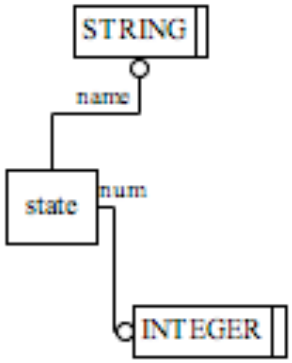

```
SCHEMA dialogHierarchy;

use from stateMachine;
use from jeton;
use from transition;

Entity dialogHierarchy;
       automatons:BAG[1:?] of stateMachine;
       jetons_attendus:BAG[1:?] of jeton;
       transEffaceJeton:LIST[0:?] of transUndo;
       end_entity;
END_SCHEMA;
```

9.5.3. Une machine à états



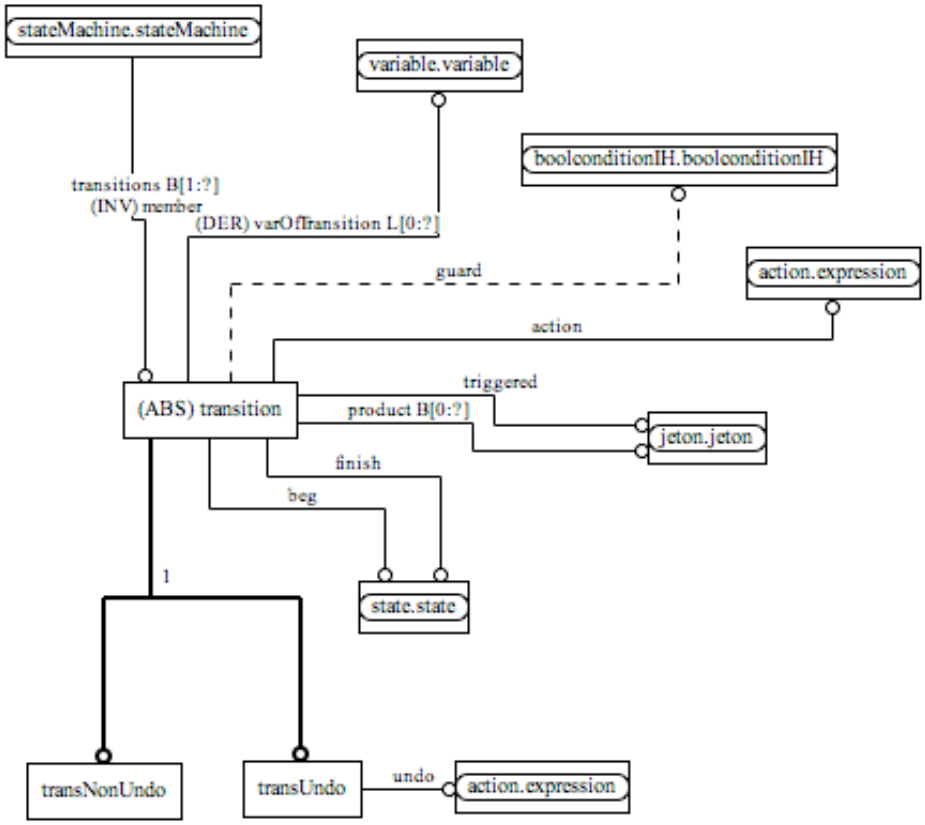

```
SCHEMA stateMachine;

use from state;
use from transition;

ENTITY stateMachine;
       level:Integer;
       init:state;
       states:BAG[1:?] of state;
       transitions:BAG[1:?] of transition;
END_ENTITY;

END_SCHEMA;
```

