THÈSE

présentée par

Sandrine BALBO

pour obtenir le titre de
DOCTEUR de l'UNIVERSITÉ JOSEPH FOURIER - Grenoble 1
(arrêté ministériel du 5 juillet 1984 et du 30 mars 1992)
spécialité : **INFORMATIQUE**

ÉVALUATION ERGONOMIQUE DES INTERFACES UTILISATEUR : UN PAS VERS L'AUTOMATISATION

Date de soutenance : 5 Septembre 1994

Composition du jury :

Président :	**M. Yves CHIARAMELLA**
Directeur de thèse :	**Mme. Joëlle COUTAZ**
Rapporteurs :	**Mme. Marie-France BARTHET** **Mme. Jocelyne NANARD**
Examinateurs :	**M. Daniel DEGRYSE** **M. John LONG** **M. Dominique SCAPIN**

Thèse préparée au sein du
LABORATOIRE de GÉNIE INFORMATIQUE - IMAG
à L'Université Joseph FOURIER - Grenoble 1

A mes parents

Merci...

... Joëlle Coutaz pour m'avoir accueilli au sein de l'équipe IHM et pour la formation acquise au long de ces quatre années de thèse. Merci aussi pour les nombreuses conférences auxquelles j'ai eu le plaisir d'assister.

... Marie-France Barthet et Jocelyne Nanard de m'avoir fait l'honneur d'être les rapporteurs de ce travail.

... Yves Chiaramella d'avoir accepté de présider ce jury.

... Dominique Scapin d'avoir consenti à être membre du jury.

... John Long pour m'avoir accueilli à l' “Ergonomic Unit” (University College London) pendant l'été 1991, et pour avoir accepté d'être membre du jury.

... Daniel Degryse, de la société Oros, pour son intérêt à mon travail et son consentement à faire parti du jury ainsi qu'à la région Rhône-Alpes pour le financement de cette thèse.

... à toi, Alain Cottereau, dont le soutien, même depuis l'archipel du Vanuatu, m'a toujours été d'un grand réconfort.

... Nadine Ozkan et Mamoun Alissali, mes supporters préférés.

... à mes critiques assidus ou d'un jour, ils sont nombreux, merci à vous.

... Patrick (mon frère) pour tes conseils quant à mon orientation au cours de ces années d'études universitaires.

... à tou(te)s mes ami(e)s : plongeurs, motards, montagnards, musiciens ou thésards, pour les bonnes bouffes et les soirées sympa qui permettaient de recharger les batteries. Petit clin d'œil en passant à Françou Oustry et Patrick Perez qui m'ont hébergé au cours des deux derniers mois de travail sur cette thèse.

... à toute la fine équipe IHM : ceux de passage et ceux qui y était, y sont et y seront encore quelques temps.

... à tous ceux qui ont toujours su avoir un petit mot agréable aux détours des couloirs du bâtiment B du Laboratoire de Génie Informatique.

Plan de la thèse

Merci....3
Plan de la thèse....5

INTRODUCTION....7
Chapitre I : Génie logiciel et évaluations ergonomiques des interfaces utilisateur....11

PARTIE 1 : LE DOMAINE DE L'ÉTUDE....37
Chapitre II : Etude de la tâche....39
Chapitre III : Evaluation ergonomique....77

PARTIE 2 : CONTRIBUTION....113
Chapitre IV : ÉMA : un système d'analyse automatique pour l'évaluation ergonomique des interfaces utilisateur....115
Chapitre V : ÉMA : Etudes de cas....149

CONCLUSION....195

ANNEXES....203
Annexe 1 : Principes d'utilisation des diagrammes syntaxiques....205
Annexe 2 : Plate-forme d'expérimentation : Compo....209
Annexe 3 : Plate-forme d'expérimentation : Automatic Teller Machines..233
Annexe 4 : Plate-forme d'expérimentation : Médiathèque....251

Références bibliographiques....267
Table des matières....283

Introduction

L'ingénierie des Interfaces Homme-Machine (IHM) s'est manifestée jusqu'ici par le développement de modèles et d'outils d'aide à la construction d'interfaces utilisateur. Ainsi, les modèles d'architecture tel Seeheim [Pfaff 85] définissent un cadre conceptuel d'aide à la définition des composants logiciels d'un système interactif. Les boîtes à outils [Motif 90, ToolboxMac 92], les squelettes d'application tel MacApp [Schmucker 86], les générateurs d'interfaces [Normand 92a, Webster 89] ou encore les systèmes auteurs comme Hypercard [Harvey 88] offrent, chacun à leur manière, des moyens logiciels pour réaliser des IHM. Dans la démarche itérative "conception-implémentation-évaluation", il convient d'observer que les modèles d'architecture visent la conception logicielle, que les boîtes à outils et les générateurs interviennent directement dans l'implémentation mais qu'aucun outil ne s'inscrit dans l'aide à l'évaluation des interfaces utilisateur. Notre objectif vise à combler cette insuffisance.

1. Le sujet

Nos travaux de recherche ont trait à l'évaluation des IHM. Comme le montre la figure 1, ils se situent à la croisée des techniques du génie logiciel et de l'ergonomie cognitive. En génie logiciel comme en ergonomie, l'évaluation permet de contrôler les objectifs qualité. La qualité "se situe au niveau de la relation entre le sujet et l'objet. Elle est au point de rencontre entre sujet et objet" [Pirsig 78]. Dans le domaine qui nous intéresse, le point de rencontre est l'interface Homme-Machine. La qualité de cette interface conditionne la relation entre l'utilisateur et le système informatique. La qualité est "ce qui fait qu'une chose est plus ou moins recommandable, le degré plus ou moins élevé d'une échelle de valeurs pratiques" [Le Petit Robert]. Dans notre domaine d'intérêt, les recommandations sont conditionnées par des critères, si possible mesurables, qui définissent une situation de référence. Chaque critère porte un nom, un domaine de valeurs souhaité et, lorsque c'est possible, le moyen de le mesurer. L'évaluation consiste à vérifier que la situation de référence est satisfaite ou transgressée. Si les motivations et les principes sont semblables d'un domaine à l'autre, les priorités diffèrent.

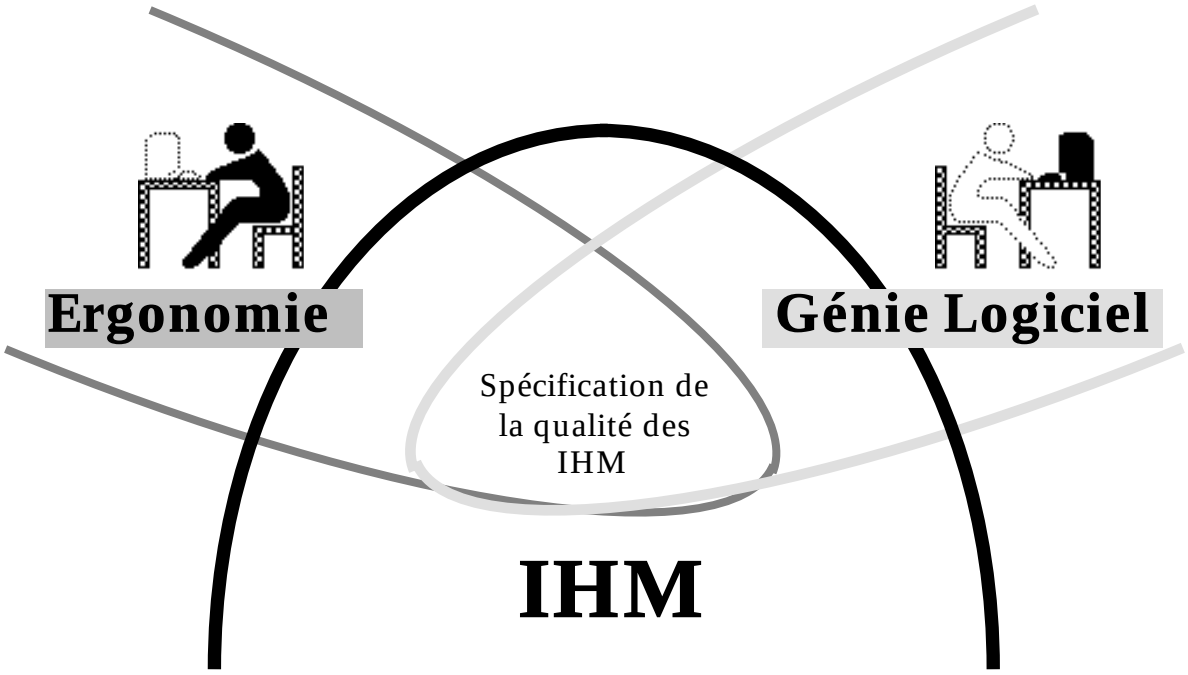


Figure 1 : L'évaluation des IHM à la croisée des techniques de l'ergonomie et du génie logiciel.

En ergonomie cognitive, le problème est de "concevoir et réaliser des interfaces Homme-Machine pour un travail efficace" [Long 89]. Typiquement, l'efficacité du travail se mesure par le temps de réalisation d'une tâche représentative, le nombre de recours à l'aide et les taux d'erreurs classées par catégorie : erreurs involontaires en action, erreurs de plan, etc. [Reason 90]. Si ces mesures servent d'indicateurs objectifs, elles ne permettent pas d'apprécier les éléments subjectifs comme le "plaisir d'utilisation des interfaces" [Nanard 90]. Il arrive parfois que les mesures de performance viennent contredire les données subjectives [Nielsen 94]. L'utilisation correcte d'une interface n'est pas le reflet de l'affectivité de l'utilisateur pour le logiciel interactif en question ; il est possible d'utiliser parfaitement un logiciel interactif qui ne plaît pas.

Le génie logiciel va au-delà de la mesure de l'efficacité du système Homme-Machine. Il embrasse tout le cycle de vie du logiciel. Les critères d'évaluation ne concernent pas seulement l'utilisabilité d'un système mais ses coûts de maintenance, sa fiabilité, etc. Ainsi, en théorie, les objectifs qualité du génie logiciel incluent les préoccupations ergonomiques. En pratique, les informaticiens sous-estiment ce problème, le traitent de manière superficielle et réductrice, ou l'ignorent délibérément. Les raisons à cela sont multiples : manque de compétences de l'équipe de développement, absence de conviction sur l'intérêt d'une telle pratique ou encore un rapport prédictif "coûts/bénéfices" non probant.

2. Objectifs

Notre constat sur la pauvreté des relations entre les pratiques du Génie Logiciel et l'évaluation ergonomique des interfaces justifie notre entreprise. Par la mise à disposition d'outils d'évaluation automatique d'interface, la mesure des performances du couple utilisateur-système informatique est facilitée, et les coûts d'évaluation s'en trouvent réduits.

Toutefois un critique logiciel aussi sophistiqué soit-il, ne peut prétendre à la complétude, surtout lorsqu'il s'agit de proposer des actions correctrices. Il doit être vu comme un support à l'évaluation, un outil de travail qui doit faciliter la détection d'anomalies de nature ergonomique. On sait aujourd'hui qu'un ergonome même expérimenté identifie en moyenne moins de la moitié des problèmes potentiels d'une interface [Pollier 91]. Notre but est de fournir à l'ergonome une aide "objective" en vue d'une évaluation plus performante. Pour ce faire, nous avons développé ÉMA, un mécanisme d'analyse automatique. Cet outil apporte à l'évaluateur un ensemble d'information sur le comportement de l'utilisateur lors de l'utilisation de l'interface en cours d'évaluation.

3. Organisation du mémoire

Nous débutons ce mémoire par une discussion sur la notion de qualité et les vues qu'adoptent à ce sujet nos deux disciplines d'intérêt : le génie logiciel et l'ergonomie. Ce premier chapitre situe notre lieu d'intervention dans le processus de développement des logiciels : l'analyse des besoins et les activités d'évaluation ergonomique dans le cycle de vie des logiciels.

Le mémoire se décompose ensuite en deux parties : un état de l'art suivi de la description de notre contribution.

L'état de l'art recouvre les activités, concept, outils et méthodes en rapport avec notre étude : l'analyse de tâche qui se pratique dans l'étape de définition des besoins, et les techniques d'évaluation applicables à chaque étape du cycle de vie. Nous consacrons un chapitre à chacun de ces points :

• Le chapitre II, sur l'analyse de tâche, vise un double objectif : clarifier la terminologie autour de la notion de tâche et définir un cadre de référence qui permette d'évaluer l'apport respectif des outils et méthodes d'analyse de tâche actuels les plus représentatifs. Au-delà de la définition d'une grille d'analyse, ce

chapitre vise un rôle intégrateur entre l'ergonomie et les pratiques du Génie Logiciel: L'analyse de tâche est souvent négligée dans la définition des besoins alors qu'elle constitue le fondement de toute conception ergonomique. Nous mettons en relief dans ce chapitre l'apport complémentaire de l'ergonomie dans une perspective informatique.

• Le chapitre III, sur les techniques d'évaluation, présente une double contribution : la définition d'un cadre taxonomique et une revue, selon ce cadre, des méthodes et outils actuels. Les dimensions de notre espace de classification permet non seulement d'étudier les techniques d'évaluation actuelles mais aussi d'envisager de nouvelles approches.

Jusqu'ici, notre contribution consiste en une définition de deux cadres taxonomiques, l'un pour l'analyse de tâche, l'autre pour les techniques d'évaluation que nous peuplons d'exemples actuels. Nous utilisons également ces espaces de référence pour situer notre propre travail qui fait l'objet de la seconde partie du mémoire :

• Au chapitre IV, nous décrivons le système d'analyse automatique que nous avons conçu : ÉMA. Ce système s'articule autour de trois composantes : celle de l'utilisateur, celle de la tâche et le mécanisme d'analyse.

• C'est au chapitre V que nous présentons les plates-formes d'expérimentation nous permettant de valider EMA : Compo, un générateur d'interfaces ; ATM, un simulateur de distributeur de billets de banque ; Médiathèque, un logiciel d'interrogation d'une base de données documentaires.

En conclusion, nous présentons les limites du développement actuel ainsi que les perspectives de cette recherche.

Quatre annexes viennent compléter ce travail. L'annexe 1 expose l'utilisation des diagrammes syntaxiques ayant servi dans l'exposé des concepts d'ÉMA. Les annexes 2, 3 et 4 présentent respectivement certains détails sur l'analyse menée pour Compo, ATM et Médiathèque.

Chapitre I

Génie logiciel et évaluations ergonomiques des interfaces utilisateur

Ce chapitre a pour objet de montrer les points d'ancrage de l'activité d'évaluation dans le processus de développement des systèmes interactifs. Le paragraphe 1 précise les liens entre les pratiques du génie logiciel (GL) et l'évaluation. Le paragraphe 2 présente les approches adoptées en ergonomie cognitive et leurs relations avec le GL. Le dernier paragraphe montre de manière synthétique comment le savoir-faire en génie logiciel peut s'enrichir des connaissances développées en ergonomie.

1. Evaluation et génie logiciel

Le génie logiciel est concerné par la production et la maintenance de logiciels sous le contrôle d'un ensemble de contraintes. C'est dans les années 70 que l'on prit conscience que le processus de développement lui-même était mal défini et qu'il nécessitait un cadre structurel. C'est ainsi que Royce proposa le modèle en cascade [Royce 70] qui fut ensuite affiné sous diverses formes avec notamment le modèle en V [McDermid 84] puis, plus récemment, le modèle en spirale [Boehm 88].

1.1. Le modèle en cascade

Dans sa première version, le modèle en cascade comportait quatre étapes que souligne la figure 1 : 1) l'analyse des besoins, 2) la conception du système, 3)l'implémentation et les tests unitaires, 4) l'intégration et le test du système.

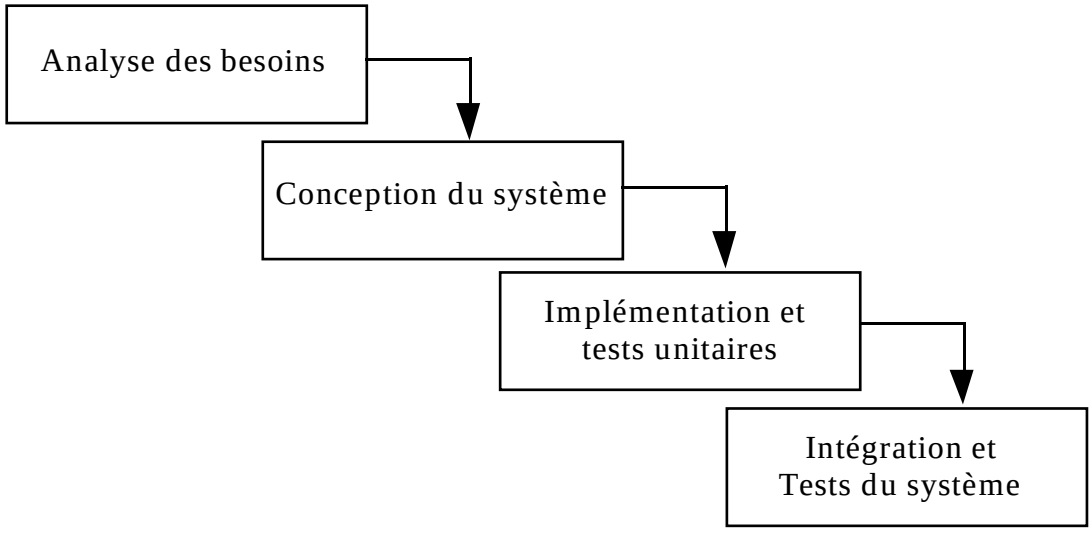


Figure 1 : Le modèle en cascade initial. Les flèches traduisent le cheminement entre étapes.

L'analyse des besoins permet d'établir, en relation avec le client, les services requis du système et les contraintes de développement. L'étape de conception consiste à définir une solution matérielle et logicielle qui répond à l'analyse des besoins et aux contraintes. L'implémentation traduit les spécifications issues de

l'étape de conception en code exécutable et modulaire. Chaque module est alors testé en vue de vérifier sa conformité avec ses spécifications. Dans la dernière étape, les modules sont intégrés et testés ensemble pour constituer un système livrable. Nous constatons que l'évaluation est effectuée tardivement et qu'elle ne concerne que les aspects logiciels du système : l'objectif directeur est l'élimination des bogues. La détection d'anomalies ergonomiques est passée sous silence.

Cette décomposition en quatre étapes s'est vite révélée insuffisante et non conforme aux pratiques. Elle s'est vue affinée de deux manières complémentaires: par l'ajout d'étapes et par la reconnaissance formelle des retours arrière. Bien qu'il n'existe pas de décomposition canonique, les étapes de la figure 2 correspondent à la pratique la plus courante. Par rapport à la structure séminale de la figure 1, les étapes supplémentaires servent à préciser les tâches de conception logicielle précédant l'activité de codage. Cette décomposition répond à la tentation de traduire directement en langage exécutable les spécifications externes produites dans l'étape de conception.

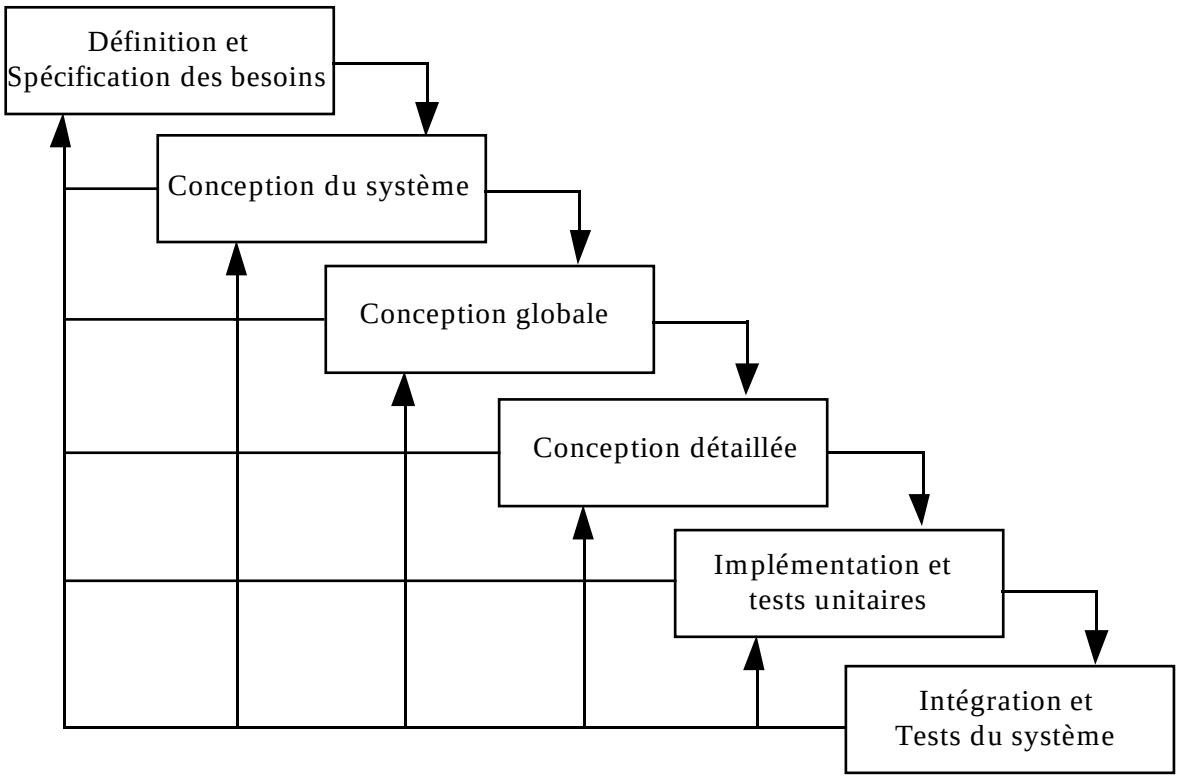


Figure 2 : Le modèle en cascade affiné. Les flèches traduisent les cheminements possibles avant et arrière. Par exemple, depuis l'étape "Intégration et tests du système", il est possible de revenir à l'étape "implémentation et tests unitaires" mais également à l'une quelconque des étapes précédentes.

Dans le schéma de la figure 2,

- on retrouve ***la définition et la spécification des besoins*** du modèle en cascade initial. Ces deux activités donnent lieu à un cahier des charges qui sert de document contractuel entre le maître d'œuvre et le client. La structure du cahier des charges répond généralement à une norme. Par exemple, la norme AFNOR regroupe à la fois la définition et la spécification des besoins. On entend par "définition des besoins" la présentation du "contexte" et une "description générale" du problème. La spécification est une "expression détaillée" du problème. Quelle que soit la norme adoptée, il faut distinguer les besoins fonctionnels c'est-à-dire les services attendus du système informatique, des besoins non fonctionnels et notamment les qualités requises. La spécification des qualités est généralement accompagnée d'un plan d'assurance qualité. Nous reviendrons sur ce point au paragraphe 1.4. Les besoins sont exprimés de manière informelle en langue naturelle mais, pour la spécification, ils peuvent être complétés de schémas ou de diagrammes formels comme le proposent SADT [Schoman 77], OOA [Coad 91], voire ADA [Watt 87].

- La ***conception du système*** donne lieu à un document de spécifications externes qui décrit le système tel qu'il sera perçu par le client. Comme pour le cahier des charges, la description est généralement informelle. En particulier, les aspects interfaces Homme-Machine sont exprimés en langue naturelle et complétés de schémas d'écran. Il existe cependant des formalismes de spécification d'IHM tels CLG [Moran 81], TAG [Payne 86], UAN [Hartson 92] et bien d'autres. Nous verrons au chapitre II l'apport de ces langages pour la vérification de propriétés ergonomiques.

- La ***conception globale*** constitue l'étape préliminaire orientée vers la mise en œuvre logicielle. Elle produit un dossier d'architecture générale. Dans le cas d'un logiciel interactif, c'est-à-dire un logiciel qui nécessite l'intervention de l'utilisateur au cours de l'exécution, l'architecture générale devra s'appuyer sur le principe de séparation modulaire entre le noyau fonctionnel qui réalise les services du domaine, et l'interface Homme-Machine qui gère les échanges avec l'utilisateur [Coutaz 90]. Les modèles conceptuels d'architecture tels Seeheim, puis Arch [Arch 92] et PAC-Amodeus [Nigay 94] aident à l'application de ce principe de base.

- La ***conception détaillée*** décrit les choix algorithmiques, précise la signature des procédures et des fonctions, définit les structures de données les plus pertinentes et spécifie les protocoles de communication. Elle peut être guidée par le principe de réutilisation logicielle ou encore être dirigée par la spécification formelle. L'avantage de la spécification formelle est la génération automatique de code qui garantit la conformité de ce code avec ses spécifications.

Le changement fondamental illustré dans la figure 2 par rapport au modèle en cascade initial tient à la possibilité de revenir sur les résultats des étapes antérieures montrant ainsi le caractère itératif du processus de développement d'un logiciel. Ces retours arrière reconnaissent que des tâches d'évaluation peuvent être entreprises à toute étape du processus de développement mais l'organisation et la nature de ces activités ne sont pas précisées. Le modèle en V vise à combler cette lacune.

1.2. Le modèle en V

Le modèle en V explicite et structure les activités de tests. Parmi ces activités, on différencie la validation de la vérification [Boehm 81] :

- La ***validation*** s'interroge sur l'adéquation du logiciel produit : "construisons-nous le bon produit ?"[1]. L'adéquation n'est pas une caractéristique mesurable mais relève d'un jugement subjectif. Le génie logiciel rejoint ici les constatations de l'ergonomie sur la pertinence des données qualitatives.

- La ***vérification*** concerne la conformité du produit avec une description de référence : "construisons-nous le produit correctement ?"[2]. La vérification n'a de sens que s'il existe un document de référence, par exemple, un dossier de spécifications détaillées.