9.5.4. Un état



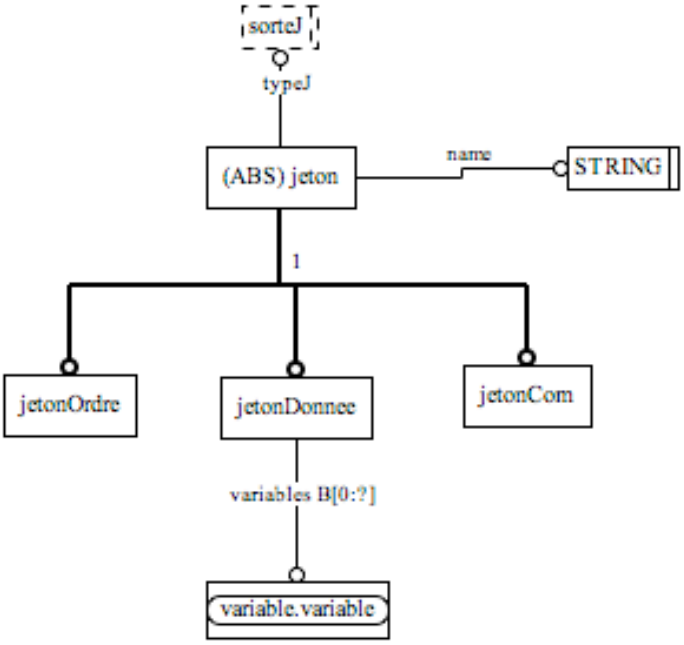

```
SCHEMA state;

ENTITY state;
       num:Integer;
       name:String;

END_ENTITY;

END_SCHEMA;
```

9.5.5. Une transition



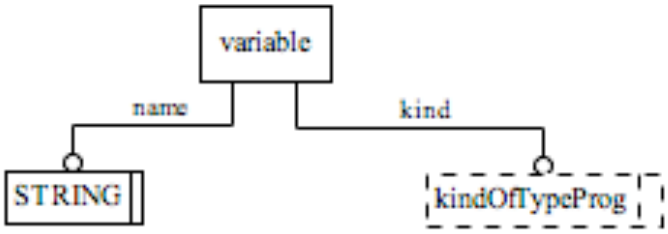

```
SCHEMA transition;
use from state;
use from boolconditionIH;
use from action;
use from variable;
use from jeton;
use from stateMachine;


ENTITY transition abstract supertype of (oneof(transUndo, transNonUndo));
       beg:state;
       finish:state;
       product: BAG[0:?] of jeton;
       triggered:jeton;
       action:expression;
       guard:OPTIONAL boolconditionIH;
       DERIVE varOfTransition: LIST[0:?] of variable:=action.param+guard.param;
       INVERSE member : stateMachine for transitions;
END_ENTITY;

Entity transNonUndo subtype of (transition);
end_entity;

Entity transUndo subtype of (transition);
       undo:expression;
end_entity;

END_SCHEMA;
```

9.5.6. Un jeton



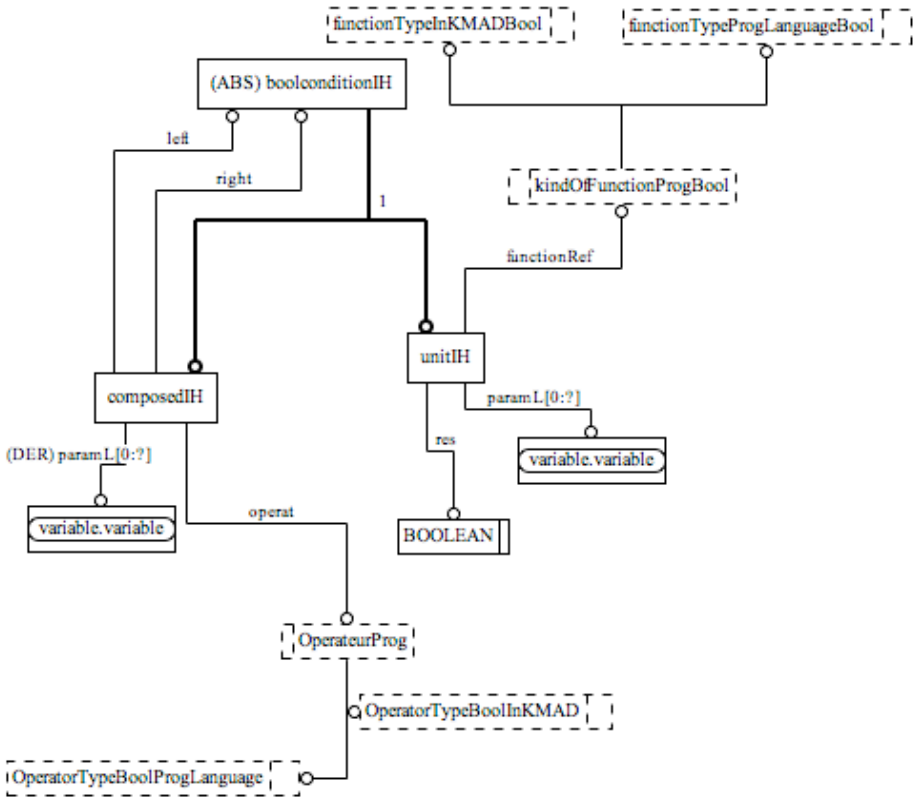

```
SCHEMA jeton;

use from variable;

type sorteJ=ENUMERATION OF(communication, commande, donnee);
end_type;

entity jeton abstract supertype of(oneof(jetonCom, jetonOrdre, jetonDonnee));
       name:String;
       typeJ:sorteJ;
end_entity;

entity jetonCom subtype of(jeton);
where typeJ=communication;
end_entity;

entity jetonOrdre subtype of(jeton);
where typeJ=commande;
end_entity;

entity jetonDonnee subtype of(jeton);
variables: BAG[0:?] of variable;
where typeJ=donnee;
end_entity;
END_SCHEMA;
```

9.5.7. Une variable




```
SCHEMA variable;

--liste non exhaustive qui dépend du language de programmation utilisé
TYPE kindOfTypeProg = ENUMERATION OF(intValueProg, realValueProg, definedTypeValueProg,
StringValueProg);
END_TYPE;

ENTITY intValueProg;
       val:integer;
END_ENTITY;

ENTITY booleanValueProg;
       val:boolean;
END_ENTITY;

ENTITY realValueProg;
       val:real;
END_ENTITY;

ENTITY stringValueProg;
       val:string;
END_ENTITY;

ENTITY variable;
       name:String;
       kind:kindOfTypeProg;
END_ENTITY;
END_SCHEMA;
```