La pente descendante du V reprend les étapes d'affinement progressif du modèle en cascade. Sur la pente ascendante, à chacune de ces étapes correspond un ensemble de tests qui permettent de vérifier et/ou de valider l'étape en regard du code produit. De manière générale, les tests sont effectués en séquence une fois le codage réalisé, mais ils doivent être spécifiés en même temps que leur

[1] : Are we building the right product ?

[2] : Are we building the product right ?

correspondant dans la pente descendante. Parfois, certaines vérifications ou validations sont effectuées lors de la descente, avant même le codage. Par exemple, si l'analyse des besoins insiste sur la modularité, les premières vérifications de ce facteur pourront être pratiquées dès les étapes de conception globale.

Dans la figure 3, on constate que les tests d'acceptation correspondent à l'étape de définition et de spécifications des besoins, que les tests du système doivent être définis en même temps que la conception, etc. Voyons ce que couvre chacune de ces catégories de tests :

- Les ***tests unitaires*** permettent de vérifier que les composants modulaires du système répondent chacun à leurs spécifications.

- Les ***tests d'intégration*** servent à vérifier que les modules réalisés indépendamment interagissent correctement.

- Les ***tests du système*** servent à vérifier que les éléments de la solution exprimés dans le dossier de spécifications externes sont présents.

- Les ***tests d'acceptation*** servent à vérifier que les besoins exprimés dans le cahier des charges du logiciel sont couverts. C'est le dernier contrôle avant la livraison du produit.

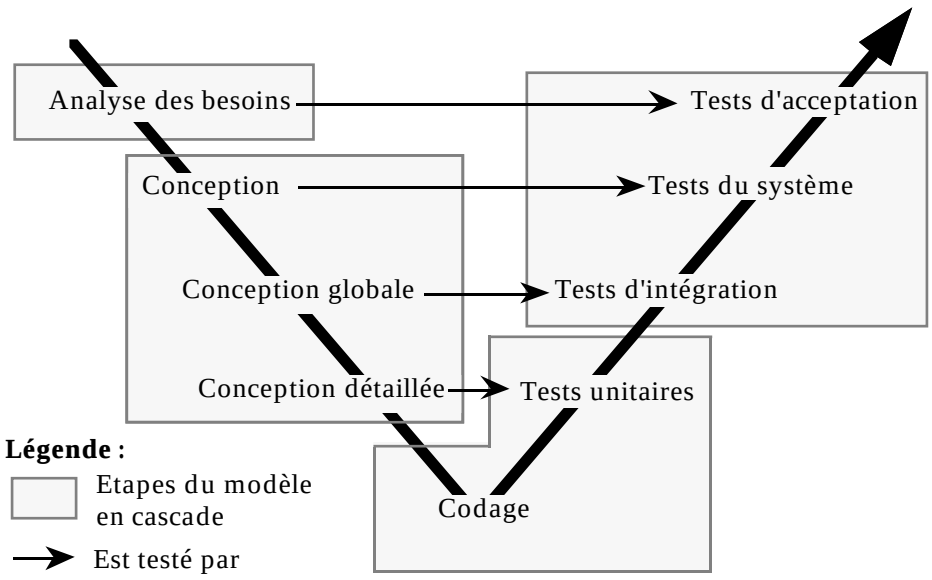


Figure 3: Le modèle en V. La flèche en trait épais et en forme de V traduit la séquence temporelle des étapes. Les carrés grisés expriment la correspondance avec les étapes du modèle en cascade.

Au travers du modèle en V, il est intéressant d'observer la corrélation entre les coûts de développement et l'étape de détection des erreurs. Comme le montre la figure 4, plus une erreur est détectée tardivement, plus le coût de correction est élevé allant de quelques dizaines de dollars lorsque le défaut est relevé pendant les spécifications jusqu'au million de dollars après la livraison du logiciel.

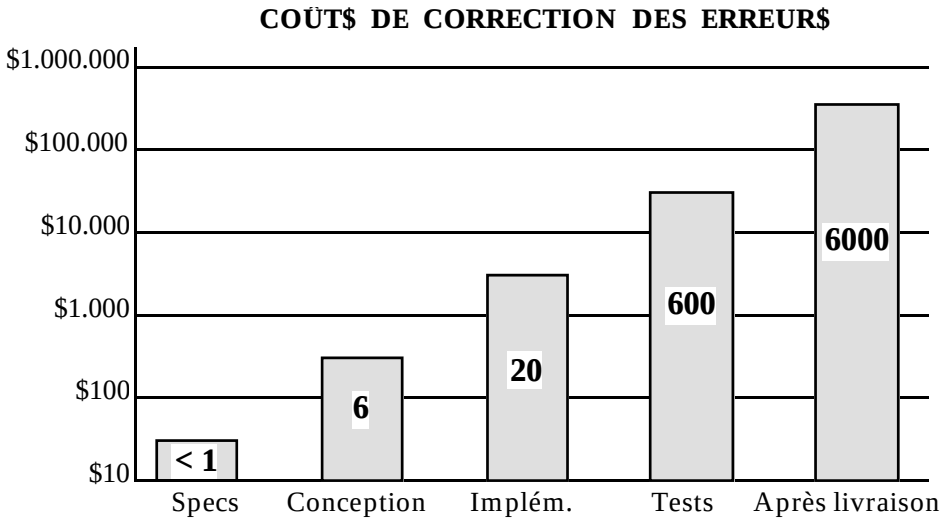


Figure 4 : Coûts de correction des erreurs (d'après une étude privée de Hewlett-Packard). Les chiffres en gras dans les colonnes correspondent aux nombres d'heures nécessaire à la correction de l'erreur détectée.

Le canevas du modèle en V ne précise pas la portée des retours arrière (limitée ou non) ni le type d'implémentation adoptée (incrémentale ou non). C'est que, de manière générale, ce modèle convient au développement de projets bien cernés. Il y a deux raisons essentielles à cela. D'une part, la définition des besoins et les spécifications externes doivent être figées dans les toutes premières étapes du processus de développement. D'autre part, le critère de transition entre deux étapes est la mise à disposition d'un document complètement élaboré. Ainsi, avec ces modèles "dirigés par les documents" [Boehm 88], les concepteurs sont amenés à produire des spécifications détaillées de services qu'ils ne maîtrisent pas encore (par exemple, l'interface utilisateur) puis conçoivent et produisent une grande quantité de code qui a de grandes chances d'être inutilisable.

Le canevas du modèle en V ne peut donc convenir au cas des systèmes à risques (objectifs mal définis comme en recherche, sécurité essentielle comme en contrôle de processus, etc.). Dans ce cas, il peut convenir d'adopter la technique du prototypage ou une approche exploratoire pour valider tout ou partie du cahier des charges et des spécifications externes. Le modèle en spirale, qui se présente comme un métamodèle, est adaptable à chacune de ces situations [Boehm 88].

1.3. Le modèle en spirale

Comme le montre la figure 5, pour chaque cycle de spirale (ou phase) le modèle rend explicites quatre activités fondamentales : 1) l'identification des objectifs de la phase, l'alternative pour atteindre ces objectifs et leurs contraintes, 2) l'analyse et la résolution des risques, 3) le développement et la vérification / validation de l'objet de la phase, 4) la planification de la phase suivante. Chaque phase définit une étape en direction du produit final. Sa nature dépend du cas à traiter. Par exemple, la phase initiale peut correspondre à l'étude de faisabilité dont l'analyse des risques conduit à la décision de développer un prototype. La seconde phase peut être la production et la validation du cahier des charges de ce prototype suivi d'un plan de développement. Notons que l'analyse des risques de cette phase a pu mettre en évidence la nécessité de valider un sous-ensemble du cahier des charges par une maquette (on assiste alors à une mini-spirale au sein de la phase). Parce que le prototype est maintenant bien cerné, le plan de développement peut traduire la décision d'utiliser un modèle en cascade pour la suite du développement du prototype.

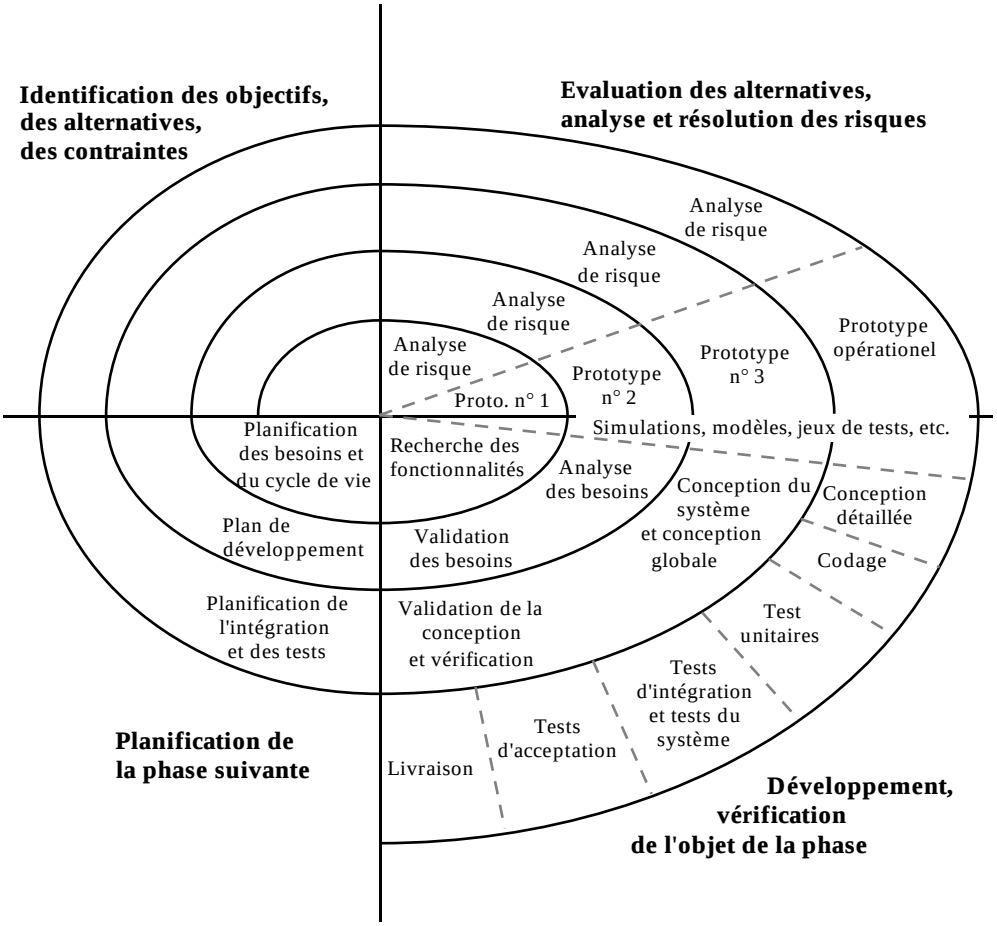


Figure 5 : Le modèle en spirale.

La modèle en spirale, en tant que métamodèle, a l'avantage d'être adaptable aux spécificités de chaque projet. Par exemple, on trouvera dans [Rettig 93] un exemple de découpage en six étapes : planification, approbation, exécution, bilan, révision et évaluation. De manière générale, le modèle en spirale aussi a l'avantage de véhiculer les activités d'analyse et de contrôle se rapportant à la gestion de projet. En particulier, il insiste sur l'identification des alternatives et exige la justification des décisions au vu d'une quantification des risques et des coûts. A ce titre, la notation QOC (Questions, Options, Criteria), conçue initialement par des concepteurs d'interface pour des concepteurs d'interfaces [McLean 91], peut utilement être étendue au cas général des prises de décision en génie logiciel et servir de notation support.

A chaque question pertinente Q, le concepteur répond par des solutions ou options O. Dans l'exemple de la figure 6, le concepteur se pose la question du choix de la représentation du curseur de la souris. Deux options s'offrent à lui : un curseur en forme de flèche ou un curseur en forme de main. Chaque option répond ou contredit un ensemble de critères C fixés par le concepteur. Dans l'exemple, la flèche est plus compacte mais plus symbolique que la main. Dans le cadre général du GL, nous suggérons que ces critères couvrent non seulement les aspects ergonomiques mais aussi le domaine du génie logiciel. En affectant des poids aux flèches, un schéma QOC facilite mais aussi documente la prise de décisions participant ainsi au facteur de traçabilité.

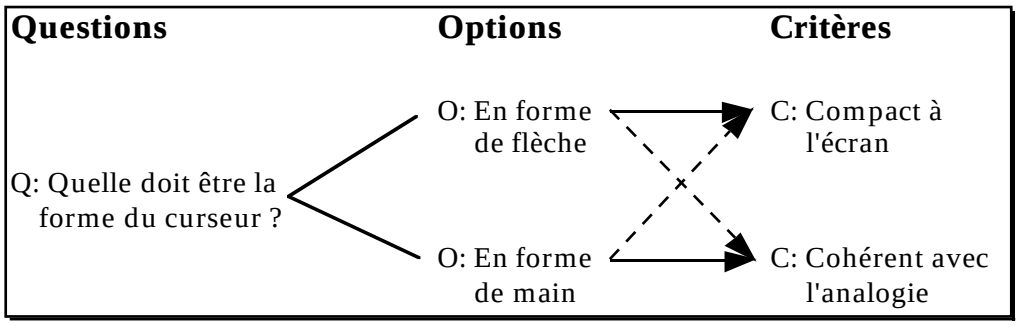


Figure 6 : Un exemple d'utilisation de la notation QOC. Les flèches pleines pointent les critères favorables à l'option en regard, et les flèches en pointillés pointent les critères non favorables.

Nous estimons que QOC n'est pas seul issu du domaine de l'ergonomie à contribuer aux bons principes du génie logiciel. Par exemple, l'affinement successif prôné par le modèle en spirale, se retrouve dans CO-SITUE [Bernsen 93]. CO-SITUE est un cadre méthodologique pour le processus de développement d'interfaces utilisateur. Ce cadre regroupe sept aspects (C,O,S,I,T,U,E) à prendre en compte lors de la conception d'artefacts informatiques, aspects qui sont progressivement affinés au cours du processus de développement :

C : la collaboration,
O : l'organisation,
S : le système,
I : l'interface utilisateur,
T : la tâche,
U : l'utilisateur,
E : l'expérience de l'utilisateur.

La présentation des modèles marquants d'aide au développement de logiciel nous a permis de situer les activités d'évaluation (validation et vérification). Nous sommes en mesure maintenant d'évoquer les pratiques de l'évaluation en génie logiciel.

1.4. Evaluation et qualité en génie logiciel

En principe, les activités d'évaluation sont régies par un plan d'assurance qualité du logiciel. La structure de ce plan répond à une norme ou encore à des directives personnalisées. Le plan précise la qualité requise, les métriques et les moyens de mesure. McCall a été le premier à caractériser la qualité en termes de facteurs et de critères [McCall 77].

Un facteur définit une propriété du logiciel susceptible d'être requise par le client. Il exprime une exigence externe. Comme le montre la figure 7, les facteurs concernent trois aspects importants d'un logiciel : ses aptitudes opératives, ses capacités de transfert et ses facultés d'évolution.

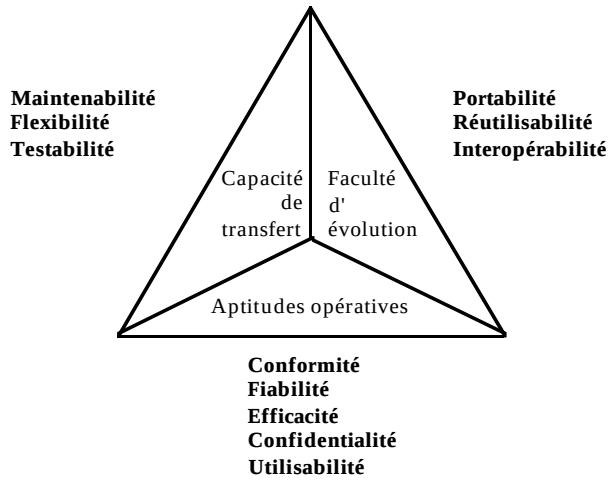


Figure 7 : Les facteurs qualité de McCall (schéma d'après [Pressman 87]).

Les facteurs qui qualifient les aptitudes opératives d'un système définissent un point d'ancrage direct avec l'ergonomie cognitive. Par exemple, l'utilisabilité traduit l'effort nécessaire pour apprendre, utiliser, préparer les entrées et

interpréter les sorties d'un programme. La conformité atteste qu'un programme satisfait à ses spécifications et notamment qu'il permet à l'utilisateur d'atteindre ses objectifs, etc.

D'autres facteurs ont également des retombées sur la qualité ergonomique d'un système mais cette fois, de manière indirecte. Par exemple, un logiciel non flexible ou non maintenable en raison d'une architecture logicielle qui ne respecte pas le principe de séparation énoncé dans [Coutaz 90], rend difficile voire impossible l'ajustement de son interface utilisateur. En conséquence, l'utilisateur doit accepter de vivre avec les défauts ou renoncer au système. Si au contraire, l'interface d'un logiciel est portable, l'utilisateur peut opérer un même logiciel sur des plates-formes distinctes sans effort de ré-apprentissage. La réutilisabilité des composants logiciels de base de l'interface permet à l'utilisateur de retrouver d'une application à l'autre le "look and feel" familier. Enfin, grâce à l'interopérabilité, il est possible d'augmenter l'espace des tâches de l'utilisateur de manière naturelle. Excel sous Macintosh, dont les services peuvent être appelés par d'autres programmes via un protocole, est un bel exemple d'extension intégrée.

Alors que le facteur définit un requis du point de vue du client, le critère est un attribut qui affecte la satisfaction d'un ou plusieurs facteurs. Le critère est un requis du point de vue du maître d'œuvre. La figure 8 montre un extrait des relations entre facteurs et critères selon McCall.

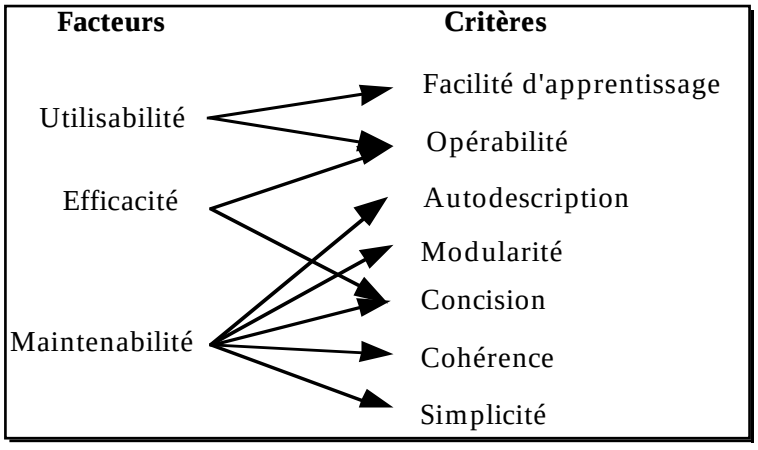


Figure 8 : Un extrait des facteurs et critères selon McCall.

Dans la figure 8, on observe que l'utilisabilité se mesure au moyen de deux critères : la facilité d'apprentissage et l'opérabilité. Comme nous le verrons plus loin, les pratiques de l'ergonomie cognitive ne peuvent se suffire de propriétés aussi générales. On trouvera dans [Palanque 92] et [Abowd 92] un effort de

précision et de structuration des critères affectant l'utilisabilité. Dans [Abowd 2], l'utilisabilité devient une catégorie qui recouvre trois classes de critères : l'apprentissage, la flexibilité de l'interaction et la robustesse de l'interaction. A leur tour, ces classes sont affinées en sous-critères :

- L'apprentissage souligne la facilité avec laquelle l'utilisateur néophyte peut démarrer effectivement une interaction et atteindre des performances raisonnables. Par exemple, la "prédictabilité", en gros synonyme de cohérence, contribue à faciliter l'apprentissage.

- La flexibilité de l'interaction reflète la multiplicité des choix dont l'utilisateur dispose pour échanger des informations avec le système informatique. Par exemple, "l'égale opportunité" laisse l'utilisateur décider des entrées à saisir comptant sur le système pour calculer le complément. Ce principe est largement appliqué dans les tableurs.

- La robustesse de l'interaction recouvre les caractéristiques de l'interface qui conduisent à l'atteinte réussie des objectifs. Par exemple, "l'observabilité" traduit la capacité du système à rendre perceptible l'état interne pertinent pour la bonne conduite de la tâche en cours.

Une fois les facteurs qualité établis en accord avec le client, il convient d'identifier les critères significatifs qui leur correspondent, de déterminer pour chacun d'entre eux, une ou plusieurs métriques c'est-à-dire des moyens de mesure, puis de fixer les domaines de valeurs autorisés. L'activité d'assurance de la qualité consiste à vérifier que la valeur du critère tombe dans les domaines de valeurs spécifiés a priori.

La difficulté surgit lorsque le critère n'est pas quantifiable en termes objectifs mais relève de jugements qualitatifs (par exemple l'autodescription). Dans ce cas, il faut définir une (ou plusieurs) métrique(s) prédictive(s) qui traduit(sent) l'attribut comportemental que l'on cherche à qualifier. Par exemple, l'autodescription peut être prédite en estimant le taux de lignes de commentaires et décider que ce taux doit être supérieur à 30%. Une deuxième métrique de l'autodescription peut s'appuyer sur le respect de normes de programmation qui fixent par exemple le format des commentaires et des mnémoniques.

Il est clair que la validité du plan qualité repose sur un savoir-faire avancé. Il faut en effet être capable de définir des métriques qui estiment "correctement" les

aptitudes du logiciel mais qui en plus respectent des contraintes de coût de mise en œuvre. Cette exigence, qui s'applique aussi à l'évaluation ergonomique, peut expliquer le scepticisme ou le désintérêt pour cette activité.

En résumé, les canevas théoriques du génie logiciel reflètent une prise de conscience sur la nécessité de représenter la part de l'utilisateur. En particulier, le plan qualité inclut les notions d'utilisabilité et de facilité d'apprentissage. La norme IEEE du cahier des charges prévoit un chapitre sur le modèle de l'utilisateur. Hélas, les rédacteurs de ces documents, les analystes du problème et les réalisateurs de tests n'ont pas toujours la compétence requise ou les méthodes pour définir et évaluer ces rubriques avec maîtrise. Alors, les descriptions sont superficielles, voire naïves, en tout cas insuffisantes pour juger de la qualité ergonomique des logiciels. Sur ces points, quel est l'apport de l'ergonomie?

2. Evaluation et ergonomie cognitive

Comme en génie logiciel, la théorie et la pratique de l'évaluation en ergonomie s'inscrivent dans des contextes différents et se rejoignent dans différentes méthodes de développement. L'ergonome est régulièrement confronté à des contextes d'évaluation différents. Chacun de ces contextes pose des questions originales et présente des contraintes spécifiques nécessitant la mise en jeu de techniques idoines [Senach 90]. Nous observons :

- le diagnostic d'usage des système existants,
- les tests réalisés en cours de conception,
- les évaluations comparatives de logiciels,
- le contrôle a priori de la qualité de l'interface.

Chacune de ces situations s'inscrit dans une méthode ou une démarche de développement.

2.1. Démarches de développement

Comme en génie logiciel, il n'existe pas en ergonomie de démarche canonique universelle pour la conception et le développement d'interfaces Homme-Machine. Dans [Long 89], Long et Dowell proposent une vision intéressante de la discipline fondée sur le savoir et la pratique. Le savoir soutient la pratique dans la recherche d'une solution au problème à résoudre (en l'occurrence, la conception d'interfaces). Comme le montre la figure 9, Long et Dowell distinguent la méthode artisanale ("craft discipline"), l'approche des sciences

appliquées (“applied science discipline”) et la démarche de l'ingénierie (“engineering discipline”).

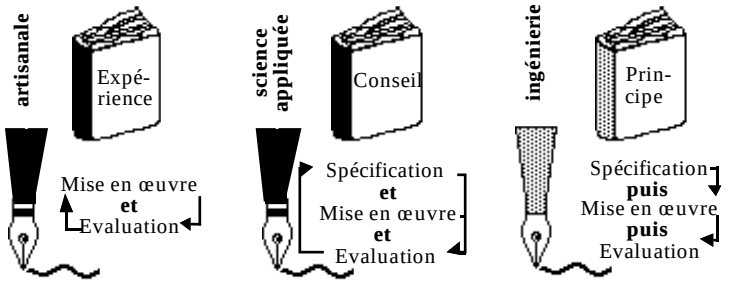


Figure 9 : Savoir et pratique dans trois approches de la discipline IHM [Long 89]. représente le savoir utilisé, la pratique associée. En noir : les approches réelles, en grisé : une approche idéale.

Approche artisanale

Dans l'approche artisanale, le savoir découle de l'expérience, de la créativité, voire de l'intuition du concepteur. Ce savoir empirique peut s'appuyer sur des principes ou des théories mais il est avant tout heuristique, c'est-à-dire implicite, informel et difficilement généralisable. Par exemple, l'énoncé “les opérations simples doivent être simples et le complexe doit être possible” ne définit pas avec précision les notions de “simplicité” et de “complexité”. Simplicité et complexité sont probablement bien cernées par les créateurs de la règle pour le cas précis du système en cours de développement. Elles exigent toutefois un effort d'interprétation certain si la règle, mise à jour au cours d'une étude particulière, doit être appliquée dans un contexte différent. Parce qu'il n'est pas transmissible en termes précis, ce savoir, sujet à interprétation, n'est pas opérationnel et ses effets ne sont pas garantis.

La pratique de l'approche artisanale rappelle la programmation exploratoire ou le prototypage progressif pratiqués en génie logiciel : d'emblée, un prototype est mis en œuvre sans spécifications préalables ou faisant suite à un minimum d'analyse. Les hypothèses de conception sont évaluées, les leçons tirées, de nouvelles solutions proposées et implémentées. On parle de conception évolutive. En génie logiciel, la pratique de la programmation exploratoire est contraire aux principes de maintenabilité ou de modularité. Elle est admise pour un sous-ensemble à risque, mais bien cerné et restreint, du système. Appliquée à tout un système, elle conduit à coup sûr au désastre logiciel. A l'inverse, en ergonomie, la démarche artisanale est une pratique courante qui conduit au succès, pour peu que le nombre d'itérations soit suffisant.