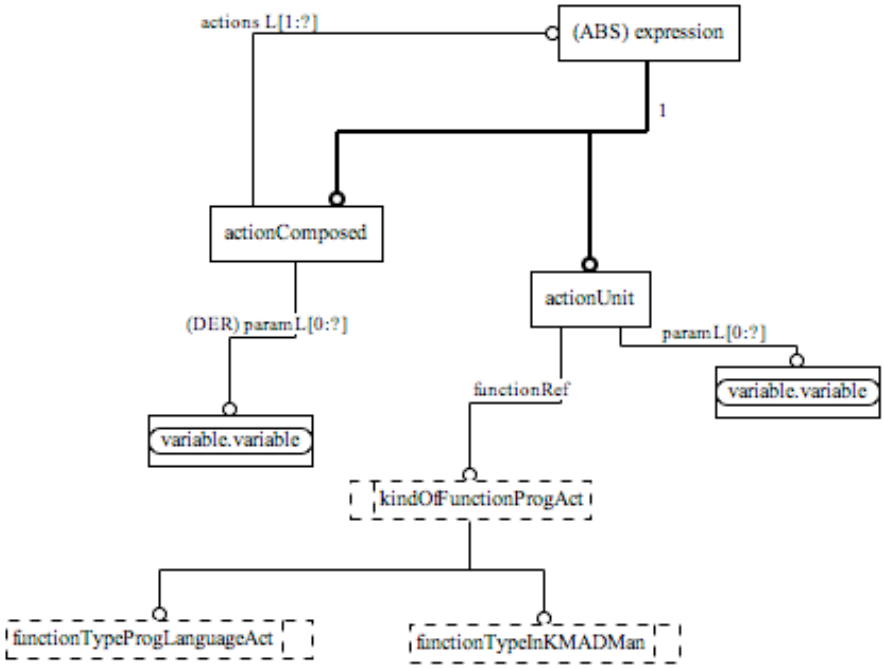
9.5.8. Une expression booléenne




```
SCHEMA boolconditionIH;
use from variable;

TYPE OperatorTypeBoolInKMAD = ENUMERATION OF(plus, moins, mult, divis, sup, inf, egal,
diff, plusEgal, moinsEgal, affect, ET, OU, NON);
END_TYPE;

--liste non exhaustive qui dépend du language de programmation utilisé
TYPE OperatorTypeBoolProgLanguage = ENUMERATION OF(sup, inf, egal, plus, moins,
affect);
END_TYPE;

TYPE OperateurProg = SELECT(OperatorTypeBoolInKMAD, OperatorTypeBoolProgLanguage);
END_TYPE;

TYPE functionTypeInKMADBool = ENUMERATION OF(card, getValue, isExist, isEmpty,
isExistAt);
END_TYPE;

--liste non exhaustive qui dépend du language de programmation utilisé et des besoins du
concepteur
TYPE functionTypeProgLanguageBool = ENUMERATION OF(concat, add, functionCreated);
END_TYPE;

TYPE kindOfFunctionProgBool = SELECT(functionTypeInKMADBool,
functionTypeProgLanguageBool);
END_TYPE;

ENTITY boolconditionIH abstract supertype of (oneof(unitIH, composedIH));
END_ENTITY;
```

```
ENTITY composedIH
       subtype of(boolconditionIH);
       left:boolconditionIH;
       right:boolconditionIH;
       operat:OperateurProg;
       DERIVE param:LIST[0:?] of variable:=varOfBool(SELF);
END_ENTITY;

ENTITY unitIH
subtype of(boolconditionIH);
       functionRef:kindOfFunctionProgBool;
       res:boolean;
       param:LIST[0:?] of variable;
END_ENTITY;

FUNCTION varOfBool(condComp:composedIH):LIST of variable;
LOCAL
       res:LIST[0:?] of variable:=[];
       resD:LIST[0:?] of variable:=[];
       resG:LIST[0:?] of variable:=[];
END_LOCAL;
       resD:=condComp.right.param;
       resG:=condComp.left.param;
       res:=resD+resG;
       return(res);
END_FUNCTION;

END_SCHEMA;
```