Approche science appliquée

De manière générale, le savoir en sciences appliquées repose sur la connaissance scientifique de théories, modèles et lois développés pour expliquer ou prédire un ensemble de phénomènes. Si la connaissance scientifique pure est explicite et formelle donc opérationnelle, la connaissance en science appliquée ne présente pas nécessairement les mêmes attributs. Par exemple, la théorie sur la mémorisation de Tulving [Tulving 72] définit une relation entre la spécificité de l'encodage et les facultés mnésiques. Le principe de représentation multiple adopté en conception d'interface est le reflet appliqué de cette théorie. Toutefois, comme dans le cas du savoir heuristique de l'approche artisanale, cette recommandation n'est pas opérationnelle et ses effets ne sont pas garantis : le principe ne prescrit pas la forme des représentations qui conduirait au comportement souhaité de l'utilisateur. Contrairement au savoir artisanal, ce principe à fondement théorique, permet cependant de prédire ou d'expliquer des différences comportementales chez l'utilisateur.

La pratique se manifeste par le cycle usuel "spécification-mise en œuvre-évaluation". Parce que le savoir n'est pas opérationnel, il est recommandé de produire des spécifications partielles du système avant d'aborder la mise en œuvre. La non complétude et la non garantie des effets du savoir exigent une évaluation avec retours arrière. Comme le montre la figure 10, on retrouve le schéma familier et simpliste du modèle en cascade affiné [Long 90].

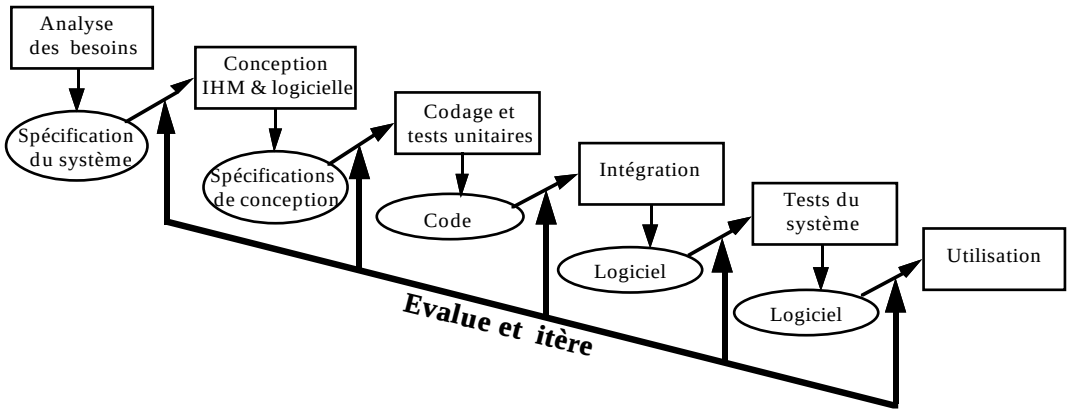


Figure 10 : Cycle de développement des logiciels, d'après [Long 90]. Les rectangles représentent les étapes de développement tandis que les ovales désignent les documents produits.

Approche ingénierie

L'approche ingénierie est une vue idéale que l'ergonomie cognitive pourrait

se fixer comme objectif. Bien que des tentatives soient menées dans cette direction, ce but est une visée à long terme. A ce propos, il faut citer [Dix 91] dont l'objectif est l'expression formelle de propriétés telles la “prédictabilité” ou l'“observabilité” (propriétés introduites au paragraphe 1.4) dans le but de prouver leur validité de manière rigoureuse.

L'ingénierie se caractérise par l'expression formalisée et opérationnelle de la connaissance. Ainsi, dans le cas qui nous intéresse, la notion de “simplicité” aurait une définition non ambiguë réutilisable et prescriptive. La pratique rappelle le modèle en cascade original du génie logiciel. Elle se caractérise par une étape de spécifications complètes du système suivie de la mise en œuvre et de l'évaluation.

En résumé, seul le savoir de l'ingénierie est opérationnel, transférable, généralisable, testable. Le savoir empirique de l'approche artisanale et le savoir scientifique appliqué sont informels et donc sujets à interprétation. Le savoir artisanal ne peut se transmettre que par l'exemple. Le savoir appliqué peut être prescriptif mais, issu de théories et de modèles qui visent à prédire ou expliquer, il n’est pas toujours adapté à la conception. Parmi les trois approches, seule la pratique artisanale raisonne directement sur l'artefact sans spécifications préalables. A la différence de l'approche science appliquée qui suppose des spécifications partielles, les spécifications en ingénierie sont complètes. Cette différence tient à la nature prescriptive du savoir de l'ingénierie. En version artisanale, l'évaluation permet d'informer. En ingénierie, elle a pour mission de modifier les spécifications. En science appliquée, elle vise non seulement à modifier l'existant mais à le compléter.

La classification de Long et Dowell a l'avantage de montrer les limites actuelles des pratiques en conception d'IHM et d'ouvrir des perspectives en direction des sciences “dures”. En l'état, l’approche science appliquée est, à notre avis, la seule méthode réaliste. Il ne faut pas cependant écarter la rigueur formelle de l’ingénierie lorsque celle-ci peut utilement informer la conception sur un point précis.

Dans ce qui suit, nous abordons la qualité telle qu’elle se pratique en ergonomie et montrons comment elle s’inscrit dans les différentes approches de la classification de Long.

2.2. Qualité en ergonomie

Comme le montre la figure 11, il est d'usage en ergonomie de distinguer deux dimensions dans l'évaluation d'un système : l'utilité et l'utilisabilité [Senach 90]. L'utilité a trait à l'adéquation fonctionnelle : le logiciel permet-il à l'utilisateur d'atteindre ses objectifs de travail? L'utilisabilité concerne l'adéquation de l'interface Homme-Machine : le logiciel est-il facile à apprendre et facile à opérer?

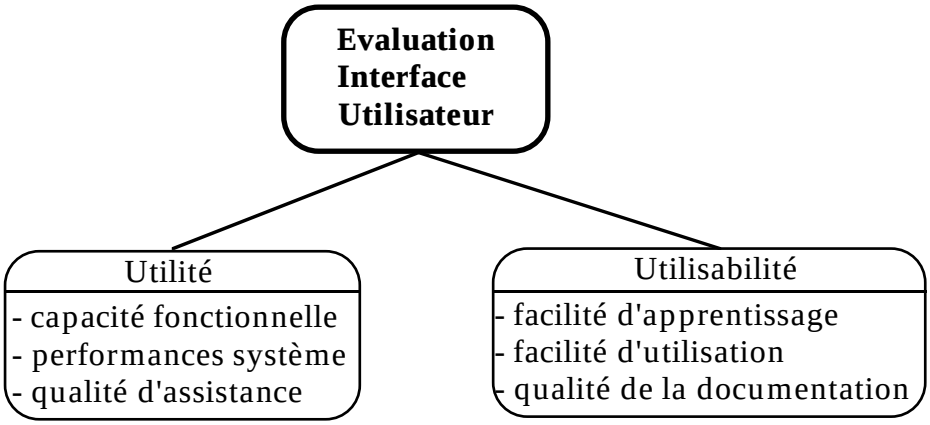


Figure 11 : Un point de vue sur l'évaluation des interfaces utilisateur [Senach 90].

Utilité et utilisabilité sont deux notions évoquées en génie logiciel en termes de besoins fonctionnels et qualités requises (voir paragraphe 1.4). Dans le cadre plus précis de l'architecture globale d'un système interactif, l'utilité du système est représentée par le noyau fonctionnel du logiciel tandis que l'utilisabilité est véhiculée par les composants de l'interface Homme-Machine. Si l'utilité et l'utilisabilité sont deux exemples parmi les 11 facteurs de McCall, ces notions constituent les piliers de l'évaluation en ergonomie. Avant de montrer comment ces deux facteurs sont mesurables en ergonomie, il nous faut présenter le schéma méthodique recommandé : la conception centrée sur l'utilisateur.

2.2.1. Conception centrée sur l'utilisateur

En génie logiciel, on s'en souvient, le développement correct d'un système informatique augmente les chances de réussite du projet et, par voie de conséquence, la qualité du produit final. La conception centrée sur l'utilisateur préconisée en ergonomie participe à cet objectif. Cette méthode signifie :

- l'implication de l'utilisateur dans l'équipe de conception dès la définition des besoins et des spécifications externes ("participative design") ;

- la réalisation de tests d'utilisabilité ("experimental design"). Selon le niveau d'avancement du système, ces tests peuvent aller du simple interview à l'expérimentation sur un prototype opérationnel, voire le système livrable (notons que le modèle en spirale convient bien à cette forme de recommandation) ;

- l'analyse des tests d'utilisabilité et la mesure attentive de l'impact de la solution d'amélioration ("iterative design"). La vérification a priori de l'effet des aménagements peut paraître réductrice. Elle reste néanmoins supérieure à l'absence de mesure prédictive. Ensuite, les coûts de mise en œuvre des modifications envisagées sont discutés avec les implémenteurs. Il est alors important de justifier le choix adopté et de le documenter par exemple au moyen de la notation QOC (cf. le facteur de traçabilité de McCall).

La méthode centrée sur l'utilisateur augmente les chances de réussite mais n'élimine pas nécessairement les erreurs de conception. Citons les écueils les plus fréquents : le choix des sujets, la validité écologique des tests et le choix des critères qualité.

- L'échantillon des utilisateurs retenus doit être représentatif des utilisateurs potentiels du logiciel. Cette représentativité est parfois difficile à valider. Même si celle-ci est satisfaisante, la motivation et la capacité d'expression des sujets sont des facteurs déterminants. Le travail d'analyse est délicat car il faut être capable de détecter les points d'inconfort du sujet même si ce dernier n'est pas conscient du problème d'utilisabilité.

- La validité écologique des conditions d'expérimentation traduit le souci de reproduire en laboratoire le contexte réel d'utilisation du logiciel ("Design in the zoo" par opposition à "Design in the wild"). Par exemple, on peut demander au sujet de réaliser une tâche réelle ou une version simplifiée de cette tâche. La version simplifiée permet de contrôler les paramètres et d'évaluer les différences de comportement en fonction de leurs variations. Il convient cependant de s'interroger sur la validité des conclusions lorsque la tâche sera réalisée dans le contexte réel. L'expérimentateur doit aussi étudier l'impact de l'aide qu'il a apportée pendant l'expérimentation sur la facilité d'apprentissage dans le monde réel.

- La difficulté du choix des critères d'utilisabilité n'est pas la moindre. Quels sont-ils et comment les mesurer? Autant de questions difficiles dont dépendra la validité de l'interprétation des résultats observés. Par exemple, on trouvera dans [Ravden 89] des recommandations pratiques quant au choix de questions portant sur la mise en évidence de problèmes d'utilisabilité.

La conception centrée sur l'utilisateur s'inscrit différemment dans les pratiques de la classification de Long. Dans l'approche artisanale, l'utilisateur intervient essentiellement dans l'étape d'évaluation, l'analyse des besoins étant généralement à peine esquissée. Pour les approches science appliquée et ingénierie, l'implication de l'utilisateur est manifeste aux étapes de spécification et d'évaluation. La différence réside, à notre sens, dans les techniques de définition et de mesures de la qualité ergonomique. Ce point sera repris au paragraphe suivant.

2.2.2. Mesure de la qualité ergonomique

En ergonomie, évaluer un système consiste à estimer la conformité des performances effectives avec les performances désirées du système [Dowell 89]. Ici, la notion de système recouvre le couple “utilisateur-artefact”. Dans le cas qui nous intéresse, l'artefact est un système informatique matériel et logiciel. A la différence du génie logiciel qui voit dans la notion de système la solution logicielle et matérielle, en ergonomie cognitive, les performances observées encapsulent celles de l'utilisateur “augmenté” de l'artefact. L'estimation de la conformité des performances effectives avec les performances désirées se pratique différemment selon les approches artisanale, science appliquée et ingénierie.

Dans l'approche artisanale, l'évaluation ergonomique repose sur l'expertise de l'évaluateur. Celui-ci, s'appuyant sur son savoir-faire, estime les performances désirées de manière intuitive. La mesure des performances effectives, déduite des observations expérimentales, est tout aussi informelle. Nous reviendrons sur ce point au chapitre 3.

Dans une pratique orientée ingénierie, les évaluations approximatives et intuitives sont autant que possible écartées. Le processus d'évaluation commence par la définition d'une situation de référence (au sens de Senach). Pour [Dowell 89], la situation de référence comprend deux ingrédients : la performance de réalisation d'une tâche donnée et son coût en ressources. Comme le montre la figure 12, Dowell et Long distinguent deux catégories de

coûts : les coûts en ressources structurelles et les coûts en ressources comportementales. Les premiers traduisent l'effort d'acquisition et de maintenance de structures. Ces structures conditionnent le comportement durant l'accomplissement de la tâche avec l'artefact. Les coûts en ressources comportementales expriment l'effort de recherche des structures nécessaires à l'accomplissement de la tâche et nécessaires à l'expression du comportement.

Structure et comportement peuvent être de nature physique ou mentale. Le squelette et sa musculature sont des structures physiques tandis que la connaissance d'un fait est une structure mentale. Les mouvements du corps sont des comportement physiques alors que l'activité de planification traduit un comportement mental. Les coûts mentaux, qu'ils soient structurels ou comportementaux, se divisent en trois branches : cognitifs, conatifs et affectifs. Les coûts cognitifs sont liés à l'effort d'apprentissage, les coûts conatifs désignent l'effort d'engagement et de prise de décision et les coûts affectifs modélisent l'état de satisfaction et plaisir.

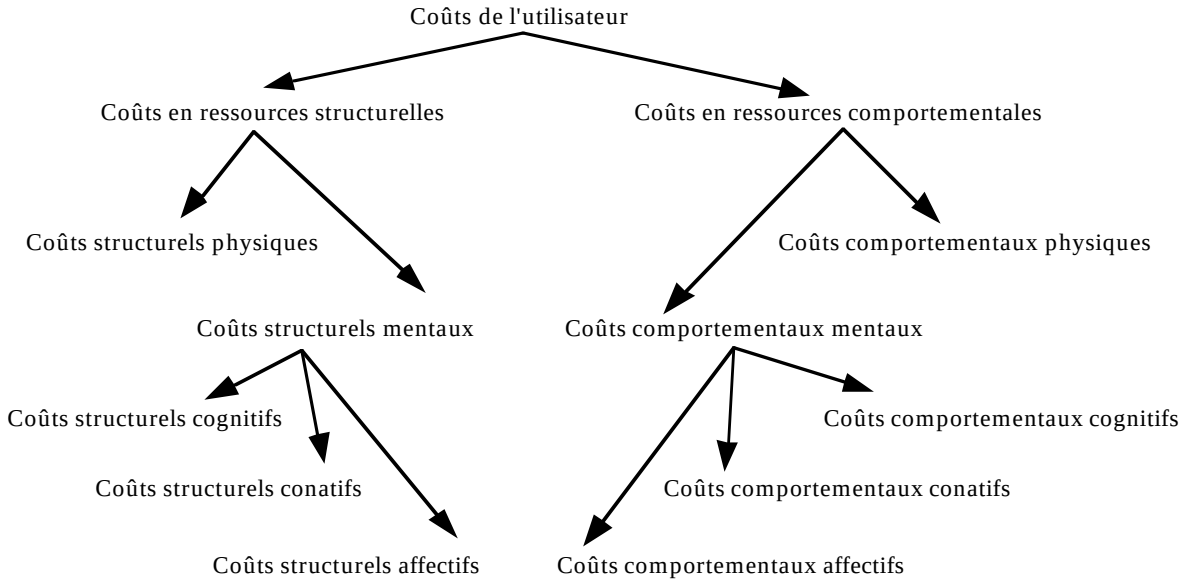


Figure 12 : Coûts en ressources mis en jeu par le sujet humain.

A titre d'illustration, nous reprenons à notre compte l'exemple du distributeur automatique de billets présenté par Whitefield et ses co-auteurs dans [Whitefield 91]. Dans cet exemple, la performance désirée est la production du billet de train désiré et la réception d'une somme d'argent donnée et répondant au coût suivant : au plus deux minutes pour planifier et prendre une décision (coût comportemental cognitif), pas d'augmentation de la frustration (coût comportemental affectif), au plus un recours à l'aide (coût structurel cognitif),

pas d'inconfort pour la saisie des données (coût comportemental physique), et un temps de réponse de l'artefact inférieur à une seconde (coût de traitement du système informatique).

Cet exemple montre la similitude de la méthode avec les principes de l'assurance qualité pratiquée en génie logiciel : il faut spécifier la nature de la "performance" en termes de critères si possible quantifiés. Comme en génie logiciel, la difficulté est de définir la situation de référence et de vérifier la conformité. En génie logiciel, les facteurs et critères à-la McCall peuvent servir de point de départ à la définition de la situation de référence. Certains et notamment l'utilisabilité rejoignent le camp de l'ergonomie. Malheureusement, l'ergonomie, tout comme le génie logiciel, n'offre pas de réponses structurées sur la façon de mesurer ce facteur. C'est l'objet de l'ingénierie de rendre ces facteurs et critères plus précis.

Les mesures permettent de détecter une part seulement des problèmes d'utilisabilité ou d'utilité. Par exemple, pour évaluer la qualité des messages d'erreur on relève le type d'erreur commise, la fréquence de l'erreur, la fréquence des recours à l'aide, le temps passé à consulter la documentation et la durée du raisonnement. Senach fait observer que ces données quantifiables ne sont pas suffisantes pour décider de la qualité informative des messages. Il estime que les données qualitatives tels les commentaires exprimés par les utilisateurs pendant les séances de test présentent davantage d'intérêt [Senach 90].

Quelle que soit la technique adoptée, la non conformité des performances observées par rapport aux performances désirées constitue la toute première étape du processus d'évaluation. Il reste ensuite à interpréter le problème soulevé puis à trouver les bons remèdes. En raison de son support théorique explicatif, l'approche science appliquée est mieux armée pour l'interprétation que la méthode artisanale. Quant à l'identification des remèdes, l'approche orientée ingénierie serait supérieure à celle des sciences appliquées par l'apport de réponses formalisées et opérationnelles.

3. Synthèse

Le génie logiciel propose un cadre structurant du processus de développement des systèmes informatiques. Dans ce cadre, l'ergonomie s'inscrit en des étapes bien précises avec ses méthodes et techniques. La figure 13 illustre cette relation pour le cas simplificateur, mais réaliste, du modèle en V. Dans notre représentation synthétique de la figure 13, nous répartissons les étapes du

processus de développement en deux espaces : l'espace IHM et l'espace logiciel. Le premier se caractérise par la priorité qu'il faudrait accorder aux aspects ergonomiques, le second par l'accent mis sur les techniques d'implémentation logicielles.

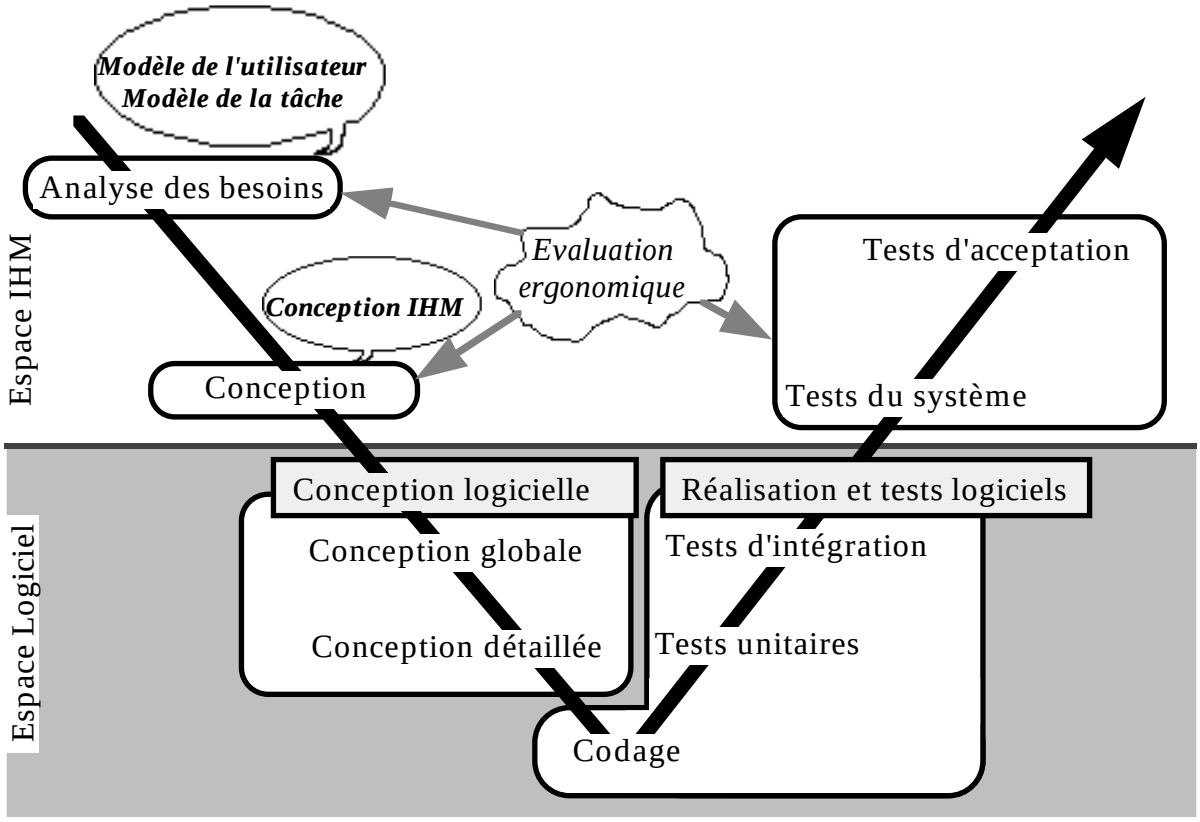


Figure 13 : Espaces IHM et logiciel dans le processus de développement des logiciels.

L'espace IHM inclut l'analyse des besoins, la conception et les tests du système et d'acceptation. Dans l'analyse des besoins, il faut inscrire l'apport de l'ergonomie et de la psychologie pour l'analyse de tâche et la modélisation de l'utilisateur. Nous reviendrons sur ces techniques au chapitre 2. A l'étape de conception, intervient l'ergonomie pour la définition et la spécification de l'interface utilisateur. Cet aspect est évoqué mais non développé dans notre mémoire. Les tests système et d'acceptation devraient nécessairement faire appel aux techniques d'évaluation. Nous ferons une revue des méthodes d'évaluation au chapitre 3 avant de présenter nos travaux sur ce sujet. Nous verrons alors que l'évaluation apparaît en filigrane dès les étapes d'analyse et de conception et que l'évaluation effectuée seulement en fin du processus de développement conduit au désastre ergonomique.

L'espace logiciel laisse la place aux compétences informatiques avec les conceptions globales et détaillées, le codage et les tests unitaires et d'intégration. Par souci de clarification, nous appelons ***conception logicielle*** les activités de conception globale et détaillée. Nous l'opposons à ***conception d'IHM***, activité amont qui implique une analyse centrée sur l'utilisateur.

A l'évidence, nos travaux sur l'évaluation des interfaces utilisateur s'inscrivent dans l'espace IHM du processus de développement. Et pour bien préciser la scène, nous allons tenter de répondre aux six questions de la méthode quintilienne[3] adoptée en technique de communication :

- Quoi ? De quoi s'agit-il ? (objet, opération, nature, quantité, etc.)
- Qui ? Qui est concerné ? (exécutants, nombre, qualification, etc.)
- Où ? Où cela se produit-il ? (situer le problème dans l'espace)
- Quand ? Quand cela survient-il ? (situer le problème dans le temps)
- Comment ? Comment procède-t-on ? (de quelle manière cela se passe-t-il?)
- Pourquoi ? Pourquoi cela se passe-t-il ainsi ? (causes, raisons, etc.)

De quoi s'agit-il?

Il s'agit de contrôler la qualité ergonomique des interfaces utilisateur. Cette qualité intègre les notions de performance et de coûts mesurables et s'inscrit dans le plan assurance qualité du génie logiciel.

Qui effectue l'évaluation?

L'évaluation de la qualité ergonomique des logiciels est le fait d'experts ergonomes qui peuvent se faire aider d'utilisateurs représentatifs. Notre objectif est de fournir des outils qui facilitent ce travail en sorte que l'évaluation puisse être prise en considération par des informaticiens, dès les premières étapes du cycle de développement.

Où s'effectue l'évaluation?

L'évaluation peut se pratiquer en laboratoire ou sur le terrain, c'est-à-dire dans le contexte réel d'utilisation. En laboratoire, l'évaluation peut se faire sans outillage spécialisé. Seule la compétence de l'évaluateur intervient (heuristiques et théories prédictives). Elle peut aussi se pratiquer dans un laboratoire d'utilisabilité, local doté d'un équipement spécialisé (régie audio-visuelle, reconstitution du contexte réel d'utilisation, etc.).

---

[3] : Marcus Fabius Quintilianus : enseigna la réthorique à Rome vers les années 100 après JC

Quand s'effectue l'évaluation?

L'évaluation se pratique dans l'espace IHM du processus de développement (voir figure 13). C'est ce que souligne l'esprit des méthodes JSD* [Lim 92] ou DIANE [Barthet 88], méthodes sur lesquelles nous reviendrons au chapitre 2.

Comment procède-t-on pour évaluer les interfaces ?

La réponse dépend de l'approche (artisanale, science appliquée ou ingénierie). Généralement, le processus commence par la définition d'une situation de référence. Il inclut ensuite une phase d'expérimentation puis une analyse comparative des performances attendues et effectives. Il s'agit ensuite d'expliquer les phénomènes et d'identifier les remèdes. Il existe d'autres techniques et notamment les approches prédictives qui ne nécessitent pas de phase expérimentale. Le détail sera présenté au chapitre 3.