9.5.9. Une expression d'action



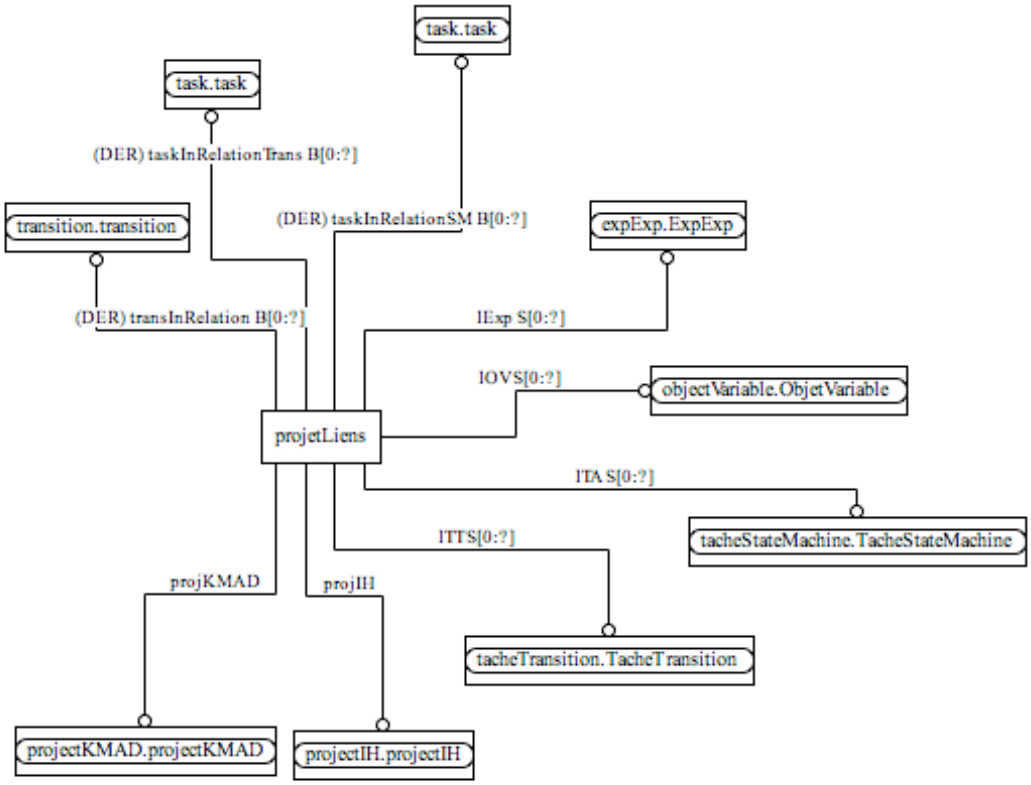

```
SCHEMA action;
use from variable;

TYPE functionTypeInKMADMan = ENUMERATION OF(card, getValue, isExist, isEmpty,
isExistAt);
END_TYPE;

--liste non exhaustive qui dépend du language de programmation utilisé et des besoins du
concepteur
TYPE functionTypeProgLanguageAct = ENUMERATION OF(concat, add, functionCreated);
END_TYPE;

TYPE kindOfFunctionProgAct = SELECT(functionTypeInKMADMan, functionTypeProgLanguageAct);
END_TYPE;

TYPE predefinedFuncTypeInKMAD=ENUMERATION OF (fadd, fcreate, fgetValue, fremove,
freplace, fset);
END_TYPE;

TYPE OperatorTypeManInKMAD = ENUMERATION OF(plus, moins, mult, divis,sup, inf, seq,
plusEgal, moinsEgal, affect, ET, OU, NON);
END_TYPE;

TYPE funcTypeInKMADMan= SELECT (predefinedFuncTypeInKMAD, OperatorTypeManInKMAD);
END_TYPE;

ENTITY actionIH;
       description:optional string;
       expression:expression;
END_ENTITY;

ENTITY expression abstract supertype of (oneof(actionUnit,actionComposed));
END_ENTITY;
```

```
ENTITY actionUnit
subtype of(expression);
       functionRef:kindOfFunctionProgAct;
       param:list[0:?] of variable;
END_ENTITY;

ENTITY actionComposed subtype of(expression);
       actions:LIST[1:?] of expression;
       DERIVE param:LIST[0:?] of variable:=varOf(SELF);
END_ENTITY;

FUNCTION varOf(actCom:actionComposed):LIST of variable;
LOCAL
       res:LIST[0:?] of variable:=[];
END_LOCAL;
       REPEAT i:=LOINDEX(actCom.actions) TO HIINDEX(actCom.actions);
              res:=res+actCom.actions[i].param;
       END_REPEAT;
       return (res);
END_FUNCTION;
END_SCHEMA;
```

9.6. Annexe 6 : La méta-modélisation des liens

Pour chacun des schémas de la méta-modélisation des liens entre les méta-modéles de K-MAD(v2) et des IH, une représentation graphique (EXPRESS-G) et une représentation textuelle du méta-modèle et de ses contraintes sont présentés.

9.6.1. Un projet de correspondance



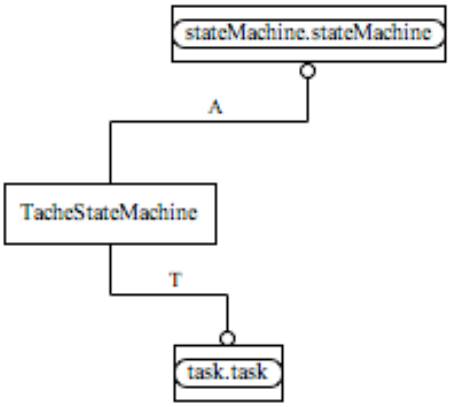

```
SCHEMA projetLiens;

use from projectKMAD;
use from projectIH;
use from TacheTransition;
use from TacheStateMachine;
use from objectVariable;
use from ExpExp;

ENTITY projetLiens;
       projKMAD:projectKMAD;
       projIH:projectIH;
       lTT:set[0:?] of TacheTransition;
       lTA:set[0:?] of TacheStateMachine;
       lOV:set[0:?] of ObjetVariable;
```

```
       DERIVE taskInRelationTrans:BAG[0:?] of task:=tacheEnTrans(SELF.lTT);
       taskInRelationSM:BAG[0:?] of task:=tacheEnAuto(SELF.lTA);
       transInRelation:BAG[0:?] of transition:=transEnTache(SELF.lTT);

END_ENTITY;
RULE RC for (TacheStateMachine, TacheTransition);
LOCAL
       composedTask:BAG[0:?] of task:=[];
       seqTaskRelation:BAG of TacheStateMachine:=USEDIN(sequence,
Task.task.decomposition);
       choiceTaskRelation:BAG of TacheStateMachine:=USEDIN(choice,
Task.task.decomposition);
       noOrderTaskRelation:BAG of TacheStateMachine:=USEDIN(noOrder,
Task.task.decomposition);
       taskConsidered:BAG of task:= taskInRelationTrans + taskInRelationSM;
END_LOCAL;
composedTask:=USEDIN(sequence, Task.task.decomposition)
       +USEDIN(noOrder, Task.task.decomposition)
       +USEDIN(choice, Task.task.decomposition)
       +USEDIN(concurrent, Task.task.decomposition);

WHERE
       RC1a :QUERY(r1 <* lTT |
                      (QUERY(r2 <* lTT-r1 |
                             (NOT(SIZEOF(r1.T.position)=SIZEOF(r2.T.position)))
                             OR (r1.D.level=r2.D.level)
                      ) = lTT -r1)
                ) = lTT;
       RC1b :QUERY(r1 <* lTT |
                      (QUERY(r2 <* lTT-r1 |
                             (NOT(SIZEOF(r1.T.position)<SIZEOF(r2.T.position)))
                             OR (r1.D.level<=r2.D.level)
                      ) = lTT -r1)
                ) = lTT;
       RC2 : QUERY( t <* composedTask |
                      (QUERY(rA <* lTA |
                             (rA.A.level <= SIZEOF(rA.T.position))
                             )
                      ) <>[]
                )=composedTask;
       RC3 : QUERY(rc <* seqTaskRelation |
                      QUERY(T1 <* rc.children |
                             QUERY(T2 <* rc.children-T1 |
                                    QUERY(RTT1 <* lTT |
                                           QUERY (RTT2 <* lTT -RTT1|
                                           ( (NOT(T2.position[HIINDEX(T2.position)] =
T1.position[HIINDEX(T1.position)]+1)
                                                  OR (RTT1.D.finish = RTT2.D.beg)) AND
((RTT1.T=T1) AND (RTT2.T=T2)))
                                           )<>[]
                                    )<>[]
                             ) = rc.children-T1
                      ) = rc.children
       ) = seqTaskRelation;