Pourquoi évalue-t-on les interfaces utilisateur?

L'objectif est de "concevoir et réaliser des interfaces Homme-Machine pour un travail efficace" [Long 89].

Nous venons de situer notre travail dans le contexte du génie logiciel. Nous observons que la première étape du processus de développement, l'analyse des besoins, inclut l'analyse de tâche. Le produit de cette analyse est un modèle de tâche, c'est-à-dire une représentation de l'espace des tâches dans lequel l'utilisateur pourra naviguer via le système informatique. Ce modèle est l'une des données support en évaluation d'interface. La description des méthodes d'analyse fait l'objet du prochain chapitre.

Partie 1

Le Domaine de l'Étude

Chapitre II

Etude de la tâche

Au chapitre précédent, nous avons situé le lieu d'intervention de l'analyse de tâche dans le processus de développement des systèmes interactifs. L'objectif de ce chapitre est de cerner la définition de la notion de tâche et de préciser le rôle des modèles et méthodes supports.

Nous introduisons le sujet par une définition du concept de tâche, élément central de notre discussion. L'analyse des modèles actuels nous conduit à clarifier et à organiser l'espace des possibilités selon deux axes d'observation : la finalité et le formalisme. Nous les présentons au paragraphe 2 et les illustrons en 3 et 4 respectivement. En conclusion, nous proposons une synthèse des modèles étudiés en fonction de leur finalité et des formalismes supports.

1. Définitions

Nous tentons ici de clarifier le sens des termes clefs de notre sujet de recherche avec les notions de tâche, d'activité, de modèle et d'analyse de tâche. Selon les perspectives, et notamment pour la psychologie et l'informatique, ces termes n'admettent pas nécessairement les mêmes définitions. Nous serons donc amenés à faire des choix lorsqu'ils s'avèrent indispensables à la cohérence de notre analyse.

1.1. Tâche

D'une manière générale, une tâche désigne un "travail déterminé qu'on doit exécuter" [Le Petit Robert]. En restreignant la portée de cette définition au cas de l'interaction Homme-Machine, Normand voit dans la tâche "un objectif à atteindre par l'utilisateur à l'aide d'un système interactif" [Normand 92a, p. 11]. Cette définition se précise ensuite.

- une ***tâche*** est un but que l'utilisateur vise à atteindre assorti d'une procédure (ou plan) qui décrit les moyens pour atteindre ce but [Normand 92b] ;

- un ***but*** est un état du système que l'utilisateur souhaite obtenir. En vérité, si l'état du système n'est pas observable (au sens où nous l'avons décrit au chapitre I), l'état effectif du système risque d'être différent de l'état que l'utilisateur perçoit et interprète (selon les étapes schématiques de Norman [Norman 86]). Dans la discussion qui suit, nous ne ferons pas de distinction entre l'état réel du système et la représentation mentale dont l'utilisateur se fait de cet état ;

- une ***procédure*** est le composant exécutif d'une tâche : un ensemble d'opérations organisé par des relations temporelles et structurelles ;

- une ***opération*** est à son tour une action ou une tâche, montrant ainsi le caractère récursif de la définition du concept de tâche. Selon Caelen et Fréchet, le comportement humain présente "une composante mentale qui se manifeste par des attitudes cognitives et une composante physique qui se manifeste par des actions" [Caelen 92] ;

- une ***action*** désigne une opération terminale ; elle intervient dans l'accomplissement d'un but terminal ;

- un but terminal et les actions qui permettent de l'atteindre définissent une ***tâche élémentaire*** ;

- une ***procédure élémentaire*** est la procédure associée à une tâche élémentaire ;

- les ***relations temporelles*** entre les opérations d'une procédure traduisent la séquentialité, l'interruptibilité, voire le parallélisme ;

- les ***relations structurelles*** servent à exprimer la composition logique des opérations ou la possibilité de choix ;

- une tâche qui n'est pas élémentaire est dite composée ou encore : une ***tâche composée*** est une tâche dont la description inclut des sous-tâches.

Une représentation hiérarchique comme celle de la figure 1 traduit cette relation de composition. Les feuilles de l'arbre constituent les tâches élémentaires. Le lien avec la planification hiérarchique adoptée en Intelligence Artificielle et qui constitue l'un des principes de fonctionnement du Modèle du Processeur Humain, est clair.

La figure 1 résume la définition à laquelle nous adhérons, de la notion de tâche.

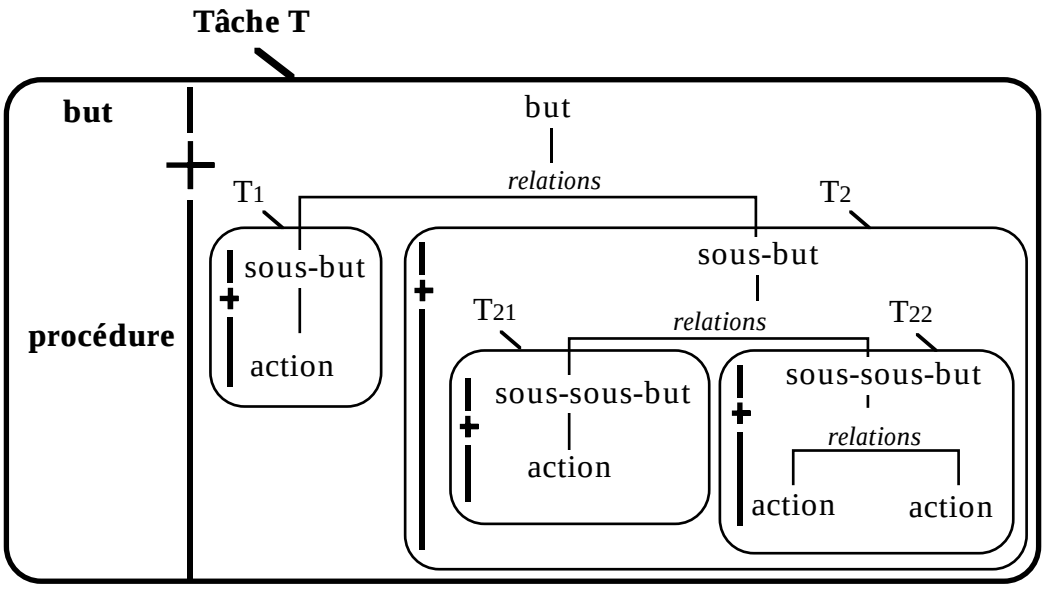


Figure 1 : Une définition de tâche [adaptée de Normand 92b]. Un rectangle encapsule une tâche. Tâche = but + procédure. Les relations sont représentées par une structure arborescente. Dans l'exemple, le but est atteint par l'accomplissement de deux sous-tâches : T1 et T2. A T1 correspond un sous-but qui peut être atteint par une action. T2 correspond à un sous-but qui doit être décomposé plus avant. T1, T21 et T22 sont des tâches élémentaires, T et T2 sont des tâches composées.

Si l'on s'accorde sur les concepts constituants d'une tâche, il convient de noter la diversité du vocabulaire et la variété des extensions qui viennent enrichir le canevas de la figure 1. Concernant le vocabulaire, selon les auteurs, une tâche élémentaire s'appelle "tâche simple" [Payne 86], "tâche unitaire" ou encore "tâche de base" [Tauber 90].

Dans TAG, une "tâche simple" (simple task) représente la perspective cognitive : "toute tâche de routine que l'utilisateur peut accomplir sans recours à des structures de contrôle, tels le branchement et l'itération, qui exigent une vérification de l'avancement du plan"[4] [Payne 86, pp. 121]. Cette définition de la notion de tâche simple est à rapprocher du niveau comportement des habiletés du modèle de Rasmussen [Rasmussen 86]. La "tâche de base" de ETAG (présenté au paragraphe 3.2.2) n'a pas de fondement cognitif : elle désigne une commande atomique du système.

Dans ETIT (dont nous préciserons le fonctionnement au paragraphe 3.2.3), Moran fait la distinction entre "tâche interne" et "tâche externe". La première correspond à la notion de tâche de base au sens de ETAG (perspective système), la seconde à celle de tâche simple au sens de Payne (perspective psychologique) [Moran 83].

[4]En anglais : "any task that the user can routinely perform with no demand for a control structure, such as branching and iteration, that requires monitoring of plan progress"

Concernant les extensions, il est fréquent que la description d'une tâche soit assortie de préconditions (comme dans UAN [Hartson 92] ou MAD [Pierret-Goldreich 89]), de l'expression du temps d'exécution prévu (comme dans CPM-GOMS [Gray 93]) ou encore du caractère obligatoire ou facultatif d'une tâche (comme dans DIANE [Barthet 88] ou MAD). Ces différences reflètent des motivations ou des usages distincts sur lesquels nous reviendrons dans la suite de ce chapitre. La granularité des tâches élémentaires illustre également des différences d'objectif.

1.2. Granularité d'une tâche élémentaire et action

Quel que soit le vocabulaire adopté, le concept de tâche élémentaire recouvre, selon le point de vue, un niveau de granularité ou un niveau de compétence cognitive. En particulier, GOMS [Card 83] recommande 4 niveaux de granularité : niveaux tâche, fonctionnel, argument et "keystroke". Les opérations d'un niveau se caractérisent par des temps d'exécution de même ordre de grandeur. Dans CPM-GOMS, ces niveaux ont été affinés pour traduire des activités, tel "percevoir signal sonore", en liaison avec les concepts du Modèle du Processeur Humain [Gray 93].

Avec UAN [Hartson 92] et MAD [Pierret-Goldreich 89], les niveaux sont laissés à la discrétion du concepteur allant des fonctions du système jusqu'à la description d'actions physiques. Dans ce contexte, une ***action physique*** est une opération indivisible, ou primitive, que l'utilisateur applique à un dispositif d'entrée physique et qui a pour effet immédiat de modifier l'état de ce dispositif. Par exemple, enfoncer et relâcher une touche du clavier sont deux actions distinctes. Symétriquement, une action physique émanant du système est une opération indivisible qui affecte l'état perceptible du système, par exemple, un changement de couleur, l'apparition d'une fenêtre, mais aussi, comme en vision active [Crowley 94], l'ouverture et l'orientation de l'objectif de la caméra.

Pour certains besoins, et nous reviendrons sur ce point au chapitre VI, il convient d'élargir la portée de la notion d'action physique. Du point de vue humain, une action physique est un ***acte perceptible*** qui résulte des traitements du système cognitif en direction du système moteur. Si l'on se réfère au modèle de Rasmussen, ces traitements sollicitent différents niveaux de performance cognitive (habiletés, procédés, et connaissance profonde). En ingénierie des logiciels, à la notion de niveau cognitif, nous faisons correspondre celle de niveaux d'abstraction. Comme le montre le modèle logiciel Pipeline [Nigay 94], une action physique système est un acte perceptible qui résulte des traitements

du système pratiqués à différents niveaux d'abstraction en direction des dispositifs physiques de sortie ; et une action physique utilisateur est un acte produit par l'utilisateur en direction des dispositifs physiques d'entrée du système.

Un acte perceptible n'est pas nécessairement perçu. Par exemple, il est possible que l'utilisateur ne détecte pas le changement de couleur d'un motif. Dans ce cas, l'action physique système n'a pas été perçue par les organes visuels de l'utilisateur. En l'absence de caméra, le système ne perçoit pas certaines actions physiques de l'utilisateur tel le mouvement des yeux. Nous retiendrons donc ceci:

- une ***action physique perçue*** est un acte capté par un dispositif physique d'entrée ou par un organe sensoriel[5] ;

- une ***action physique perceptible*** est un acte qui n'est pas nécessairement détecté.

En résumé, le niveau d'abstraction d'une tâche élémentaire dépend étroitement des objectifs d'une modélisation donnée. Une tâche élémentaire recouvre un but terminal et une procédure constituée d'actions. Une action est une opération terminale pour un niveau de granularité donné. Au niveau de granularité le plus fin qui soit, une action s'appelle "action physique". Une action physique est un acte unitaire observable par un (ou plusieurs) dispositif(s) artificiel(s) ou naturel(s) et qui provoque une transition d'état de ce(s) dispositif(s).

La granularité des tâches élémentaires est un signe distinctif entre les approches. Les notions de tâche effective et de tâche prévue en sont un autre.

1.3. Tâche prévue et tâche effective

M.F. Barthet introduit les notions de tâche prévue et de tâche effective [Barthet 88]. Une ***tâche effective*** est la trace du plan que l'utilisateur applique en réalité pour atteindre un but donné. Elle représente ***une part de l'activité*** de l'utilisateur car, dans l'absolu, l'activité est l' "ensemble des phénomènes psychiques et physiologiques correspondant aux actes de l'être vivant (ici, l'utilisateur), relevant de la volonté, des tendances, de l'habitude, de l'instinct,

[5] Dans le contexte de l'interaction homme-machine, un dispositif physique d'entrée est un capteur, artefact qui a pour fonction la capture de signaux émis par un utilisateur. Un organe sensoriel désigne, chez l'Homme, un capteur naturel.

etc." [Le Petit Robert]. Ici, l'activité considérée par la tâche effective s'appuie sur les actions observables de l'utilisateur. Elle peut être comparée à la ***tâche prévue***, tâche conçue par celui qui en commande l'exécution (par exemple, le concepteur d'interface).

La comparaison entre tâches prévues et effectives s'appuie nécessairement sur l'existence d'un système de représentation commun : un modèle de tâche.

1.4. Modèle de tâche

Dans le langage usuel, un modèle est "ce qui sert ou doit servir d'objet d'imitation pour faire ou reproduire quelque chose" [Le Petit Robert]. Lorsque nous abordons les modèles de tâche, de même les modèles en psychologie ou en architecture logicielle, ce n'est pas seulement à un objet d'imitation que nous faisons allusion, mais à une référence. En psychologie, cette référence permet de prédire ou d'expliquer un comportement (par exemple avec le modèle explicatif du comportement humain de Rasmussen [Rasmussen 86]). En architecture logicielle, elle permet de dégager ou valider des solutions de mise en œuvre (comme avec le modèle PAC qui structure de manière récursive l'architecture d'un système interactif en agents [Coutaz 90]). En analyse de tâche, un modèle de tâche permet de structurer cette activité d'analyse mais aussi d'en représenter le résultat :

- un modèle de tâche, en tant que ***support à l'analyse***, engage le concepteur à étudier le cas à traiter selon les concepts directeurs de ce modèle. Par exemple, le modèle DIANE suggère au concepteur d'identifier, pour chaque tâche, le responsable de son exécution. Dans UAN, le concepteur est invité à identifier les tâches interruptibles, etc. Nous illustrerons ce point en détail au paragraphe 2 ;

- un modèle de tâche est aussi une ***représentation*** de cette tâche. S'il s'agit d'une tâche prévue, alors le modèle représente la tâche résultant du processus d'analyse de tâche mené par le concepteur. S'il s'agit d'une tâche effective, le modèle est une représentation du plan d'exécution que l'utilisateur a adopté pour atteindre l'objectif de la tâche. Dans tous les cas, une représentation est associée à un formalisme ; et les éléments de ce formalisme traduisent les concepts directeurs du modèle de tâche utilisé comme support d'analyse.

Nous venons de définir la terminologie en liaison avec la notion de tâche. Nous sommes maintenant en mesure d'en étudier l'état de l'art selon des axes d'observation que nous présentons dans la section qui suit.

2. Axes d'observation : finalité et formalisme

Nous avons identifié deux axes d'observation pour l'étude comparative des modèles de tâche : la finalité du modèle et le formalisme utilisé comme support d'expression. Nous caractérisons la finalité d'un modèle de tâche par sa portée dans le cycle de vie des logiciels et nous étudions les formalismes en fonction de leurs éléments directeurs, de leur rigueur sémantique, de leur puissance d'expression ou encore de la lisibilité.

Dans les paragraphes qui suivent nous décrivons les modèles les plus représentatifs en structurant la discussion selon nos deux axes : finalité (paragraphe 3) et formalisme (paragraphe 4). Nous nous attarderons sur les modèles DIANE, JSD*, MAD et UAN et n'exposerons que les grandes lignes de GOMS et de CLG présentés par ailleurs dans la littérature.

3. Modèles de tâche et finalité

Un modèle de tâche peut être utile comme aide à la résolution de tâche au moyen d'un système informatique ou bien il peut servir d'outil dans la conception de système. Dans le premier cas, le modèle s'adresse à l'utilisateur final : il est intégré de manière explicite dans le système. Dans le second cas, le plus fréquent, il a pour client le concepteur.

3.1. Modèle de tâche comme composant du système final

Les modèles de tâche comme composants logiciels du système final participent, pour l'essentiel, à la robustesse de l'interaction. Nous avons retenu deux exemples illustratifs que nous présentons très brièvement : le cas des interfaces langagières et l'aide à l'accomplissement de tâche.

Dans les interfaces langagières fondées sur la langue naturelle, le modèle de tâche sert de support à la définition sémantico-pragmatique, complément indispensable du lexique [Pierrel 87]. Cette représentation explicite permet de résoudre en partie le problème des expressions [6]déictiques, des références [7]anaphoriques et des [8]ellipses. Pour les interfaces avancées et notamment les interfaces multimodales, un modèle de tâche explicite devient incontournable.

---

[6]déictique : qui sert à désigner un objet singulier. Exemple : **cet** outil.

[7]anaphorique : qui reprend un mot ou une phrase antérieurs. Exemple : de l'argent, j'**en** ai.

[8]ellipse : omission syntaxique ou stylistique de un ou plusieurs mots que l'esprit supplée de façon plus ou moins spntanée. Exemple : "**chacun son tour**", pour exprimer "chacun doit agir à son tour"

Cubricon [Neal 88] et MMI2 [Wilson 91] sont deux exemples de tels systèmes incluant un modèle de tâche embarqué.

La seconde utilisation intéressante des modèles de tâches embarqués est l'aide dynamique à la résolution de tâche. Le principe est le suivant : le comportement de l'utilisateur est capté par le système, puis comparé au modèle de tâche prévu (qui est, précisément, le modèle embarqué). En cas de détournement, le système en avertit l'utilisateur et peut proposer une approche (comme dans les didacticiels) ou suggérer de prendre en charge l'accomplissement de la tâche (on parle alors de migration dynamique de tâche) comme dans les systèmes de conduite de processus. Cette technique donne de bons résultats pour les domaines opératifs où la structure de la tâche obéit à un protocole bien défini. Une bonne illustration de ce principe est le système OPAL (Operator Plan Analysis Logic) [Hoshstrasser 89]. OPAL fournit un modèle des intentions de l'usager pour des systèmes complexes. Il a été utilisé pour modéliser les intentions des opérateurs pour une variété de domaines incluant l'aviation militaire, les processus de contrôle ou les combats subaquatiques.

3.2. Modèles de tâche comme outil de conception

En génie logiciel, le processus de conception d'un système répond à des étapes bien identifiées (cf. chapitre I, paragraphe 3). Parmi ces étapes, nous avons retenu celles où le modèle de tâche est pertinent: la définition des besoins (paragraphe 3.2.1), les spécifications externes (paragraphe 3.2.2) et l'évaluation (3.3.3). Nous observons qu'à chacune de ces étapes correspondent des modèles de tâche privilégiés mais nous constatons également que certains modèles couvrent plusieurs phases du cycle de vie (3.3.4).

3.2.1. Modèles de tâche et définition des besoins : DIANE, JSD*, MAD, CLG

La définition des besoins, nous l'avons vu au chapitre I, recouvre l'identification des services attendus du système. A cet effet, les méthodes DIANE ou JSD* peuvent être adoptées. L'une et l'autre incluent une analyse de tâche que concrétise un modèle de tâche exprimé dans un formalisme idoine.

DIANE : une méthode de conception des interfaces utilisateur

> DIANE [Barthet 88] est une méthode de conception d'applications interactives fondée sur Merise [Tardieu 83]. Elle recouvre les phases de spécification et de conception d'IHM (au sens où nous l'avons défini au chapitre I). Comme dans Merise, nous retrouvons dans DIANE les concepts de tâche et d'opération. Mais en prolongement de Merise, DIANE offre au processus d'analyse un

support de réflexion structuré en trois niveaux de représentation : les représentations conceptuelle, externe et interne. La première correspond au point de vue de l'analyste, la seconde définit la perspective de l'utilisateur, la troisième celle du programmeur. Dans le cadre de notre analyse, nous retenons les deux premières :

- ***la représentation conceptuelle*** décrit la logique d'utilisation du système. La logique d'utilisation "fait référence à la représentation mentale que l'utilisateur a de son travail" [Barthet 88, p. 44]. Elle vient compléter la logique de fonctionnement qui décrit les services du système avec ses effets. Un système doit se voir comme un ensemble de postes de travail. Chaque poste remplit une fonction précise caractérisée par l'ensemble des tâches qu'il permet d'accomplir (ou buts qu'il est possible d'atteindre). A chaque but il convient d'associer les procédures que les futurs utilisateurs devront respecter pour l'atteindre. DIANE, comme beaucoup d'autres, utilise une planification hiérarchique : les procédures sont constituées d'opérations décomposables en opérations plus élémentaires.

  Les procédures liées à un but sont multiples. Comme le montre la figure 2, DIANE recommande d'associer à un but une procédure minimale, une ou plusieurs procédures prévues, et éventuellement une ou plusieurs procédures effectives.

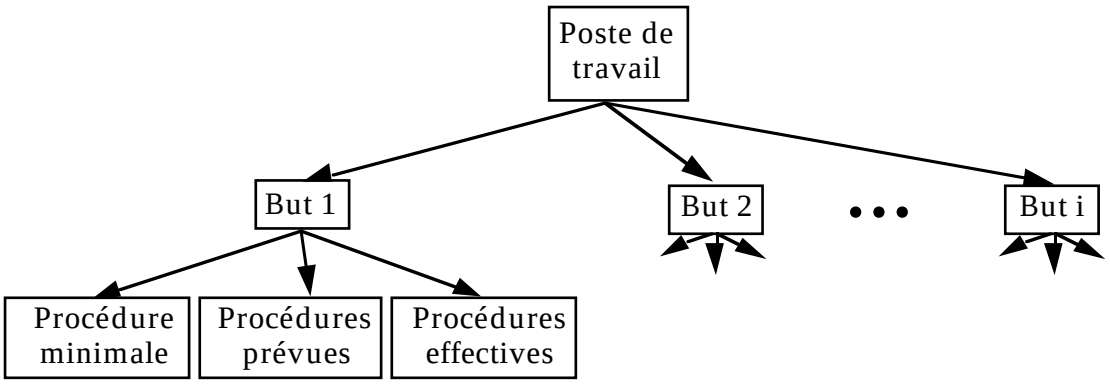


Figure 2 : Buts et procédures dans DIANE.

- la *procédure minimale* représente l'ensemble des opérations et des enchaînements minimaux pour atteindre le but,
- une *procédure prévue* est la description de l'exécution d'une tâche telle qu'elle est prévue dans son déroulement normal ou usuel,
- une *procédure effective* est un modèle observé de l'activité d'accomplissement de la tâche.

Une procédure encapsule un ensemble d'opérations liées par des relations temporelles de précédence. Une opération comporte un nom, un contenu (c'est-à-dire les traitements qu'elle recouvre), un type (obligatoire ou facultatif), un mode de déclenchement (automatique, interactif ou manuel) et un déclencheur (l'utilisateur ou le système). De plus, DIANE isole des fonctions générales offertes à l'utilisateur indépendamment de toute application. Ces fonctions comprennent les aides au travail (interruption, transfert de données, quitter, différer, etc.), les aides à l'apprentissage (guidages sur le fonctionnement et l'utilisation du système) ;

- ***la représentation externe*** rend perceptible à l'utilisateur les éléments de la représentation conceptuelle. Ce rendu doit s'appuyer sur des critères issus de la psychologie cognitive ou de l'ergonomie (structure de la mémoire à court terme, langage d'interaction, temps de réponse, etc.). Elle doit aussi tenir compte des contraintes techniques (caractéristiques physiques de l'écran et logiciels de gestion d'écran). DIANE, en tant que méthode, propose des règles ou des recommandations qui permettent de traduire la représentation conceptuelle en une représentation externe.

L'apport de DIANE tient pour l'essentiel aux concepts véhiculés dans la représentation conceptuelle : planification hiérarchique et ses procédures minimales, prévues et effectives pour traduire la variabilité et la latitude décisionnelle de l'utilisateur, expression de la migration des tâches entre l'utilisateur et le système, etc. Tous ces concepts se voient concrétisés dans un formalisme inspiré de la notation Merise. A l'inverse, la représentation externe est un concept structurant sans formalisme support.

En conséquence, nous retenons que DIANE, en tant que méthode et formalisme, est utile dans la phase d'analyse des besoins. Cependant, il convient de relever DIANE+ qui prolonge la méthode originale par un mécanisme de génération automatique d'interface [Tarby 93].

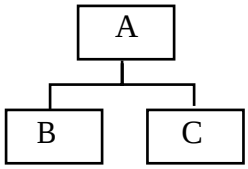
On retrouve dans JSD* [Lim 92] des préoccupations semblables à celle DIANE.

JSD* : Jackson System Development *

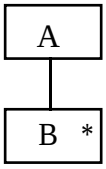
JSD* propose une approche structurée de conception d'interface calquée sur le cycle de développement des logiciels. Alors que DIANE s'appuie sur Merise, JSD* reprend, comme son nom l'indique, la structuration de la méthode

Jackson. Pour DIANE comme pour JSD*, l'objectif est l'intégration des facteurs humains dans le processus de développement.

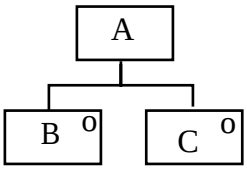
JSD* va cependant plus loin que DIANE dans sa rigueur de développement : à chaque étape de la méthode, les produits de l'analyse de tâche sont décrits sous forme de diagrammes structurés comme ceux de la figure 3 [Lim 93]. Ces représentations arborescentes de buts et sous-buts sont complétées de tables qui expriment de manière informelle en langue naturelle le rôle des buts et leurs relations, mais aussi les décisions et les choix de conception. L'ensemble des modèles produits à chaque étape forme un continuum qui, tout en gardant un contact direct avec la pratique du génie logiciel, relie progressivement l'analyse du problème à une solution possible.



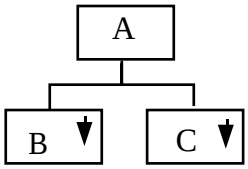
A est une séquence ordonnée de B et C



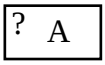
A est une itération de B



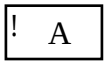
A est une sélection de B ou C



A est une séquence de B et C, séquence pour laquelle l'ordre est indifférent



'Posit'



'Quit'

Le 'Posit' et le 'Quit' sont utilisés pour décrire des événements incertains qui peuvent entraîner l'abandon de la tâche entreprise ('Quit') et la reprise vers une autre tâche ('Posit')

Figure 3 : Constructions de base du formalisme pour l'analyse de tâche dans JSD*.

Comme le montre la figure 4, JSD* structure le processus de conception en trois étapes : l'analyse, la synthèse et la spécification :

- L'***analyse*** commence, comme dans DIANE, par l'acquisition d'informations sur le système existant pour livrer un modèle général de la tâche. Le système existant désigne le système actuel (informatisé ou non) tel qu'il est utilisé mais aussi des systèmes équivalents utilisés dans d'autres organisations.
  - La raison d'être de l'analyse du système existant est multiple : 1) identifier les pratiques, les besoins mais aussi les difficultés rencontrées par l'utilisateur avec le système actuel ; 2) choisir, parmi

ces caractéristiques, celles qu'il faudra intégrer au système cible ; en gros, définir un espace problème qui favorise une approche de conception en largeur (par opposition à une méthode verticale qui engagerait la conception dans une voie unique).

- Le modèle général de la tâche qui résulte de l'analyse de l'existant est une représentation formelle synthétique de l'espace problème. Il permet d'asseoir un vocabulaire commun entre les concepteurs.

• La ***synthèse*** se scinde en trois sous-étapes : 1) la formulation des besoins de l'utilisateur, 2) la création du modèle composite de la tâche et 3) la définition des modèles système et utilisateur de la tâche. Comme le montre la figure 4, l'étape de synthèse s'appuie sur les informations du modèle général produit dans l'étape d'analyse.

- L'expression des besoins résume les conclusions sur l'analyse des systèmes existant et cible. Elle formalise un point de vue centré sur l'utilisateur qui inclut notamment les critères de performance attendues. Cette expression alimente, bien évidemment, le cahier des charges mais aussi la rubrique "utilisabilité" du plan qualité (se reporter au chapitre I).
- Le modèle composite de la tâche est ainsi nommé parce qu'il synthétise en une seule représentation le modèle de la tâche existante et celui de la tâche cible. A ce niveau, le concepteur distingue les tâches "en-ligne" effectuées avec le système de celles qui seront réalisées "hors ligne" par l'utilisateur sans support informatique. On retrouve ici le niveau de description conceptuelle de DIANE et sa notion de déclencheur.
- Les modèles système et utilisateur sont des affinements du modèle composite. Le premier est dérivé des tâches "en ligne" tandis que le second reflète les tâches "hors ligne".

• La ***spécification de l'interface utilisateur*** comprend deux niveaux d'affinement : la génération du modèle d'interaction de la tâche suivie du modèle de l'interface et la définition des écrans.

- Chaque tâche feuille du modèle système de l'étape de synthèse est affinée en un modèle d'interaction. Ce modèle décrit l'interface utilisateur au moyen d'objets et d'actions disponibles, par exemple, dans l'environnement d'accueil. A titre illustratif, pour accomplir la tâche système "éditer un document" dans le Finder du Macintosh, il est possible d'appliquer un double clic sur l'icone représentative du document ou bien commencer par lancer l'éditeur puis utiliser la

commande d'ouverture. Les nœuds du graphe du modèle d'interaction peuvent être décorés d'un identificateur qui désigne un nom d'écran. La description de cet écran fait partie du niveau d'affinement suivant : le modèle de l'interface et la structuration des écrans.

- Le modèle de l'interface et la définition des écrans définissent les détails fins de l'interaction. Les écrans sont représentés au moyen d'images, celles obtenues par exemple au moyen d'un outil de prototypage. Le modèle de l'interface spécifie le comportement en entrée comme en sortie des objets référencés dans le modèle du niveau supérieur. Par exemple, le double clic sur l'icone de l'exemple précédent, conduit à un changement de présentation de l'icone (elle est grisée). Le modèle de l'interface prévoit aussi le libellé des messages d'erreur et leurs conditions d'affichage.

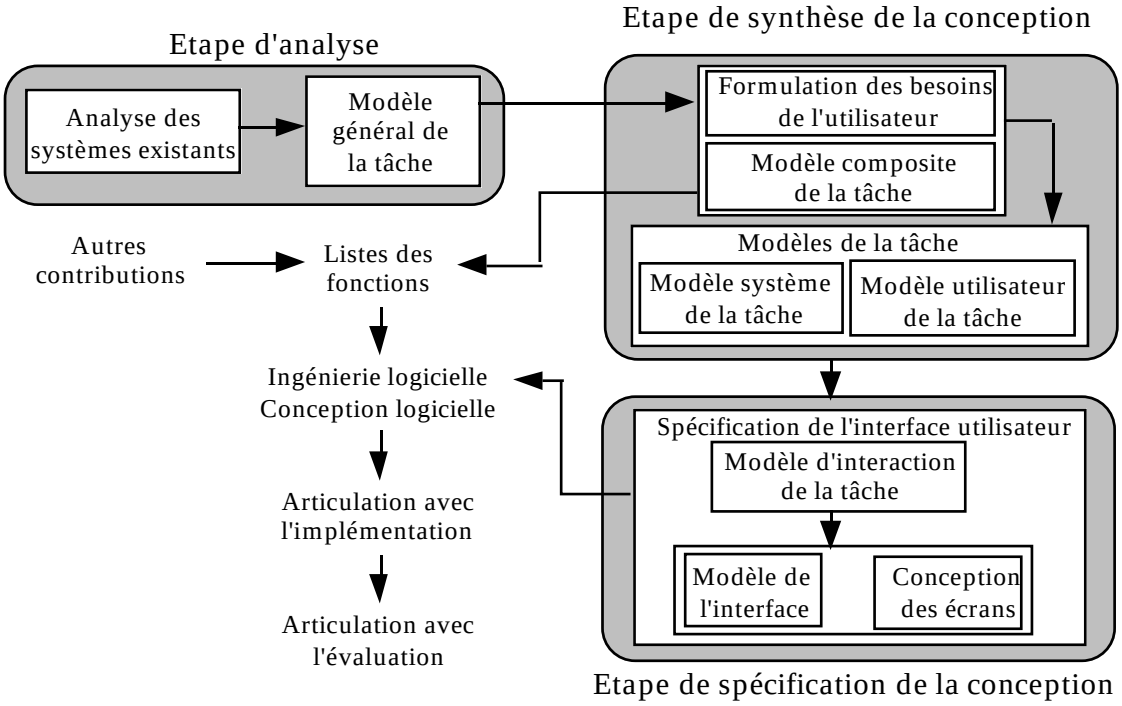


Figure 4 : Diagramme schématique de la méthode JSD*.

En résumé, JSD*, tout comme DIANE, s'appuie sur une décomposition hiérarchique et conceptuelle des tâches à partir d'une étude approfondie de l'existant. Si l'un et l'autre traduisent des motivations semblables (intégrer les facteurs humains dans le processus de développement du Génie Logiciel), JSD* est cependant plus précis et mieux structuré que DIANE dans son processus de construction : JSD* préconise des étapes bien définies avec à la clef une représentation explicite et formelle du produit de la réflexion. JSD* et DIANE se recouvrent jusqu'à la spécification des détails de l'interface utilisateur. Chez DIANE, on l'a vu, la représentation externe n'a pas de support formel.

MAD [Pierret-Goldreich 89, Hammouche 93], que nous présentons maintenant, adopte aussi une décomposition hiérarchique des tâches. Comme pour DIANE, l'accent est mis sur la représentation conceptuelle. La description des détails fins de l'interaction n'est pas couverte explicitement par MAD mais sera, à terme, intégrée dans un générateur automatique d'interface

MAD : une Méthode Analytique de Description de tâches

Dans MAD, comme dans les méthodes vues jusqu'ici, la représentation arborescente exprime des relations logiques de composition mais aussi des liens temporels d'enchaînement. Dans MAD, on trouve :

- la séquence (les sous-tâches doivent être exécutée les unes après les autres),
- le parallélisme (les sous-tâches peuvent être exécutées dans n'importe quel ordre, voire en parallèle),
- la boucle (pour les tâches itératives),
- l'alternative (pour exprimer différentes manières d'exécuter une tâche),
- l'option (pour les tâches non obligatoires).

La figure 5 montre un exemple d'arbre MAD pour une application de messagerie électronique. L'expression MAD indique qu'il est possible d'envoyer en parallèle un ou plusieurs messages ou d'en recevoir. Envoyer un message exige que l'utilisateur définisse le contenu du message avant de signifier au système son désir de l'émettre. Spécifier le contenu demande la saisie du sujet, celle du texte et du destinataire, ces trois tâches pouvant être menées dans un ordre quelconque.

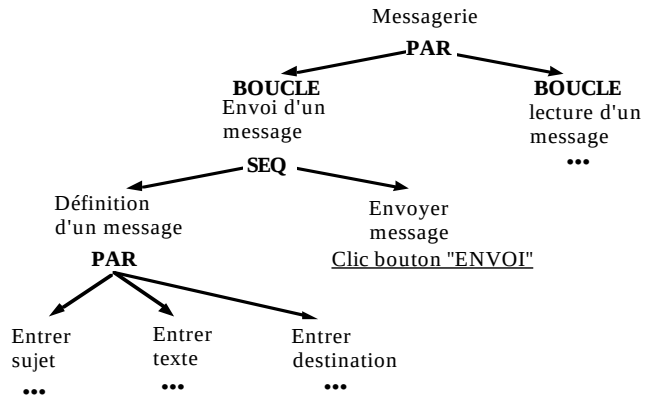


Figure 5 : Extrait de l'arbre logico-temporel pour une tâche de messagerie.

La notation graphique MAD est complétée par un formalisme textuel qui permet de préciser pour chaque tâche, les éléments suivants : l'état initial (sous-ensemble de l'état du monde constitué de la liste des arguments

d'entrée de la tâche), l'état final (sous-ensemble de l'état du monde constitué de la liste des arguments de sortie de la tâche), le but (sous-ensemble de l'état final, indiquant explicitement le but à atteindre par l'exécution de la tâche), les préconditions (ensemble de prédicats qui doivent être satisfaits pour que l'exécution de la tâche soit possible) et les post-conditions (ensemble de prédicats qui doivent être satisfaits après l'exécution de la tâche).

Ainsi, MAD repose sur une notation graphique augmentée d'un formalisme textuel. L'ensemble est véhiculé par des outils logiciels d'édition : éditeur de tâches, d'arbres, d'objets et de schémas.

Le modèle de tâche conduit à une double interprétation selon que l'on adopte la perspective fonctionnel système ou le point de vue cognitif. Il peut représenter le fonctionnement effectif du système, c'est-à-dire l'ensemble des services système et leurs enchaînements ou bien il peut modéliser la connaissance que l'utilisateur devra posséder pour utiliser le système. Cette double interprétation vaut pour CLG [Moran 81].

CLG : Command Language Grammar

CLG permet comme JSD* de structurer l'analyse d'une interface par affinements successifs, du niveau tâche jusqu'au niveau des actions physiques. Cette structuration comprend six niveaux regroupés en trois composantes fonctionnelles : la composante conceptuelle, la composante de communication et la composante physique (cf. figure 6).

Composante conceptuelle	niveau tâche
	niveau sémantique
Composante de communication	niveau syntaxique
	niveau interaction
Composante physique	niveau de la présentation physique
	niveau des unités d'entrée/sortie

Figure 6 : Les niveaux d'une description CLG.

Chaque niveau se projette dans les termes du niveau inférieur visant ainsi à garantir la conformité entre les représentations de niveaux adjacents. On retrouve dans le formalisme CLG l'expression des relations logico-temporelles entre tâches mais aussi des "nouveautés" comme la spécification de valeurs par défaut, ou la présence d'activités spécifiques à l'utilisateur comme "lire l'information X sur l'écran".

En résumé, cette revue des outils d'aide à la conception montre comme élément saillant la décomposition hiérarchique des tâches. Cette hiérarchie traduit l'organisation structurelle de l'espace de travail et précise la nature des concepts manipulés dans le domaine. Le graphe orienté de tâches ainsi obtenu est le plus souvent décoré d'attributs qui traduisent des phénomènes importants observés sur le terrain comme la fréquence d'utilisation, la difficulté cognitive ou l'ordre temporel. Ces attributs conceptuels peuvent servir ensuite à guider la définition des spécifications externes. Par exemple, en se référant au modèle Keystroke [Card 83] il est clair qu'une tâche fréquente doit être présentée de manière directe (non pas suite à une navigation dans un menu en cascade). Inversement, la présentation de tâche complexe pourra être "masquée", obéissant ainsi au principe des "training wheels" de Carroll [Carroll 84].

Pour DIANE et JSD*, le modèle conceptuel des tâches recouvre les tâches accomplies aussi bien par le système que par l'utilisateur. Dans CLG, le concepteur introduit des tâches "cognitives" telle la recherche d'information sur l'écran. De tels systèmes de représentation offrent une vue interactionnelle complète mettant en évidence les responsabilités et besoins des agents communicants intervenant dans l'accomplissement de tâche.

Les modèles de cette catégorie sont, à notre avis essentiels : 1) ils s'inscrivent dans les toutes premières étapes du processus de développement et visent d'emblée l'intégration des facteurs humains dans ce processus ; 2) ils documentent les besoins de manière formelle ; 3) ils représentent l'espace de travail à un haut niveau d'abstraction. Cette représentation conceptuelle dégagée des détails de l'interaction a l'avantage d'être portable mais aussi elle ouvre la voie à la génération automatique d'interface, voire l'évaluation prédictive.

Aux modèles de représentation conceptuelle de la tâche qui concernent essentiellement l'analyse des besoins, il convient d'opposer ceux qui interviennent uniquement au niveau des spécifications externes.

3.2.2. Modèles de tâche et spécifications externes : ETAG, UAN

Les spécifications externes d'un logiciel interactif définissent la vue que l'utilisateur aura du système final. Cette vue doit donner accès aux services définis dans l'analyse des besoins. Les formalismes orientés tâche simple et action tels que ETAG et UAN sont particulièrement indiqués pour cette représentation.

ETAG : Extended Task-Action Grammar

ETAG permet de représenter le savoir que doit posséder un utilisateur (compétent) sur le fonctionnement du système et sur la manière d'exécuter des tâches élémentaires [Tauber 90]. Les auteurs font l'hypothèse que la structuration hiérarchique des tâches composées est disponible par ailleurs. ETAG s'inspire de CLG, de TAG [Payne 86] mais aussi des travaux de Reisner sur les grammaires d'action [Reisner 81].

Le formalisme d'ETAG s'articule autour de trois composants : l'User's Virtual Machine (UVM) qui décrit la sémantique de chaque tâche, un dictionnaire des tâches, et les règles de production qui définissent la grammaire du langage de commande. Nous illustrons brièvement ces trois composants avec l'exemple de la messagerie électronique. Cet exemple inclut l'événement conceptuel MARQUER_MESSAGE_LU (figure 7), la tâche élémentaire de marquage d'un message comme étant "lu" (figure 8) et quelques règles de production associées à l'événement MARQUER_MESSAGE_LU (figure 9).

Les concepts sémantiques sont notés entre [], un > sépare le type du concept de son nom, les relations conceptuelles sont notées entre <> et les concepts syntaxiques entre %%. {*x} désigne un ensemble d'x. Les commentaires sont placés derrière des /* et les mots clés sont en minuscules.

- L'UVM spécifie la sémantique de chaque tâche au moyen d'un cadre précis ("frame"). Cette structure décrit les concepts impliqués dans la tâche : le type, les attributs avec leur domaine de valeur, les opérateurs applicables aux objets assortis de pré- et de postconditions, des relations de localisation (conceptuelle) entre objets (par exemple, un message "est dans" une boîte aux lettres).

  Le modèle, qui se veut orienté objet, n'adopte pas toujours la terminologie qui convient. En particulier, les opérations sont appelées des "événements".

```
type [event > MARQUER_MESSAGE_LU]
   description :
         for [MESSAGE ]                                   /* pour un MESSAGE
           /* positionner l'attribut <MARQUE_LU> à "lu"
           [event.set_val([MESSAGE], <MARQUE_LU>, "lu")];
   comments :     "marquer le message à lu";
end [MARQUER_MESSAGE_LU].
```

Figure 7 : Définition UVM de l'événement conceptuel pour marquer un MESSAGE à "lu".

- Le dictionnaire des tâches élémentaires du système énumère les tâches, les actions et les événements de base disponibles.

```
entry 9:
   [task > MARQUER_MESSAGE_LU],          /* tâche de base
   [event > MARQUER_MESSAGE_LU],         /* événement de base en rapport
   T9      /* symbole associé à la tâche

           /* concepts contrôlés par l'utilisateur
           [event > MARQUER_MESSAGE_LU] [object > MESSAGE: {*x}],

   /*commentaire
   "marque un  message comme lu".
```

Figure 8 : Dictionnaire des tâches élémentaires : marquage d'un MESSAGE à "lu".

- Les règles de production décrivent la grammaire du système à plusieurs niveaux d'abstraction : spécification, dialogue, lexique et action ("keystroke"). Elles définissent la correspondance entre une tâche et les actions physiques nécessaires à son accomplissement.

```
Niveau spécification :
/* détermine la séquence de spécification des composants conceptuels pour la tâche référencée en
   partie gauche
T9 [event > MARQUER_MESSAGE_LU] [object > MESSAGE: {*x}] ::= specify [object] +
                                                              specify [event]

Niveau dialogue :
/* en partie gauche : composant conceptuel à spécifier, en partie droite : style de référence à ce
   composant.
specify <ATTRIBUT> [VALUE] ::= select 'symbol <ATTRIBUTE> [VALUE]'
                    /* où 'symbol <ATTRIBUTE> [VALUE]' est défini au niveau lexical

Niveau lexical :
symbol <MARQUAGE_de_LECTURE>  [VALUE] ::=
                                  [%DIALOG_BOX% : "lecture"] [%BUTTON% : "lu"]

Niveau keystroke :
/* indique que l'on doit cliquer avec le bouton gauche de la souris sur le bouton %IMAGE%
select [%IMAGE%]::= mouse-position [%IMAGE%] + click-left-button
```

Figure 9 : Règles de production.

En résumé ETAG considère des tâches élémentaires. Ces tâches correspondent aux feuilles d'une modélisation conceptuelle façon DIANE ou JSD*. Le niveau UVM d'ETAG définit le point de jonction avec une modélisation conceptuelle de tâche. Les règles de production offrent un moyen de formaliser la spécification de l'interface utilisateur. Contrairement à JSD*, ETAG ne considère que les actions utilisateur. L'expression du comportement du système en sortie est passé sous silence. UAN tend à combler cette lacune [Hartson 92].

UAN : User Action Notation

Le formalisme UAN s'appuie sur une représentation tabulaire à trois colonnes. Une tâche comprend trois facettes et chaque facette occupe une colonne. Comme le montre l'exemple de la figure 10, la colonne de gauche spécifie la suite temporelle des actions physiques que l'utilisateur doit effectuer pour accomplir la tâche. La colonne centrale décrit, en regard des actions utilisateur, les retours d'information fournis par le système. La colonne de droite sert à noter le changement d'état de variables pertinentes pour l'interface.

Dans l'exemple de la figure 10, la tâche de sélection d'un objet est notée Sélectionner(objet) : "Sélectionner" est le nom de la tâche et "objet" en désigne le paramètre. Pour accomplir cette tâche, l'utilisateur doit déplacer le curseur de la souris sur la représentation iconique de l'objet (~[icone(objet)]) puis enfoncer le bouton de la souris (Mv). Alors, le système réagit en mettant en évidence l'icone de l'objet (icone(objet)!) et, s'il existe un objet objet' dont l'icone est en évidence (∀ icone(objet')!:), alors celle-ci est éteinte (icone(objet')-!). Lorsque l'utilisateur relâche la souris (M∧), le système ne fournit pas de retour d'information mais note dans une variable d'état (ObjetSélectionné) que "objet" est maintenant sélectionné.

Tâche : Sélectionner(objet)		
Action de l'utilisateur	Retour de l'interface	Etat de l'interface
~[icone(objet)] Mv	icone(objet)!, ∀ icone(objet')!: icone(objet')-!	
M∧		ObjetSélectionné = objet

Figure 10 : Description UAN de la sélection d'un objet.

La notation dont on trouvera une description détaillée dans [Hartson 92, p. 13] permet de représenter les actions de type manipulation directe. Le symbole ~ de la figure 10 est un premier exemple. L'expression des formes élastiques et l'asservissement de fantômes aux déplacements de la souris, typiques de la manipulation directe, sont également prévus dans le langage. Pour notre part, nous avons étendu la notation pour permettre l'expression d'interfaces multimodales [Coutaz 93a].

Nous venons de présenter UAN comme un outil de spécification de l'interaction au niveau des tâches élémentaires. S'il est vrai qu'à l'origine, UAN était centré sur les actions physiques utilisateur et système, il en va différemment dans sa forme actuelle. De fait, les tâches UAN peuvent être

élémentaires ou composées. Par extension, la colonne de gauche d'une description UAN de tâche peut comprendre des actions physiques comme celles de la figure 10 et/ou des noms de tâche. Dans les deux cas, actions et tâches sont liés par des relations temporelles. Le tableau de la figure 11 présente les différentes possibilités.

Relation temporelle	**Symbole UAN**
Séquence	A B
Attente	A (t>n) B
Disjonction répétée	(A | B)*
Indépendance d'ordre	A & B
Entrelacement monodirectionnel	A → B
Entrelacement bidirectionnel	A ↔ B
Parallélisme	A || B

Figure 11 : Résumé des relations temporelles dans UAN.

- La relation de séquencement temporelle est forte. *A B* signifie que *B* ne peut être entreprise que si *A* est terminée.
- L'attente permet d'exprimer des contraintes de temporisation : dans *A (t>n) B*, l'exécution de B n'a lieu que si A est terminée depuis un temps t>n.
- La disjonction *A | B* signifie que l'utilisateur a le choix entre l'exécution de A ou de B.
- Dans *A & B*, l'utilisateur doit exécuter A et B mais dans l'ordre qui lui convient.
- L'expression des entrelacements mono- et bidirectionnel entre tâches traduit les dialogues à plusieurs fils. Dans *A → B*, l'exécution de la tâche A peut être interrompue par B, puis reprendre une fois B achevée. Dans *A ↔ B*, A et B peuvent s'interrompre mutuellement.
- *A || B* exprime l'exécution simultanée des deux tâches A et B.

Par exemple, dans Eudora, la tâche de gestion des messages permet l'exécution entrelacée des tâches d'envoi et de lecture de messages et ces tâches peuvent être exécutées un nombre de fois quelconque (notation *). La figure 12 donne une représentation UAN de cette utilisation.

Tâche : Gestion_messagerie_Eudora
(envoi_d'un_message ↔ lecture_d'un_message)*

Figure 12 : Description UAN de la tâche Gestion_messagerie_Eudora.

Chacune des sous-tâches (ici, envoi_d'un_message et lecture_d'un_message) peuvent être décrites par décomposition successive jusqu'au niveau des actions physiques.

En dépit des limites que nous évoquerons plus loin, les formalismes orientés action ont une double utilité : comme outils de spécification et comme éléments d'analyse prédictive.

- En tant qu'outil de spécification, ils fournissent un complément précieux aux spécifications externes informelles. Cet argument tient pour UAN dont la motivation, à l'origine, est de servir d'outil de communication entre le concepteur-ergonome et le concepteur de logiciel.

- L'analyse prédictive à partir de modèles de tâche simple concerne nécessairement l'épiderme de l'interface. Nous approfondissons ce point au paragraphe suivant.

3.2.3. Modèles de tâche et évaluation : GOMS, CCT, ETIT, spécifications externes

Pour ce qui est de l'évaluation prédictive, GOMS et ses dérivés tels la Théorie de la Complexité Cognitive (théorie développée dans [Kieras 85] et sur laquelle nous reviendrons au chapitre III) ou ETIT offrent des métriques, certes réductrices mais utiles si l'on considère qu'elles permettent de détecter les erreurs de conception les plus grossières ou d'effectuer des analyses comparatives et donc de guider des choix de conception avant même la mise en œuvre, réputée coûteuse, du prototype.

GOMS : Goal, Operator, Method, Selection

GOMS est un modèle de description du comportement attendu d'un utilisateur expert. Un modèle GOMS peut se pratiquer à quatre niveaux d'abstraction en fonction des objectifs recherchés par le concepteur : tâche, fonction, argument et actions physiques [Card 83].

Les ingrédients de GOMS sont les buts (Goal), les opérateurs (Operator), les méthodes (Method) et les sélections (Selection).

- ***un but*** définit un état recherché. Il est réalisé par l'exécution d'une suite d'opérateurs et est décomposé de manière hiérarchique ;
- ***un opérateur*** est une action atomique dans le niveau de modélisation considéré. Son exécution provoque un changement d'état du modèle mental de l'usager mais aussi de l'environnement et notamment du

système. Tous les opérateurs d'un niveau de modélisation GOMS ont une durée d'exécution de même ordre de grandeur. Cette métrique est une façon de définir le niveau d'abstraction d'un modèle GOMS.