       RC4: QUERY(rc <* choiceTaskRelation |
                      QUERY(T1 <* rc.children |
                             QUERY(T2 <* rc.children-T1 |
                                    QUERY(RTT1 <* lTT |
                                           QUERY (RTT2 <* lTT-RTT1 |
                                           ((RTT1.T=T1) AND (RTT2.T=T2) AND
(RTT1.D.beg=RTT2.D.beg))
```

```
                                        )<>[]
                                    )<>[]
                                ) = rc.children-T1
                            ) = rc.children
                        ) = choiceTaskRelation;
        RC6: QUERY( r<*lTT |
                            (NOT ( (R.T.iteration.typeOfIteration <> num) OR
(R.T.iteration.val<>1) ) OR (R.D.finish=R.D.beg))
                    )=lTT;
        RC8: QUERY(t <*taskInRelationTrans |
                            QUERY(r <* lTT |
                                    ((r.T=t) AND (t.executor=Tinteractive))
                            )<>[]
                    )=taskInRelationTrans;
        RC9: QUERY(t1 <* taskConsidered |
                            QUERY ( t2 <* taskConsidered - t1 |
                                    QUERY ( m1<*t1.machine |
                                            QUERY (m2 <* t2.machine |
                                                    (m1=m2)
                                            ) <>[]
                                    )<>[]
                            )=taskConsidered - t1
                    )=taskConsidered;
        RC10 : QUERY(r <* lTT |
                            QUERY( roa <* lOV |
                                    QUERY( o <* r.T.action.action.param |
                                            QUERY ( v <* D.action.action.param |
                                                    ((r.T=roa.T) AND (r.D=roa.D))
                                            )<>[]
                                    )<>[]
                            )=lOV
                        )=ltt;
        RC11 : QUERY(r <* lTT |
                            QUERY( roa <* lOV |
                                    QUERY( o <* r.T.precondition.ensOA |
                                            QUERY ( v <* D.guard.param |
                                                    ((r.T=roa.T) AND (r.D=roa.D))
                                            )<>[]
                                    )<>[]
                            )=lOV
                        )=ltt;
END_RULE;


FUNCTION tacheEnTrans(lR : set of TacheTransition):BAG of task;
LOCAL
resTask:BAG of task:=[];
END_LOCAL;
REPEAT it:=LOINDEX(lR) TO HIINDEX(lR);
resTask:=resTask+lR[it].T;
END_REPEAT;
END_FUNCTION;

FUNCTION tacheEnAuto(lR : set of TacheStateMachine):BAG of task;
LOCAL
resTask:BAG of task:=[];
END_LOCAL;
REPEAT it:=LOINDEX(lR) TO HIINDEX(lR);
resTask:=resTask+lR[it].T;
END_REPEAT;
END_FUNCTION;
```