- ***une méthode*** décrit le procédé qui permet d'atteindre un but. Elle représente le savoir-faire et s'exprime par composition de sous-buts et d'opérateurs.
- ***une règle de sélection*** est utilisée lorsque plusieurs méthodes permettent d'atteindre un but donné. Elle offre le moyen de choisir une méthode parmi les candidates possibles.

GOMS s'utilise ainsi : ayant une décomposition de buts et de sous-buts, GOMS permet de prédire le temps mis par l'utilisateur (expert) pour atteindre un but donné. Cette prédiction s'obtient simplement en additionnant les temps d'exécution des opérateurs impliqués pour atteindre le but. La valeur obtenue peut être comparée, par exemple, avec celle des seuils fixés dans le plan qualité. La méthode peut également servir à guider un choix de conception : il suffit de pratiquer une évaluation comparative des performances attendues pour chaque solution envisageable.

On a souvent reproché à GOMS son caractère réducteur et son applicabilité au seul niveau "keystroke", là où les opérateurs liés à l'activité mentale sont plus simples à déterminer. Cette simplicité est aussi un avantage pour le concepteur d'interface qui n'est pas nécessairement un spécialiste en ergonomie ou en psychologie. Il est vrai que le GOMS original ne traite pas le cas des erreurs. On trouvera dans [Lerch 89] une étude qui montre comment on peut prédire la fréquence des erreurs dues aux surcharges cognitives.

Même si la portée pratique de GOMS relève du niveau keystroke, ce modèle offre des éléments analytiques qui permettent de justifier simplement des choix entre plusieurs alternatives syntaxiques et lexicales. Ces justifications, fondées sur une prédiction analytique et non pas sur la seule intuition heuristique, peuvent alors être documentées dans des tables façon JSD* ou encore dans un schéma de décision de type QOC (voir chapitre I, figure 6) : dans un projet logiciel, on n'insiste jamais assez sur l'importance de la traçabilité des décisions.

Toujours dans l'optique d'offrir des moyens de comparaison de différentes interfaces, ETIT repose sur la distinction entre tâche externe et tâche interne [Moran 83].

ETIT : Mapping from External Task space to Internal Task space

ETIT consiste à reformuler les tâches du modèle externe (celui de l'utilisateur) vers le modèle interne (celui de la machine). Un espace de tâches est représenté par un ensemble de termes (concepts) ainsi qu'un ensemble de tâches décrites dans ces termes. L'espace des tâches externes représente l'ensemble des tâches possibles que l'utilisateur voudrait faire avec le système. L'espace des tâches internes représente l'ensemble des tâches possibles à l'intérieur du système. Les règles de correspondance expriment pour chaque tâche externe, sa réalisation à l'aide des tâches internes disponibles. Un exemple de tâche pour un éditeur de texte est le suivant :

- tâche externe : ajouter du texte
- tâche interne : insérer une chaîne
- Règles de correspondance : ajouter --> insérer
  texte --> chaîne de caractères

L'observation de la correspondance construite entre tâches externes et tâches internes va permettre des évaluations de trois ordres :

- évaluer la complexité de la relation d'un système avec l'espace de tâches externes cible en examinant les règles de correspondances,
- évaluer les répercussions d'un élément de savoir manquant sur les tâches réalisables par l'utilisateur,
- évaluer le transfert de savoir nécessaire entre plusieurs systèmes en comparant les règles de correspondances mises en œuvre pour chacun d'eux.

Spécifications externes et l'évaluation prédictive

L'analyse prédictive à partir de modèles de tâche simple concerne nécessairement l'épiderme de l'interface. Reisner avec sa grammaire d'action [Reisner 81] fut l'une des premières à évaluer l'utilisabilité et la facilité d'apprentissage de la surface de l'interface à partir de métriques sur la grammaire comme le nombre de règles.

Nos premières expériences avec UAN nous ont permis de dégager quelques heuristiques de tests d'utilisabilité fondés sur des principes et critères simples comme l'observabilité (voir chapitre I et [Abowd 92]. Nous citons ici quelques unes de ces règles applicables de manière systématique à une spécification UAN [Coutaz 93a] :

- si, dans la spécification UAN d'une tâche, les préconditions de déclenchement de cette tâche ne peuvent s'exprimer en terme d'éléments

perceptibles par l'utilisateur, alors le principe d'observabilité est transgressé : l'utilisateur n'a pas les moyens de savoir que l'état du système lui permet d'accomplir cette tâche ;

- si, dans la spécification UAN d'une tâche, une action utilisateur n'est pas suivie d'une réaction perceptible du système, le principe de retour d'information immédiat n'est pas observé ;

- si, dans la spécification UAN d'une ou plusieurs tâches, pour deux suites identiques d'actions utilisateur, le retour d'information système est distinct, alors le comportement du système peut être perçu comme incohérent (typiquement, dans un cas, le feedback est fourni sur l'enfoncement du bouton de la souris ; dans l'autre, sur le relâchement) ;

- si, selon les tâches, un même objet est affiché par le système en des endroits distincts de l'écran, le principe de stabilité n'est pas observé.

Nous venons d'esquisser comment il est possible d'effectuer une évaluation prédictive à partir d'une spécification formelle de l'interface. Même si nos suggestions sont embryonnaires, elles nous semblent utiles à peu de frais.

Nous avons jusqu'ici étudié des modèles de tâches en relation avec l'espace IHM du processus de développement des logiciels (voir figure 13 du chapitre I) : certains sont utiles à l'expression de l'analyse des besoins (DIANE, MAD), d'autres à l'expression des spécifications externes (ETAG, UAN), d'autres encore recouvrent les deux étapes (JSD*). Certains ne visent pas uniquement la spécification mais aussi l'évaluation prédictive. Il convient maintenant d'étudier les modèles qui pénètrent dans l'espace logiciel.

3.2.4. Un pont entre les différentes phases du cycle de vie d'un logiciel : SIROCO, ADEPT

Pour répondre à cette question, étudions les objectifs que se donnent chacune des méthodes auxquelles nous nous sommes intéressées. Pour JSD* et DIANE l'objectif est clair : intégrer une approche ergonomique dans le développement des logiciels. Ces méthodes préconisent ainsi une modélisation qui s'appuie sur une analyse du travail centrée sur l'utilisateur [Norman 86].

Pour MAD, comme pour UAN, CLG ou ETAG, l'objectif est plus flou. Il s'agit là de structurer l'analyse de tâche, dans le but plus ou moins exprimé de faire communiquer les différents intervenants au sein du cycle de vie des logiciels

(nous pensons ici explicitement à UAN), voire tendre vers une conception automatique (cela sera vrai pour MAD ou ETAG). Pour ETAG, la construction du pont entre le formalisme de description de la tâche et la réalisation de l'interface Homme-Machine reste à mettre en place. A l'heure actuelle, certaines questions non encore résolues sont un frein à cette évolution. Par exemple, nous n'avons pas trouvé de formalisation des relations temporelles entre les tâches.

L'objectif pour SIROCO et ADEPT (que nous présentons ci-dessous) est de fournir un outil de prototypage rapide, et donc de conception automatique des interfaces utilisateur. ADEPT s'appuie sur une formalisation TKS de l'espace des tâches et sur un modèle de l'utilisateur [Johnson 91]. Il est capable de produire automatiquement une interface virtuelle projetable ensuite dans les termes d'une boîte à outils réelle telle que Motif. MAD vise un objectif similaire. SIROCO réalise explicitement la dernière arche du pont. Il n'inclut pas de modèle de tâche explicite mais suggère au concepteur de refléter sous forme d'espaces de travail la décomposition hiérarchique qui résulte de l'analyse de tâche [Normand 92a].

SIROCO : un langage de spécification conceptuelle

SIROCO est un outil d'aide au développement des interfaces utilisateur. Il consiste en un langage de spécification conceptuelle d'une interface utilisateur doublé d'un générateur de code qui implémente cette interface dans un environnement de programmation à objets : Guide [Krakoviak 90]. Le langage de spécification SIROCO repose sur une représentation conceptuelle d'une interface orientée objet et distingue, dans un système interactif, les dimensions "fonctionnement" et "utilisation". Dans ce qui suit, nous nous intéressons uniquement à cette représentation conceptuelle de la tâche sans développer la méthode utilisée pour la génération automatique d'interface. Les systèmes interactifs visés sont des applications axées sur les données plutôt que sur les traitements, typiquement des applications de bureautique. L'interaction repose sur les entrées clavier et la désignation directe d'éléments par la souris.

Les éléments de base du modèle sont de trois types : les concepts de l'application (ou noyau fonctionnel), qui expriment la dimension de fonctionnement de l'application, ainsi que les perspectives qui, avec les espaces de travail, expriment la dimension d'utilisation de l'application.

- ***Les concepts de l'application*** : la conception d'un système nécessite tout d'abord de définir les entités du domaine ainsi que l'objectif de ce

système, c'est-à-dire ce que l'utilisateur va pouvoir effectuer avec le système. L'ensemble des objectifs à réaliser par l'utilisateur à l'aide du système informatique s'organise, informellement, en un arbre de tâches (cf. figure 13).

- ***Les perspectives et espaces de travail*** : il s'agit alors d'organiser les différentes tâches de l'arbre en espaces de travail (espaces qui définissent la dimension d'utilisation), et de définir les perspectives sur les concepts manipulés dans ces espaces de travail. Il n'existe pas de règles pour cette mise en correspondance mais quelques indications (au nombre de 7) ([Normand 92a, p. 132]). Ces recommandations portent essentiellement sur la conduite à respecter pour décomposer la tâche (par exemple : "Parmi les tâches strictement séquentielles, isoler les tâches effectivement distinctes").

Un exemple de définition d'espace de travail à partir de l'arbre des tâches est présenté dans la figure 13 : de l'arbre des tâches de gestion d'une messagerie, deux espaces de travail (WorkSpaces) "WS_Envoi" et "WS_Lecture" ont été identifiés. Parmi les concepts impliqués dans ces tâches, nous relevons celui de message. (Pour une description détaillée de ces concepts se référer à [Normand 92a, pp. 124-142].)

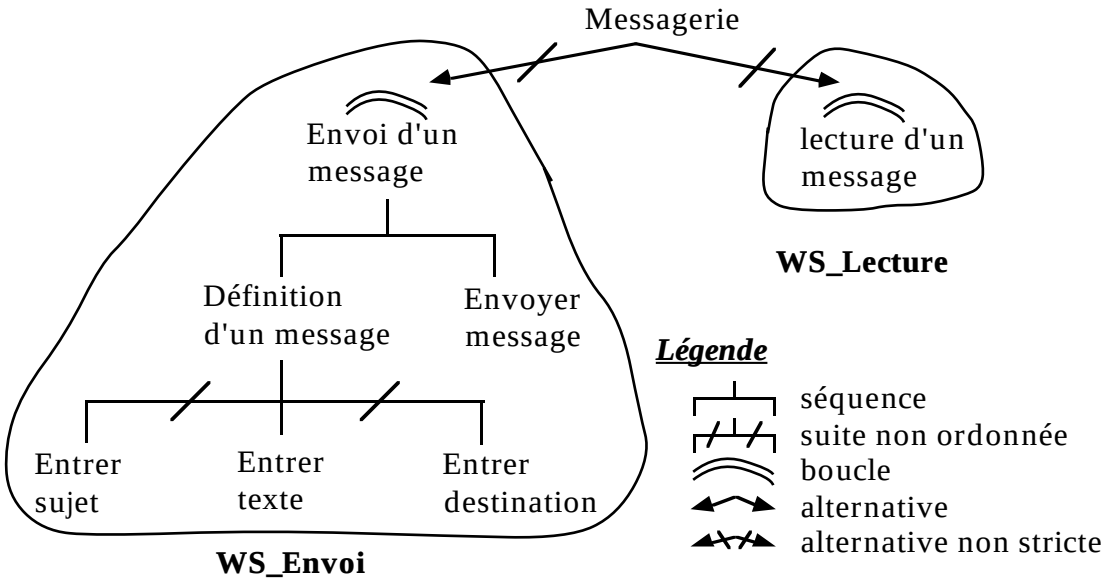


Figure 13 : Arbre de tâche et extraction d'espaces de travail dans SIROCO.

Le formalisme utilisé pour décrire l'arbre des tâches est purement indicatif. La construction de l'arbre des tâches ne fait pas partie intégrante de SIROCO. L'arbre des tâches sert de "brouillon" pour mettre au clair la structuration des dimensions d'utilisation de l'interface. Nous donnons ci-dessous (cf. figure

14) un bref extrait de la spécification SIROCO pour une messagerie. L'extrait décrit la spécification de l'espace de travail WS_Envoi de la figure 13. Cet espace permet la définition et l'envoi d'un message quelconque. Les concepts mis en jeu se réduisent à l'objet EnvoiMess représentant le message à envoyer.

```
WORKSPACE WS_Envoi IS
   _name : "écriture d'un message"
   _instances : MULTIPLE;                          / / prop. d'instanciation multiple

         / / objet Concept accessible dans l'Espace : le message à envoyer, donnée locale
   LOCAL mess : EnvoiMess
                  / / perspective utilisée, définie localement
         WITH PERSPECTIVE EnvoiMess_2 IS
               _kind : PROTOTYPE;                  / / prop. de nature “prototype”
               _input_validation_mode : IMPLICIT   / / prop. de mode de validation

                  / / attributs retenus :
               ATTRIBUTE subject
               ATTRIBUTE tex

                  / / opérations retenues:
               OPERATION Post
                        / / pré-actions de l'opération
                     PRE_ACTION NEW_INSTANCES SCAN_INPUT END
                        / / post-actions de l'opération
                     ON SUCCES DO RESET_PROTO END
         END EnvoiMess_2      / / fin de la définition de la perspective
END WS_Envoi.
```

Figure 14 : Spécification SIROCO de l'espace de travail WS_Envoi.

“L'espace WS_Envoi est destiné à d'éventuelles instanciations multiples (propriétés _instances). WS_Envoi met en jeu une donnée locale décrite comme un prototype de message à envoyer (propriété _kind de la perspective utilisée), sur lequel est portée une perspective permettant la modification des champs du message. Ce prototype permet la récupération de données qui seront utilisées lors de l'instanciation véritable de l'objet, en vue de l'initialisation de ce dernier. La validation des données entrées est implicite (propriété _input_validation_mode) ; l'instanciation du message s'effectue lors de l'activation de l'opération Post (valeur de PRE_ACTION). En cas de succès de l'opération, la donnée prototype est réinitialisée (ON SUCCES DO...) à une valeur vide de façon à permettre la composition et l'envoi d'un nouveau message.” [Normand 92a, p. 141].

En résumé, SIROCO génère l'interface utilisateur à partir d'une représentation conceptuelle de cette interface. Cet outil vient en prolongement d’une spécification MAD. L’union des services offerts par MAD (dans sa version

actuelle), et par SIROCO se retrouve dans ADEPT : ADEPT guide le concepteur depuis la construction du modèle de tâche jusqu'à la génération automatique d'un prototype d'interface Homme-Machine [Johnson 91].

ADEPT : Advanced Design Environment for Prototyping with Task models

ADEPT est le nom donné à un environnement pour la génération rapide de prototypes d'interfaces utilisateur. Cette génération nécessite la conception de plusieurs modèles : Le modèle de la tâche, le modèle de l'utilisateur et les modèles de l'interface (cf. figure 15).

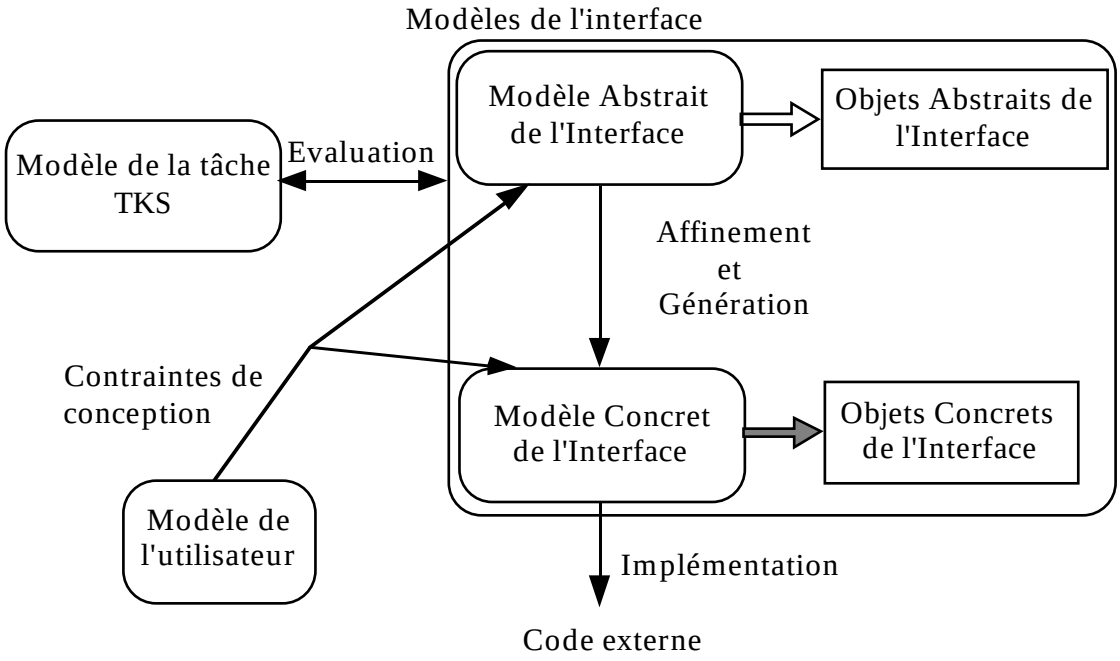
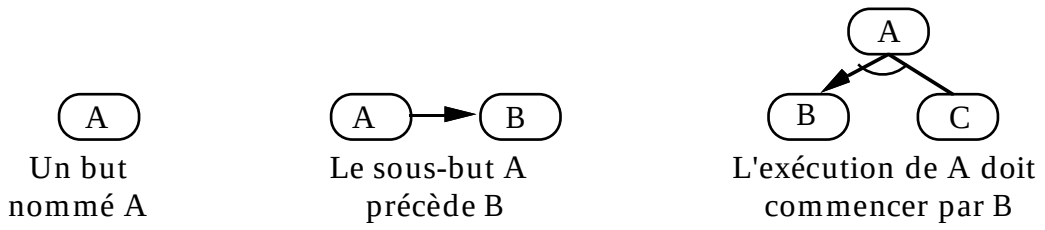
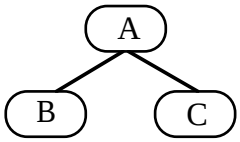


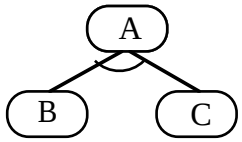
Figure 15 : Les modèles utilisés dans ADEPT.

- ***Le modèle de la tâche*** s'appuie sur le concept de Task Knowledge Structure (TKS). Une TKS est un modèle cognitif de la tâche. Elle fournit un cadre théorique pour la modélisation du savoir requis pour exécuter une tâche. Elle se présente comme un ensemble structuré d'objets, d'actions, de procédures et de buts. Plusieurs formalismes sont utilisés pour représenter une TKS. Le formalisme graphique dont la figure 16 donne un extrait est une première possibilité [Johnson 92].





B et C sont des sous-buts
de A et représentent des alternatives
pour l'exécution de A



B et C sont des sous-buts
de A. B ET C doivent
être exécutés

Figure 16 : Structures pour la représentation graphique d'une TKS.

- ***Les modèles de l'interface*** sont conçus à deux niveaux : le modèle abstrait de l'interface (AIM : Abstract Interface Model) et le modèle concret de l'interface (CIM : Concrete Interface Model) (cf. figure 15, cadre "Modèles de l'interface"). Le modèle abstrait de l'interface reflète (cf. flèche blanche) la structure de l'interaction en terme d'objets abstraits de l'interface utilisateur (AUIO : Abstract User Interface Object). Le modèle concret de l'interface définit (cf. flèche grise) les objets concrets de l'interface (CIO : Concrete Interface Object) utilisés pour matérialiser les objets abstraits de l'interface utilisateur. Par exemple, un objet abstrait de l'interface utilisateur est un menu, alors que l'objet concret de l'interface qui lui correspond pourrait être un menu déroulant ou une barre de menus. Les seconds se déduisent des premiers via un ensemble de règles de correspondances prédéfinies. Tout comme les TKS, l'AIM s'exprime sous forme graphique à l'aide des schémas de la figure 15.

- ***Le modèle de l'utilisateur*** reflète les caractéristiques de l'utilisateur. Nous ne disposons malheureusement pas plus d'informations à ce sujet.

Le principe de la génération du prototype d'interface repose sur une comparaison des spécifications des TKS et de l'AIM conjuguées. Ces descriptions sont reformulées, de manière textuelle, sous le format défini par CSP (Communicating Sequential Processes) [Hoare 85]. CSP permet de décrire le comportement d'un système de processus communicants. Ce format a été étendu afin de représenter de manière textuelle le dialogue décrit par les schémas de la figure 15.

L'environnement ADEPT fournit des outils pour la construction :

- du modèle de l'utilisateur,
- du modèle de la tâche,
- du modèle abstrait de l'interface,
- du modèle concret de l'interface,
- de l'implémentation graphique associée.

Les efforts réalisés par SIROCO ou ADEPT, nous apportent des éléments de réponse à la réalisation d'un pont qui recouvre les différentes phases du cycle de développement des logiciels. Cette voie est à poursuivre en tenant compte, par exemple, des techniques nouvelles de l'interaction multimodale [Coutaz 93b].

Dans cette section 3, nous avons étudié des modèles de tâche en fonction de leur finalité. Nous considérons maintenant notre deuxième axe d'observation : le formalisme d'expression.

4. Modèles de tâche et formalisme

Le formalisme choisi pour représenter la tâche dépend fortement de l'objectif visé [Sébilotte 91]. Pour mettre en évidence une composante psychologique dans l'exécution de la tâche, un formalisme procédural, des réseaux sémantiques [Collins 69] ou des règles de production ou de décision [Richard 90] seront choisis. Pour concevoir une application informatique, les différentes méthodes d'analyse de tâche offrent de nombreux formalismes. Ces formalismes vont des grammaires (CLG ou ETAG), aux réseaux de Pétri (AMME [Rauterberg 92], [Palanque 92]), en passant par des techniques orientées objets (SIROCO), des représentations graphiques (JSD* ou MAD) ainsi que des langages tel CSP (ADEPT).

Les cinq points d'intérêt que nous considérons sur cet axe sont : les éléments directeurs, la rigueur sémantique, la puissance d'expression, la lisibilité et la facilité d'utilisation.

4.1. Eléments directeurs : fonctions et concepts

Le formalisme utilisé pour l'expression d'un modèle reflète l'approche dans l'analyse du problème. Nous avons identifié deux éléments directeurs : l'analyse par la fonction et l'analyse centrée sur les concepts du domaine.

Dans TKS, MAD, UAN, GOMS et CLG, l'objet central de la description est la tâche : le formalisme support suppose une analyse fonctionnelle du problème. Dans d'autres, comme SIROCO et ETAG, les objets directeurs sont les concepts du domaine : le formalisme support induit alors une approche par objets sur lesquels sont appliquées des opérations perçues comme des tâches.

4.2. Rigueur sémantique

La rigueur sémantique est une propriété appréciée : absence d'ambiguïtés, propriétés démontrables et développement d'outils automatiques.

Les formalismes de modélisation de tâche souffrent généralement d'un manque de rigueur. CLG est un exemple notoire. Notre expérience avec UAN nous a permis d'identifier plusieurs problèmes et notamment l'ambiguïté des relations temporelles entre les instructions situées dans des colonnes distinctes. Prenons l'exemple de la figure 10 avec l'hypothèse que l'exécution des instructions se produit de gauche à droite à travers les trois colonnes puis de haut en bas. Dans ces conditions, la variable d'état ObjetSélectionné est modifiée après que l'utilisateur ait relâché le bouton de la souris. Que faudrait-il écrire pour exprimer que cette variable soit modifiée en un instant quelconque entre le début et la fin d'exécution de la tâche? XUAN (Extended UAN) vise à combler ce vide sémantique [Gray 94].

SIROCO et TKS, qui conduisent à la génération automatique d'interfaces, offrent au contraire une sémantique rigoureuse mais leur puissance d'expression est-elle suffisante ?

4.3. Puissance d'expression

La puissance d'expression est une troisième propriété intéressante puisqu'elle détermine la portée descriptive du modèle. Pour les formalismes de modèles de tâche, nous relevons les niveaux de description, l'expression des relations entre tâches, la description des relations entre les expressions d'entrée formulées par l'utilisateur et les expressions de sortie émanant du système.

4.3.1. Niveaux de description

Selon les formalismes, les niveaux de description ont un usage différent. Dans CLG, les niveaux tâche, sémantique, syntaxique, et interaction servent de pont entre les domaines de conception. En particulier, le niveau tâche représente l'utilité du système dans le domaine considéré tandis que le niveau sémantique enrichit le modèle du niveau tâche avec des concepts dirigés par le système. Le niveau tâche définit les besoins et le niveau sémantique sert de relais vers la spécification logicielle détaillée. Dans GOMS, un niveau correspond davantage à un niveau d'abstraction : les opérations d'un niveau donné présentent la même granularité c'est-à-dire une durée d'exécution de même ordre de grandeur. Dans UAN ou MAD, l'affinement décrit essentiellement les plans d'actions possibles.

4.3.2. Relations entre tâches/Attributs des tâches

Les formalismes qui expriment la décomposition hiérarchique de tâches incluent tous la possibilité d'exprimer le séquencement (tâche A puis tâche B), l'alternative (tâche A ou bien tâche B), la composition (tâche A et tâche B dans

un ordre quelconque), l'itération, ou encore l'exécution concurrente de plusieurs tâches (tâche A en même temps que tâche B). Toutefois, contrairement à UAN, la plupart des formalismes ne distinguent pas l'exécution véritablement parallèle de l'exécution entrelacée. Ils ignorent le concept d'interruptibilité ou l'évoquent de manière implicite. De même, ils ne permettent pas d'exprimer des délais de garde entre l'exécution de plusieurs tâches. Quant à DIANE, il est un des rares formalismes avec MAD capables d'exprimer le caractère obligatoire ou facultatif d'une tâche, ou, comme JSD* à préciser les déclencheurs (c'est-à-dire qui de l'homme ou de l'ordinateur a l'initiative de la tâche).

4.3.3. Relations entre les entrées et les sorties

La plupart des formalismes de modèle de tâche mettent l'accent sur la spécification des entrées, c'est-à-dire sur la description du comportement que l'utilisateur doit observer pour se servir du système avec succès. Si la finalité du modèle est de prédire l'utilisabilité du système, c'est ignorer le rôle des retours d'information. Et pourtant il a été démontré que les "skilled users become good at getting the right information from the display." [Howes 91, pp. 113]. A ce titre TAG a été étendu par D-TAG (Display-TAG) pour tenir compte des informations visuelles [Howes 90]. La grammaire "multipartite" de Shneiderman en est un autre exemple [Shneiderman 82]. Les formalismes qui considèrent explicitement la description des sorties, tels CLG et UAN, manquent cependant de rigueur ou bien sont réducteurs. Par exemple, l'opération UAN, "display[Object]" ne fournit aucune information sur le rendu visuel de cet objet (texture, couleur, structure, etc.). Il convient alors de faire comme dans JSD* d'accompagner la spécification d'images d'écran obtenues, par exemple, au moyen d'un outil de prototypage.

4.4. Lisibilité

La lisibilité d'un formalisme vient souvent en contradiction avec la rigueur et le pouvoir d'expression. il semble que les graphes orientés conviennent à la représentation des modèles de tâche. S'ils permettent d'exprimer les relations logico-temporelles, ils couvrent mal d'autres besoins plus "algorithmiques".

La présentation tabulaire d'UAN a l'avantage de mettre en regard les actions de l'utilisateur avec celles du système. Cette correspondance permet de vérifier notamment qu'à toute action utilisateur correspond un retour d'information système. Si ce n'est pas le cas, le concepteur doit en justifier les raisons. Inversement, la présentation tabulaire traduit mal la décomposition hiérarchique.

4.5. Facilité/Complexité d'utilisation

Les langages formels de description de tâche ont leurs avantages mais aussi quelques inconvénients non négligeables.

Les avantages sont les suivants : une approche systématique qui amène le concepteur à la précision ainsi que la mise en place d'hypothèses réalistes.

Les inconvénients majeurs de tels formalismes sont les coûts d'apprentissage mais aussi, pour la plupart, l'absence de méthode pour trouver la "bonne structuration" de l'espace des tâches. Cette remarque vaut pour SIROCO, UAN et bien d'autres. Nous avons cependant trouvé des éléments de réponse dans le processus structuré de JSD* et, dans une moindre mesure dans DIANE. Sébilotte fournit également une description détaillée de la démarche à suivre pour l'analyse (enquête, interview, questionnaires, etc.) [Sébilotte 91]. De même, elle donne quelques moyens de vérification, comme le recueil de scénarios suivi par une vérification du recouvrement du scénario par le modèle ou la mise en place de discussions avec l'utilisateur sur le modèle élaboré. Toutefois cette démarche est empirique, avec les biais et la compétence que cela suppose.

5. Synthèse

Cette conclusion s'articule autour de trois tableaux synthétiques de notre réflexion sur la notion de tâche. Le premier (figure 17) répertorie les modèles de tâches en fonction de leur finalité. Les deux autres montrent le bilan en fonction du formalisme : figures 18 et 19.

Modèles	FINALITE				
	Analyse des besoins	Concept° IHM	Concept° Logicielle	Réalisat°	Evaluat°
ETAG	-	+	?	en projet	?
UAN	+	+	-	-	-
DIANE	+	+	-	-	-
JSD*	+	+	?	?	-
MAD	+	+	en projet	en projet	-
CLG	+	+	-	-	+
GOMS	+	-	-	-	+
ETIT	+	-	-	-	+
SIROCO	-	-	+	+	-
ADEPT	+	+	+	+	-

Figure 17 : Tableau récapitulatif des modèles étudiés en fonction de leur finalité dans le processus de développement des logiciels. Un “+” indique un apport pour l'activité de génie logiciel en regard tandis qu’un “-” dénote l’absence de support pour cette activité; “en projet” signifie des perspectives envisagées par les auteurs ; un “?” souligne une interrogation à laquelle la littérature ne nous a pas fourni de réponse.

Dans notre classement de la figure 18, on pourra nous reprocher nos jugements de valeur sur la lisibilité et la facilité d'utilisation des formalismes. Lorsque nous affichons un engagement, qu'il soit positif ou négatif, celui-ci s'appuie sur l'expérience des membres de notre équipe. En toute rigueur, nous aurions dû mener une évaluation plus approfondie élargie à d'autres communautés.

	FORMALISME						
	Elément	Rigueur	Puiss. d'expression			Lisibilité	Facilité
Modèles	directeur	sémantique	Niv.	Rel.	E/S		d'utilisat°
ETAG	Concept	?	4		E	-	?
UAN	Tâche	-	qqs		E+S	+-	+-
DIANE	Tâche	+	qqs		E	+	+
JSD*	Tâche	?	qqs	*cf.*	E+S	+-	?
MAD	Tâche	?	qqs	*fig.*	E	+-	?
CLG	Tâche	-	6	*19*	E+S	+-	?
GOMS	Tâche	-	4		E	+	+
ETIT	Tâche	-	2		E	+	?
SIROCO	Concept	+	1		E+S	+-	+-
ADEPT	Tâche	+	qqs		E	+	?

Figure 18 : Tableau récapitulatif des systèmes étudiés en fonction du formalisme. Un "+" indique que le caractère du formalisme en regard est souligné par le modèle étudié. "+-" indique que le caractère en regard est plus ou moins souligné par le modèle étudié, alors qu'un "-" signifie que le caractère étudié n'est pas ou peu souligné par le modèle. "qqs" signifie que le nombre des niveaux de description est laissé à l'initiative du concepteur. Un "?" indique notre incapacité à porter un jugement.

La figure 19 développe la colonne “Relations et Attributs” du tableau de la figure 18. Les relations envisagées ici sont temporelles :

seq : la séquence (A puis B),

alter : l'alternative (A ou B),

com : la composition (A et B dans un ordre quelconque),

itér : l'itération,

entr : l'entrelacement (A en même temps que B, de manière entrelacée),

// : le parallélisme vrai,

dél : l'existence de délais entre l'exécution de plusieurs tâches.

Les attributs répertoriés sont :

obli : le caractère obligatoire ou facultatif d'une tâche,

décl : la capacité de préciser le déclencheur de la tâche (l’utilisateur ou le système),

	FORMALISME								
	Relations entre tâches							Attributs	
Modèles	seq	alter	com	itér	entr	//	dél	obli	décl
ETAG	+	-	?	?	-	-	-	-	-
UAN	+	+	+	+	+	+	+	-	-
DIANE	+	+	+	+	+-	+	-	+	+
JSD*	+	+	+	+	+-	-	-	+-	+
MAD	+	+	+	+	+-	+	-	-	-
CLG	+	+	+	+	-	-	-	+	+-
GOMS	+	+	+	+	-	-	-	-	-
ETIT	?	?	?	?	?	?	?	?	?
SIROCO	+	+	+	+-	+	+	-	-	-
ADEPT	+	+	+	+	-	-	-	-	-

Figure 19 : Tableau récapitulatif des relations/attributs offerts par le formalisme du modèle en regard. Un(e) relation/attribut peut être formulé(e) de manière explicite (+) ou implicite (+-. Un “?” indique qu'un(e) tel(le) relation/attribut n'existe pas ou que les documents lus ne nous ont pas apporté de réponse.

Dans ce chapitre nous avons passé en revue des méthodes et modèles participant à l'analyse de tâche. Cette analyse est une étape préliminaire à toute conception et son rôle dans le processus d'évaluation est généralement central. C’est le cas pour l’outil d'aide à l'évaluation ergonomique que nous avons conçu et que nous présentons au chapitre IV. Mais avant, il convient d’étudier les méthodes et techniques d’évaluation existantes. C’est l’objet du prochain chapitre.

Chapitre III

Evaluation ergonomique

On reconnaît la nécessité de produire des logiciels utiles et utilisables mais l'évaluation ergonomique est encore jugée trop coûteuse, réductrice et peu fiable. Les concepteurs et les développeurs de logiciels ne sont pas toujours convaincus que le rapport "coût/bénéfice" de l'évaluation ergonomique soit à leur avantage. A cela, nous répondons qu'à chaque situation correspond un ensemble de méthodes idoines. En génie logiciel, l'Assurance Qualité se pratique à façon, en fonction des ressources et des objectifs. A chaque cas, sa méthode. Nous reprenons ce principe à notre compte en l'appliquant à l'évaluation ergonomique.

La figure 1 traduit notre point de vue : pour remplir sa fonction, une technique d'évaluation a des requis et produit des résultats. Les ressources matérielles et les moyens humains mis en jeu par une méthode constituent des requis. Les données quantitatives de performance sont des exemples de résultat. Connaissant le coût des requis de chaque méthode, la nature des résultats souhaités et les moyens disponibles dans l'environnement, chaque évaluateur doit pouvoir identifier une ou plusieurs méthode(s) qui réponde(nt) à son analyse de coût et de risque.

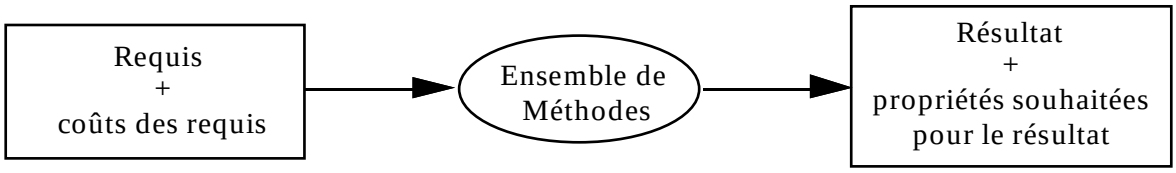


Figure 1 : Requis et produits d'une méthode d'évaluation. Principe du choix d'une méthode en fonction des coûts et des moyens.

Il convient maintenant d'identifier les dimensions qui permettent de caractériser puis d'évaluer les requis et les résultats d'une méthode. C'est ce que nous nous proposons de faire dans ce chapitre. Nous offrons ainsi un espace de classification qui, utilisable comme aide au choix d'une méthode d'évaluation, va bien au-delà des taxonomies usuelles.

Ce chapitre est structuré comme suit : dans le premier paragraphe, nous présentons les classifications les plus répandues. Une analyse de l'existant nous amène à définir une nouvelle taxonomie qui ouvre la voie au choix d'une méthode d'évaluation adaptée à la situation. Ce travail est décrit au paragraphe 2. En 3, nous retenons un sous-ensemble simplifié de notre espace de classification pour présenter nos réflexions sur les méthodes et outils d'évaluation actuels les plus représentatifs.

1. Classifications usuelles

La classification la plus fréquente est fondée sur la distinction entre approches prédictives et approches expérimentales [Nielsen 90a, Jeffries 91, Bastien 91]. Senach parle respectivement de méthodes analytiques et de méthodes empiriques [Senach 90]. Whitefield et ses co-auteurs organisent leur classification en fonction de la présence de l'utilisateur et du système [Whitefield 91]. Les paragraphes qui suivent traduisent ces deux points de vue.

1.1. Approches prédictives et approches expérimentales

La figure 2 illustre la classification des méthodes d'évaluation en fonction du recours ou non à l'observation expérimentale.

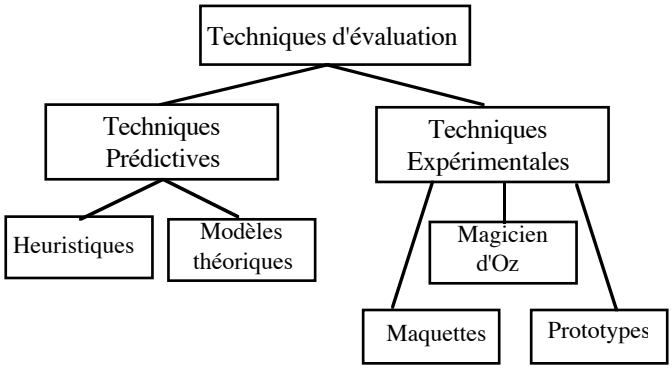


Figure 2 : Classification usuelle des techniques d'évaluation.

Les méthodes prédictives (ou analytiques) ne nécessitent pas la présence de l'utilisateur final ni l'implémentation, même partielle, du système. De fait, elles conviennent aux toutes premières étapes du cycle de vie d'un logiciel : analyse des besoins et conception. Ces techniques permettent, à partir d'une description du système, voire de l'utilisateur, d'identifier des problèmes potentiels d'utilisabilité. Selon les modèles, théoriques ou heuristiques, les descriptions utilisées sont formelles, semi-formelles ou informelles :

- Les modèles théoriques, tels GOMS, PUMS et ICS sur lesquels nous reviendrons au paragraphe 3, requièrent une représentation formelle explicite. Ces derniers, en raison de leurs fondements théoriques, peuvent également servir d'outils explicatifs applicables à des phénomènes observés sur un système réel. D'autres recherches s'appuient sur une notation mathématique pour exprimer de manière non ambiguë des propriétés ou critères d'utilisabilité telles que la prédictibilité et l'observabilité [Monk 87, Dix 91]. Ces travaux devraient

servir de base au développement d'outils capables de détecter de manière automatique qu'un système donné satisfait certaines propriétés.

- Les approches heuristiques tels les "walk-through" et les "jog-through" sur lesquels nous reviendrons ultérieurement, s'appuient sur la connaissance d'évaluateurs experts qui, au vu de la description semi-formelle ou informelle du système, détectent des difficultés potentielles. La connaissance mise en jeu peut aller des guides ergonomiques façon Smith et Mosier [Smith 86] à des ensembles de règles simplifiés comme celles de Scapin [Scapin 86] et Nielsen [Nielsen 90a]. Nous reviendrons sur ce point au paragraphe 3.1.1.2.

Les méthodes expérimentales reposent sur le recueil de données comportementales de sujets mis en situation. Le recueil s'effectue sur support audiovisuel, et/ou par capture automatique d'actions ("monitoring") ou encore par interviews et questionnaires. Les données observées font ensuite l'objet d'une analyse. L'analyse porte généralement sur les chemins suivis et les procédures adoptées par les utilisateurs, les temps d'exécution, la nature, la cause et la fréquence des incidents [Valentin 93]. Elle conduit à une synthèse qui indique notamment le degré de gravité des incidents et si possible, des recommandations.

A l'inverse des méthodes prédictives, les approches expérimentales :

- requièrent la présence d'une population représentative du futur utilisateur du système,
- fournissent des données observées du monde réel, non pas des résultats prédictifs approximatifs voire réducteurs,
- se pratiquent généralement à partir de la phase de conception.

Pourtant, comme les méthodes prédictives, les approches expérimentales peuvent être appliquées dès les premières étapes du cycle de vie sous forme par exemple d'enchaînement d'écrans esquissés sur papier (en anglais : "story-board") ou produit par un générateur d'interfaces de type "présentation" [Nigay 94]. On n'insistera jamais assez que dès l'analyse des besoins, une maquette même rudimentaire peut utilement servir à comprendre le problème, à impliquer et convaincre le client, et, pour ce qui nous intéresse, à construire les modèles de tâche et de l'utilisateur. Selon les outils de développement disponibles, la nature du projet et la compétence des architectes du logiciel, la

maquette sera jetée[9] ou bien se transformera progressivement en prototype puis en produit.

De manière générale, les méthodes prédictives et expérimentales se complètent et peuvent se pratiquer tout au long du processus de développement. La pratique itérative "tests-corrections" définit le fondement de l'évaluation "formative" [Hix 93] : chaque évaluation fournit de nouveaux enseignements dont l'intégration conduit à une nouvelle version du produit de la conception et/ou du logiciel. Le développement d'OMS, la messagerie vocale des jeux olympiques de Los Angeles, illustre parfaitement l'évaluation formative [Gould 87a]. La conception itérative convient au processus de développement en spirale mais elle s'inscrit également dans un cycle de vie en V avec retours arrière. A l'évaluation formative, on oppose l'évaluation "summative" pratiquée une seule fois en fin de mise au point ou pour comparer des logiciels existants [Hix 93].

1.2. Présence virtuelle ou réelle de l'utilisateur et du système

La figure 3 reproduit la classification de Whitefield et de ses co-auteurs présentée dans [Whitefield 91]. Ici, le critère directeur est la ressource matérielle (le système informatique) et les moyens humains (les sujets) requis pour la mise en application d'une méthode. La présence de l'utilisateur (ou du système) peut être virtuelle ou réelle. "Présence virtuelle" de l'utilisateur (ou du système) signifie que la méthode ne nécessite pas l'intervention d'un utilisateur réel (ou l'utilisation d'une version implémentée du système). La technique requiert au plus une représentation (formelle ou non) de l'utilisateur représentatif ou du système.

Présence de l'utilisateur / Présence de l'ordinateur	Virtuelle	Représentationnelle
Virtuelle	**Méthodes analytiques**	**Rapports sur l'utilisateur**
Réelle	**Rapports de spécialistes**	**Méthodes d'observation**

Figure 3 : Classification des méthodes d'évaluation d'après [Whitefield 91].

[9] Cette approche répond au principe de la "philosophie en matière de crêpe" (Pancake philosophy) appelé aussi "sagesse en matière de gauffre" (waffle wisdom) [Hix 93, p. 283] : comme pour les crêpes et les gauffres, les premières maquettes sont faites pour être jetées.

Cette organisation centrée sur les ressources sujet/système, conduit à quatre classes de méthodes :

- les méthodes analytiques où l'utilisateur et le système sont virtuels. Ces approches recouvrent la classe des modèles théoriques du paragraphe précédent. Elles nécessitent peu de ressource, elles sont rapides mais, dans l'état actuel de nos connaissances, la portée des résultats est limitée et les prédictions n'ont pas toujours la fiabilité ou la précision attendues ;

- à l'autre extrême, les méthodes d'observation où l'utilisateur et le système sont réels. Ces méthodes s'inscrivent dans la classe des techniques expérimentales présentées au paragraphe 1.1. Nous y avons évoqué les avantages et les inconvénients : les données sont précises mais en raison de leur manque de concision, l'analyse est longue et leur interprétation est parfois difficile ;

- les méthodes hybrides qui combinent sujet virtuel et système réel : il s'agit des approches heuristiques du paragraphe 1.1 dont la précision et la couverture reposent sur la compétence et le nombre de spécialistes engagés dans l'évaluation. A ce titre, nous rappelons qu'il faut prévoir entre trois et cinq spécialistes pour obtenir une détection raisonnable des problèmes [Pollier 91, Nielsen 90a] ;

- les méthodes hybrides qui combinent sujet réel et système virtuel : il s'agit d'une sous-classe des méthodes expérimentales du paragraphe 1.1. Ici, la pratique s'appuie sur les interviews, les questionnaires. A l'inverse des méthodes d'observation, qui recueillent des données comportementales d'un sujet manipulant un système réel, ces méthodes reposent sur une utilisation imaginaire du système. Par conséquent, les données recueillies sont essentiellement subjectives avec le risque d'être peu fiables. Mais elles ont l'avantage d'être plus synthétiques que les données des méthodes d'observation. Les approches "sujet réel-système virtuel" sont néanmoins utiles dans la toute première phase du processus de développement.

1.3. Lacunes des taxonomies existantes

Le découpage entre méthodes prédictives et expérimentales met en lumière les deux grands courants de recherches et de pratiques dans le domaine de

l'évaluation ergonomique. Si cette distinction est utile comme élément de structuration du domaine, elle n'offre aucun support au problème du choix d'une méthode face aux contraintes d'une situation donnée.

La classification de Whitefield et de ses co-auteurs se rapproche davantage de nos préoccupations : la notion de présence de l'utilisateur et du système fournit quelques indications sur la nature des ressources requises pour appliquer une méthode donnée. Elle en donne aussi sur la qualité des résultats obtenus (précision, fiabilité, coût d'analyse). Nous estimons cependant que d'autres ressources sont en cause et notamment, comme l'indique Senach, les outils informatiques d'aide à l'évaluation [Senach 90].

Dans le paragraphe qui suit nous présentons une taxonomie plus précise motivée par le problème du choix de méthodes d'évaluation.

2. Une taxonomie pour le choix de méthodes d'évaluation

La figure 4 présente de manière synthétique les dimensions de notre espace de classification. Ces dimensions caractérisent cinq sortes de préoccupations : les moyens humains, les ressources matérielles, les connaissances requises, les facteurs situationnels, et la nature des résultats fournis. Nous analysons en détail chacun de ces points dans les paragraphes qui suivent.

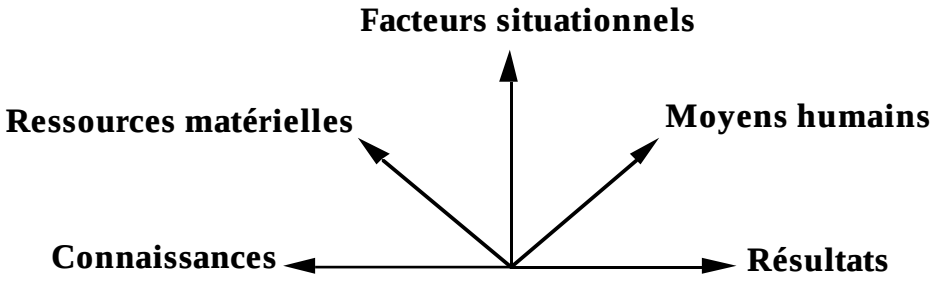


Figure 4 : Notre proposition de taxonomie. Se reporter aux figures 5, 6, 7, 8, 9 pour un affinement des axes moyens humains, ressources matérielles, connaissance, facteurs situationnels, résultats.

2.1. Moyens humains

Les moyens humains désignent l'ensemble des acteurs impliqués directement ou indirectement dans l'activité d'évaluation. Nous distinguons trois classes d'intervenants : les sujets, les évaluateurs spécialistes et une catégorie qui regroupe tous les autres acteurs (les développeurs, les concepteurs d'interface, les clients, les agents commerciaux, etc.).

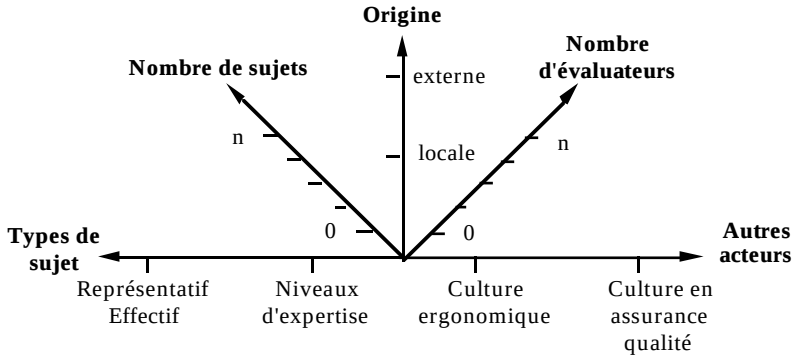


Figure 5 : Le sous-espace des ressources humaines de notre taxonomie.

Comme le montre la figure 5, les acteurs peuvent *faire partie de l'organisation* chargée du développement ou bien peuvent être recrutés à *l'extérieur*. Cette distinction a de l'importance dans l'évaluation des coûts de recrutement. Les facteurs situationnels (voir paragraphe 2.4 ci-dessous) et notamment le type de projet, l'enveloppe budgétaire et le planning pourront avoir un impact sur l'origine et le nombre de sujets et de spécialistes souhaitables.

En l'absence de sujet, il faudra se contenter d'informations prévisionnelles et avoir recours aux techniques prédictives. En l'absence de sujet et de spécialiste, seuls les modèles théoriques pourront être appliqués. On s'interrogera alors sur l'accessibilité des méthodes, c'est-à-dire la connaissance requise pour les appliquer (voir ci-dessous, paragraphe 3). Si, en l'absence de sujet, on choisit une méthode heuristique, on retiendra qu'il faudra prévoir entre trois et cinq spécialistes pour une couverture raisonnable des problèmes potentiels d'utilisabilité.

Si l'on adopte une méthode d'observation, le réalisme des données recueillies dépendra de la représentativité des sujets et des scénarios. Bien que la règle ne soit pas absolue, Valentin et ses co-auteurs recommandent de commencer par définir la population [Valentin 93]. Cette démarche est bien adaptée lorsque l'on dispose de l'*utilisateur effectif* du futur logiciel. Si l'on doit avoir recours à des *utilisateurs représentatifs*, la définition préalable des scénarios et d'un modèle utilisateur peuvent utilement diriger le recrutement des sujets. Typiquement, les sujets sont organisés par groupes en fonction de leur niveau d'expérience dans le domaine de la tâche, voire dans l'utilisation du produit, et leur familiarité avec l'informatique.

Pour la dernière catégorie d'acteurs (c'est-à-dire les intervenants autres que les sujets et les spécialistes), il est important de s'interroger sur leur niveau de

culture en matière d'assurance qualité (au sens large du génie logiciel) mais aussi sur leur attitude vis-à-vis des principes de l'ergonomie. A fortes réticences doit correspondre, nous semble-t-il, une technique d'évaluation légère de type heuristique.