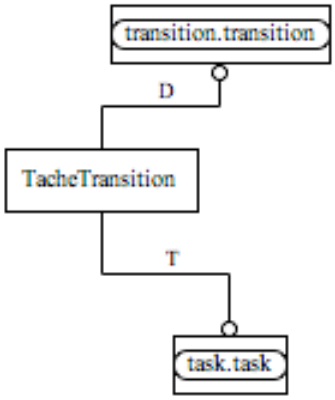
```
FUNCTION transEnTache(lR : set of TacheTransition):BAG of transition;
LOCAL
resTrans:BAG of transition:=[];
END_LOCAL;
REPEAT it:=LOINDEX(lR) TO HIINDEX(lR);
resTrans:=resTrans+lR[it].D;
END_REPEAT;
END_FUNCTION;
END_SCHEMA;
```

9.6.2. La relation Tâche/Automate



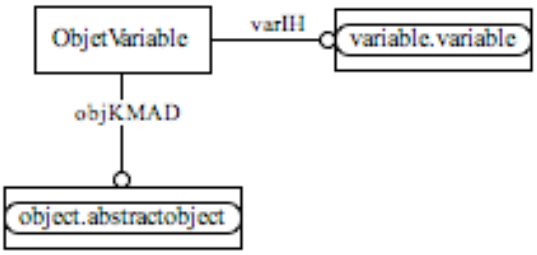

```
SCHEMA tacheStateMachine;
use from task;
use from stateMachine;

ENTITY TacheStateMachine;
T:task;
A:StateMachine;
END_ENTITY;
END_SCHEMA;
```

9.6.3. La relation Tâche/Transition




```
SCHEMA tacheTransition;
use from task;
use from transition;

ENTITY TacheTransition;
T:task;
D:transition;

WHERE
       RC7 : (SELF.T.executor<>Tuser);
END_ENTITY;
END_SCHEMA;
```

9.6.4. La relation Objet/Variable




```
SCHEMA objectVariable;
use from object;
use from variable;

ENTITY ObjetVariable;
objKMAD:abstractobject;
varIH:variable;
END_ENTITY;
END_SCHEMA;
```

Résumé. Actuellement, les applications interactives sont utilisées dans de nombreux domaines (guichets automatiques, tours de contrôle…), par des publics très différents (enfants, experts, handicapés…) et par un nombre important d'utilisateurs (interfaces de téléphones portables…) ou au contraire très spécifiques (logiciels conçus spécifiquement pour une entreprise). Elles sont de ce fait très diverses. De par la multiplicité des paramètres à prendre en compte, la conception et le développement des applications interactives sont devenus très coûteux. Afin de réduire ces coûts, des recherches sont actuellement menées sur le processus de conception. Cette thèse s'inscrit dans ces travaux. L'un des axes étudiés pour réduire le coût de production des applications interactives est la détection des erreurs le plus en amont possible pendant le processus de conception. Nous proposons de faciliter la vérification et la validation de la dynamique des applications (plus spécifiquement dénommée *dialogue*) tout au long de la conception, en fonction des spécifications recueillies auprès des futurs utilisateurs, exprimés sous forme de modèles de tâches. Les modèles de dialogue et les modèles de tâches représentent deux points de vue différents et complémentaires pour une même application. Nous proposons une approche de vérification de cohérence entre ces deux modèles tout au long du cycle de vie de l'application. Pour cela, nous avons défini des règles de cohérence entre les modèles que nous vérifions formellement en utilisant une méta-modélisation des formalismes que nous avons choisis après évaluation de leur utilisation pour une conception centrée-utilisateur.

Title. Model-Driven approach for design and checking of interactive applications : an approach based on task model

Abstract. Nowadays, interactive applications are used in many contexts (automatic cash dispenser, traffic-control…), by different ser types (children, disabled people…), for all users (phone interfaces…) or on contrary for very specific users (software designed for a specific company). They therefore are very different and they imply specific design research techniques. In order to enable such diversity, research is performed to integrate new technologies by developing new interaction techniques, new interaction supports…Due to this need of adapted interactive software and the multiplicity of the parameters that the designer has to take into account, the cost of the design and the development of interactive application increases. In order to decrease these costs, studies are performed on the design process, such as the work presented in this thesis. One of the issues studied to decrease the interactive application production cost is to detect errors as early in the design process as possible. We propose to facilitate the checking and the validation of the dynamics of applications (named *dialog*) in the whole design, according to the design requirements gathered from future users and expressed by task models. Dialog and task models express two different and complementary viewpoints on a same application. We propose an approach to check the coherence between both models during the whole life cycle of applications. This approach uses coherence rules defined during the thesis, and formally checked using a meta-modelling of both formalisms chosen after evaluation of their use in a user-centered design.

Discipline : Informatique
Mots clés : Interaction Homme-Machine, Conception centrée-utilisateur, Ingénierie Dirigée par les Modèles, Validation de la dynamique, Méta-modélisation, Modèles de Tâches, K-MAD, Interacteurs Hiérarchisés
Laboratoire : Laboratoire d'Informatique Scientifique et Industrielle