2.2. Ressources matérielles

Les ressources matérielles recouvrent tous les moyens physiques impliqués dans l'activité d'évaluation. La figure 6 montre le sous-espace que nous proposons comme élément de structuration. Nous convenons de distinguer les ressources utilisées ou disponibles comme instruments de capture et les ressources matérielles qui font l'objet de l'évaluation.

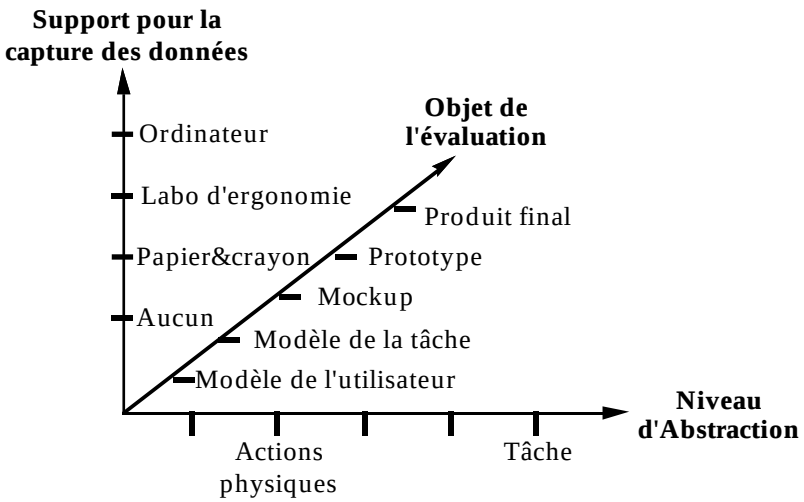


Figure 6 : Le sous-espace des ressources matérielles de notre taxonomie.

Ressources matérielles pour la capture

Quelques exemples de ressources d'aide à la capture : les questionnaires et interviews papier, les régies vidéo (éventuellement portatives) ou laboratoires spécialisés, et les ordinateurs. Toutes ces ressources peuvent être caractérisées par un coût mais aussi par la *portée*, la *nature* et le *niveau d'abstraction* des données qu'elles permettent de recueillir. Au chapitre II sur les tâches, nous avons défini les notions d'actions physiques perceptible et perçue ainsi que les notions de tâches élémentaire et composée. Ces concepts fournissent des éléments pour identifier la nature et le niveau d'abstraction des données du recueil.

Pour illustrer l'intérêt de ces dimensions, nous allons prendre comme exemple le cas de la capture automatique par ordinateur. Dans l'état actuel de cette technique, seules sont captées les actions physiques de type clavier-souris et au mieux la manipulation d'objets liés à l'accomplissement de tâche élémentaire [MacLeod 93, Hammontree 92]. De plus, seule la manipulation des objets

implémentés via une boîte à outils comme Motif est détectable. En conséquence, les outils actuels de capture automatique, ne couvrent que les phénomènes de bas niveau d'abstraction. De plus, pour un niveau donné, ils présentent des ruptures de portée : capture d'un sous-ensemble des actions physiques et capture d'un sous-ensemble des objets possibles. Si l'objectif est une évaluation fiable, ces limitations impliquent des compléments de test. Le projet NEIMO vise à combler ces lacunes [Coutaz 93b].

Ressources faisant l'objet de l'évaluation

Comme le montre la figure 6, les produits du processus de développement peuvent faire l'objet d'une évaluation. Ce peut être : le modèle de tâche, le modèle de l'utilisateur, les spécifications externes du système, une maquette, un prototype, un produit, etc.

L'objet de l'évaluation peut être décrit de manière formelle, semi-formelle ou informelle ; et peut admettre plusieurs formes de supports (papier, maquette carton, ordinateur). A chacune de ces formes de présentation correspondent un coup de développement et des compétences.

2.3. Connaissances requises

Les connaissances requises pour mettre en application une méthode sont de deux sortes (voir figure 7) :

- les premières ont trait au **coût cognitif d'accès** à la méthode. Par exemple, toutes les approches heuristiques nécessitent une formation en ergonomie cognitive. Si l'évaluateur dont on dispose n'a pas cette compétence, il est sans doute préférable pour la fiabilité des résultats de s'orienter vers un modèle prédictif de type keystroke (sous réserve que ce niveau d'abstraction soit suffisant pour l'étude) ; ou encore appliquer les recommandations de Nielsen qui permettent de pratiquer une "ingénierie de l'utilisabilité à faible coût"[10] [Nielsen 89] ;

- les secondes connaissances concernent les **descriptions nécessaires en entrée** au bon fonctionnement de la méthode : modèle de l'utilisateur, modèle de la tâche, spécifications externes de l'interface, définition de scénarios. Plus le spectre des informations d'entrée est large, plus la

[10]En anglais, "discount usability engineering".

méthode est coûteuse à mettre en œuvre. En revanche, on peut espérer obtenir en sortie des résultats robustes et concis.

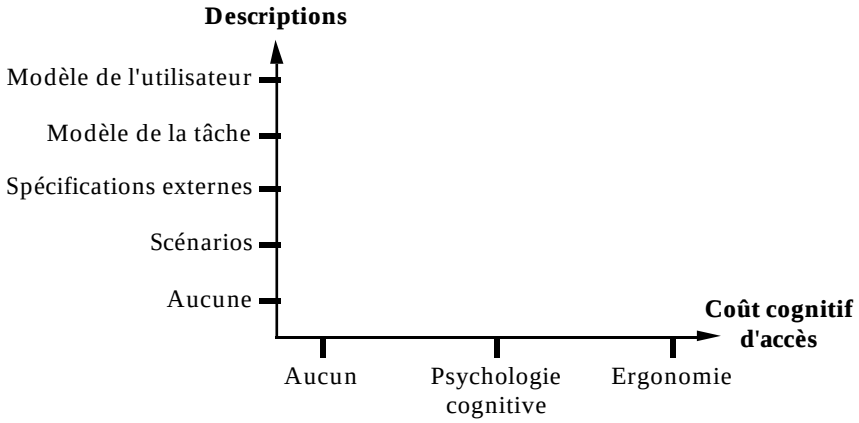


Figure 7 : Le sous-espace des connaissances requises dans notre taxonomie.

2.4. Facteurs situationnels

Les facteurs situationnels définissent le contexte de l'évaluation. Comme le montre la figure 8, nous proposons de distinguer : l'étape considérée dans le processus de développement, le lieu dans lequel l'évaluation peut être pratiquée, la typologie des applications et des techniques d'interaction, l'enveloppe budgétaire et les contraintes de planning.

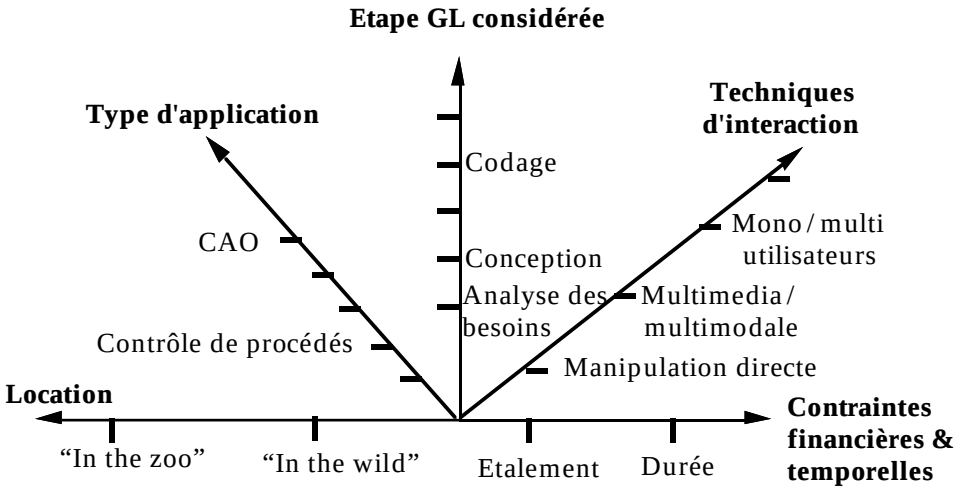


Figure 8 : Le sous-espace des facteurs situationnels de notre taxonomie.

- **Etape dans le processus de développement** : il convient de noter que l'étape considérée du cycle de vie est orthogonale à l'objet de l'évaluation. En particulier, une maquette (objet de l'évaluation) peut servir à l'analyse des besoins tout comme elle peut servir à valider des spécifications externes. Toutefois, le niveau de réalisme de la maquette variera en fonction de l'étape et de la nature du projet, du type donc de

cycle de développement : durée de vie prévue du logiciel (version unique d'un logiciel expérimental publicitaire pour valider des concepts ou versions multiples de logiciel pérenne à forte pénétration industrielle, etc.).

- **Lieu de l'évaluation** : l'évaluation peut être pratiquée dans l'entreprise cliente ou bien en laboratoire[11]. Les expériences de laboratoire sont plus faciles à contrôler mais on devra s'interroger sur la validité "écologique" des données recueillies : le comportement dans le Zoo n'est pas nécessairement celui de la Nature.

- **Typologie des applications et des tâches** : on convient généralement de distinguer les applications de type CAO pour lesquelles les tâches sont à structure variable, les applications de type contrôle de procédés où l'on relève des tâches opératives de routine, les applications grand public qui militent en faveur d'un dialogue guidé par le système, etc. A l'évidence, les exigences sur la nature du recueil varient en fonction du domaine applicatif. En particulier, le réalisme des données recueillies est un critère prioritaire pour les applications où la sécurité est un facteur dominant.

- **Typologie des interfaces** : interfaces graphiques à manipulation directe, interfaces multimédia et multimodales (on trouvera dans [Nigay 94] une définition précise de ces termes). Aucun modèle théorique ne peut prétendre aujourd'hui assurer l'évaluation des interfaces multimédia et multimodales. Pour ces nouvelles interfaces, les heuristiques émergent à peine [Sutcliffe 94]. A l'heure actuelle, les méthodes d'observation constituent, à notre sens, les seules approches réalistes.

- **Enveloppe budgétaire et planning** : l'enveloppe budgétaire et le planning sont deux données incontournables qui conditionnent le choix d'une technique d'évaluation. Le planning définit deux formes de contraintes temporelles : la durée que l'on peut dédier à l'évaluation (par exemple, trois jours au total) mais aussi son étalement dans le temps (trois jours répartis, par exemple, au maximum sur deux semaines).

[11] En anglais, on utilise parfois les expressions métaphoriques "design in the zoo" pour désigner les expérimentations en laboratoire et "design in the wild" pour les expériences in situ.

2.5. Résultats fournis

La figure 9 synthétise les dimensions qui permettent de caractériser les résultats d'une méthode d'évaluation donnée. Certaines dimensions ont trait au contenu, d'autres à la qualité, d'autres encore au support de restitution.

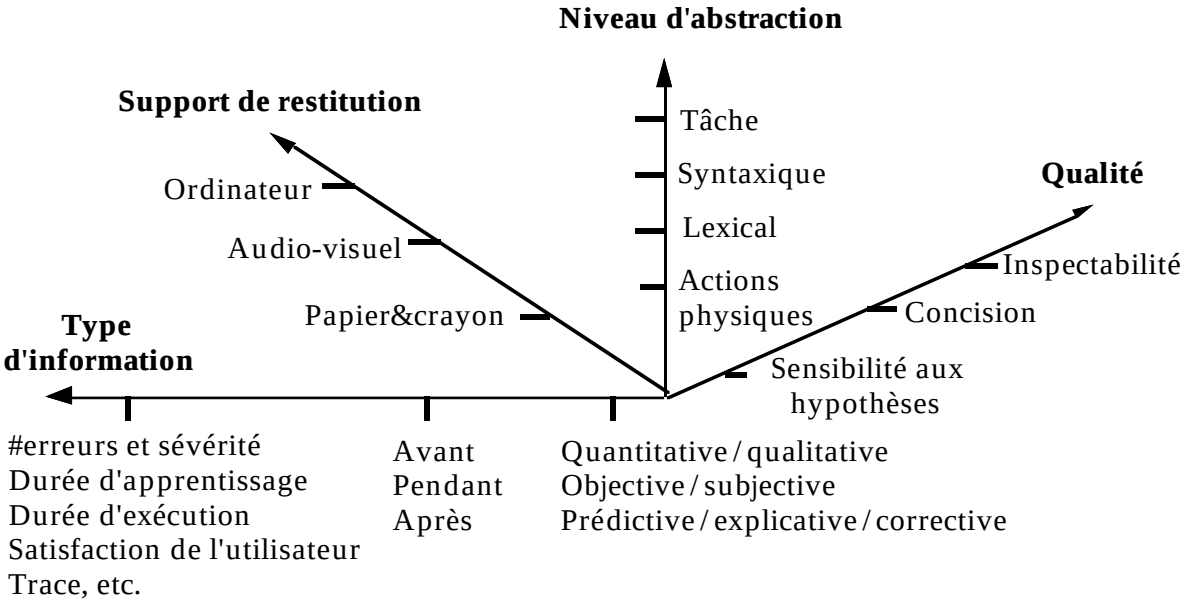


Figure 9 : Le sous-espace des résultats d'une méthode dans notre taxonomie.

- Le contenu s'étudie selon deux volets orthogonaux : le niveau d'abstraction des informations (niveaux tâche, syntaxique, lexical et physique) et le type d'information. A son tour, une information d'un type donné se distingue par :
  - sa valeur sémantique : nombre d'erreurs et degré de gravité, nombre de recommandations transgressées, temps d'exécution et d'apprentissage, degré de satisfaction de l'utilisateur, traces, etc.,
  - la tranche d'exécution du scénario à laquelle elle correspond: avant l'exécution du scénario, pendant et après,
  - sa nature : quantitative (il s'agit d'un résultat numérique comme un temps d'apprentissage) ou qualitative (résultat non numérique comme une liste de problèmes d'utilisabilité) ; subjective (qui représente une opinion) ou objective (c.-à-d. une mesure observée) ; explicative si elle permet de comprendre la raison du problème et corrective si elle fournit le remède au problème d'utilisabilité ;

- Sous l'en-tête qualité nous regroupons des critères généraux qui dénotent la robustesse des résultats de la méthode d'évaluation :
  - sensibilité de l'information aux hypothèses. Typiquement GOMS est

sensible à l'hypothèse du comportement expert ;
- concision de l'information qui joue sur les temps d'analyse. Comme contre-exemple, citons le manque de concision des enregistrements audiovisuels ;
- inspectabilité de l'information qui permet au spécialiste de parcourir les résultats et de comprendre les phénomènes.

• Le support de restitution est aussi un ingrédient dans la restitution des résultats. Nous distinguerons entre la présence et l'absence de support automatisé. Aujourd'hui, la plupart des outils informatisés assurent l'inspectabilité du recueil des données comportementales, mais vont rarement au-delà de la détection automatique des problèmes. De plus, les problèmes détectés relèvent des niveaux d'abstraction les plus bas et s'appuient uniquement sur des informations quantitatives objectives. Nous reviendrons sur ce point au paragraphe 3.

Nous venons de présenter les dimensions qu'il convient de considérer dans le choix d'une méthode d'évaluation ergonomique. Nous ne prétendons pas à l'exhaustivité mais nous estimons avoir fourni une bonne couverture de l'espace actuel et à venir des méthodes d'évaluation. Au paragraphe suivant, nous présentons un sous-ensemble simplifié de ce cadre taxonomique mais suffisant pour l'analyse des méthodes et outils actuels les plus pertinents.

2.6. Un cadre taxonomique simplifié

Nous retiendrons trois dimensions : l'utilisateur et l'interface, le savoir utilisé pour l'évaluation, et l'automatisation.

• La dimension de l'utilisateur et de l'interface précise si le mécanisme d'évaluation collabore avec l'utilisateur et utilise ou non une implémentation informatique de l'interface à évaluer. Cette première dimension reprend l'idée de Whitefield et de ses co-auteurs présentée en 1.2.

• La dimension du savoir indique les connaissances utilisées pour mener à bien l'évaluation. Senach distingue les connaissances empiriques et analytiques. Nous prolongeons cette distinction et prenons en considération :
- d'une part, l'utilisation d'un savoir heuristique. Ce savoir ne résulte pas de connaissances formelles mais s'appuie principalement sur l'expérience de l'expert,

- et d'autre part, nous considérons l'utilisation de modèles ou de théories sur l'interface, l'utilisateur ou la tâche. Ces connaissances peuvent être détenues par un être humain ou codées en machine.

• L'automatisation repère les capacités d'un système d'aide à l'évaluation : absence d'automatisation, capture de données comportementales, détection de problème d'utilisabilité, et génération d'actions correctives.

2.6.1. Utilisateur et Interface

Nous ne revenons pas sur les différents modes de participation du sujet dans le processus d'évaluation (interviews, questionnaires, Magicien d'Oz, etc.) et ne conservons que la distinction dichotomique "avec" ou "sans" participation de l'utilisateur. Nous ferons de même pour l'implémentation informatique de l'interface. Les quatre états pour la dimension "Utilisateur et Interface" sont donc:

- **U**tilisateur **P**résent et **I**nterface **I**mplémentée (***UP/II***).
- **U**tilisateur **P**résent et **I**nterface **N**on implémentée (***UP/IN***).
- **U**tilisateur **A**bsent et **I**nterface **I**mplémentée (***UA/II***).
- **U**tilisateur **A**bsent et **I**nterface **N**on implémentée (***UA/IN***).

Le tableau de la figure 10 rappelle, s'il en est besoin, différentes méthodes d'évaluation ergonomique selon les critères de la dimension Utilisateur/Interface.

\ **U**tilisateur **I**nterface \	**A**bsent	**P**résent
Non implémentée	Evaluation de spécifications externes, de simulations papiers. Outils formels d'évaluation (grammaires, cognitive walkthrough, etc.)	Questions à l'utilisateur (interview, enquêtes, etc.), simulations par Magicien d'Oz
Implémentée	Evaluation de prototype, de simulation de l'interface et/ou de simulation de l'utilisateur	Observation de l'utilisateur par le biais de l'ordinateur (trace des actions essentiellement), ou en train d'utiliser ce dernier (audiovisuel)

Figure 10 : Tableau récapitulatif de la dimension de la présence de l'utilisateur et de l'implémentation de l'interface.

2.6.2. Savoir utilisé

Cette dimension souligne la prise en compte du type de savoir utilisé pour l'évaluation ergonomique. Elle se décompose en sept états.

- Le premier état rassemble les techniques d'évaluation reposant sur un savoir heuristique. Le savoir heuristique tient à l'expérience personnelle de l'évaluateur qui peut également s'inspirer de conseils ("guidelines") [Smith 86], de principes [Nielsen 90b] ou de métriques [Whiteside 88, Colbert 92] (ces techniques sont présentées au paragraphe 3.1.2).
- Les six autres états sont issus de la combinaison deux à deux du détenteur et l'objet du savoir. Le détenteur du savoir peut être un individu ou une machine. Le savoir peut porter soit sur l'utilisateur, soit sur l'interface, soit sur la tâche. Par exemple, PUM (Programmable User Model) s'appuie sur une représentation compilée des connaissances de l'utilisateur sur l'interface et sur la tâche à exécuter [Young 90] (technique développée au paragraphe 3.3.1.2) ; dans ce cas, l'objet du savoir utilisé pour l'évaluation est un savoir sur l'utilisateur et ce savoir est codé en machine.

En résumé, les sept états (1 + 2*3) de la dimension du savoir sont :

- le savoir heuristique (***Heuristique***).
- le savoir humain, sur l'utilisateur (***HUtilisateur***).
- " " " " " , sur l'interface (***HInterface***).
- " " " " " , sur la tâche (***HTâche***).
- le savoir codé en machine, sur l'utilisateur (***CUtilisateur***).
- " " " " " " " , sur l'interface (***CInterface***).
- " " " " " " " , sur la tâche (***CTâche***).

2.6.3. Automatisation

La dimension de l'automatisation souligne différents niveaux d'automatisation d'outils d'aide à l'évaluation. Nous distinguons les techniques d'évaluation :

- non automatiques (***non auto.***).
- qui autorisent une ***capture*** automatique
- qui produisent automatiquement une ***analyse***.
- qui engendrent automatiquement une ***critique***.

- Les méthodes non automatiques sont menées de bout en bout par un ou plusieurs évaluateurs humains.

- Les méthodes à capture automatique enregistrent aussi bien des informations audiovisuelles, que la frappe au clavier, la manipulation de la souris, la parole, etc. (nous ne revenons pas sur le niveau d'abstraction et la portée des données recueillies).
- L'analyse automatique détecte des problèmes d'utilisabilité.
- Les critiques localisent les problèmes rencontrés et fournissent des propositions d'amélioration.

3. Systèmes et techniques d'évaluations ergonomiques : état de l'art

Nous avons choisi d'organiser cet état de l'art selon la dimension de l'automatisation : systèmes non automatisés, systèmes à capture automatique, système avec analyse automatique et critique automatique.

3.1. Techniques non automatisées

Nous distinguons trois classes de méthodes non automatisées : celle pour lesquelles le savoir heuristique reste la base incontournable pour mener à bien l'évaluation, les techniques fondées sur des métriques et les méthodes qui s'appuient sur des connaissances formelles.

3.1.1. Où le savoir heuristique domine

Ce paragraphe regroupe deux ensemble de directives pour l'évaluation : des recommandations suivies de principes plus synthétiques.

3.1.1.1. Les recommandations de Smith & Mosier

Les recommandations de Smith et Mosier sont le fruit d'une pratique approfondie [Smith 86, Mosier 86]. Elles paraissent simples d'accès même pour le non spécialiste. En réalité, cette simplicité n'est qu'apparente : le rapport [Smith 86] contient 944 recommandations consignées dans un document de 480 pages. Nous citons ci-dessous deux exemples illustratifs :

- recommandation 2.4•20 (aire de fonctionnalité : affichage des données, fonction: graphique) : "Affichage graphique pour l'impression : quand des écrans doivent être imprimés, permettre à l'utilisateur de les visualiser comme ils apparaîtront sur le listing en sortie".
- recommandation 6.3•9 (aire de fonctionnalité : protection des données, fonction : Entrée des données/changement) : "Correction immédiate des erreurs : quand une donnée erronée en entrée est détectée par

l'ordinateur, permettre à l'utilisateur d'effectuer une correction immédiate".

Peut-on imaginer le temps que prendrait une évaluation qui voudrait vérifier chacun des conseils pertinents pour l'application, même si le total de ces conseils était bien inférieur à 944! De plus, ces recommandations sont quelquefois obscures. Par exemple :

- recommandation 2.4•18 (aire de fonctionnalité : affichage des données, fonction: graphique) : "Estimer l'animation, le mouvement des données sous le contrôle de l'ordinateur pour l'affichage illustré d'objets complexes".

Aucune explication n'est donnée sur la façon de mettre en œuvre l'estimation d'une animation. Cet exemple n'est pas sans nous rappeler combien le savoir de l'approche artisanale (au sens de Long, cf. Chapitre I) est difficilement transmissible.

3.1.1.2. Des principes plus synthétiques

Nous trouvons dans [Scapin 90], [Vanderdonckt 90] ou [Nielsen 90a] des principes plus synthétiques que ceux de Smith et Mosier. Ces principes donnent au concepteur une démarche à suivre, permettent de structurer l'espace de travail sous forme de règles. Ci-dessous nous présentons le premier niveau du jeu de critères ergonomiques de Scapin.

Les critères proposés par Scapin sont structurés en trois niveaux. Le premier comprend huit critères principaux. Le second niveau adjoint des sous-critères aux critères principaux et ainsi de suite. Nous ne présentons ici que les huit critères principaux : le guidage, la charge de travail, le contrôle explicite, l'adaptabilité, la gestion des erreurs, l'homogénéité, la signifiance des codes, et la compatibilité. Les définitions données ci-dessous sont en partie extraites de l'Annexe 1 de [Bastien 91].

- Le ***guidage*** offre une information rapide et pertinente sur l'état du système. C'est l'ensemble des moyens mis en œuvre pour conseiller, orienter, informer et guider l'utilisateur lors de ses interactions avec l'ordinateur (messages, alarmes, labels, etc.).

- La ***charge de travail*** permet de réduire la charge mnésique (mémoire à court terme). Elle concerne l'ensemble des éléments de l'interface qui ont un rôle, pour l'utilisateur, dans la réduction de sa charge perceptive ou mnésique et dans l'augmentation de l'efficacité du dialogue.

- L'utilisateur doit ***contrôler explicitement*** le logiciel et si tel n'est pas le cas, le logiciel doit en donner l'illusion. Le contrôle explicite se réfère à la fois au contrôle de l'utilisateur sur l'interface ou le logiciel, et au caractère explicite de ses actions.

- L'***adaptabilité*** propose différents niveaux d'utilisation du logiciel. Elle se réfère à la capacité du système à réagir selon le contexte et selon les besoins et préférences de l'utilisateur.

- La ***gestion des erreurs*** doit réduire les occasions d'erreur et toute erreur doit être détectable dès son occurrence et doit pouvoir être corrigée. Elle concerne tous les moyens permettant d'une part d'éviter ou de réduire les erreurs, et d'autre part de les corriger lorsqu'elles surviennent.

- L'***homogénéité*** s'apparente à la notion de cohérence (par exemple, séquence de commandes identiques pour un même résultat). Elle se réfère à la façon avec laquelle des choix d'objets de l'interface (code, procédures, dénominations, etc.) sont conservés pour des contextes identiques, et des objets différents pour des contextes différents. L'homogénéité s'applique aussi bien à la localisation et au format qu'à la syntaxe et la dénomination.

- La ***signifiance des codes*** doit être non ambiguë. Elle se réfère à l'adéquation entre l'objet ou l'information affichée ou demandée, et son référent.

- La ***compatibilité*** suppose un faible recodage des informations entre le savoir de l'utilisateur et le format imposé par le logiciel. Elle réfère à l'accord pouvant exister entre les caractéristiques de l'utilisateur (mémoire, perceptions, habitudes, etc.) et l'organisations des sorties, des entrées et du dialogue.

Dans [Bastien 93], Bastien démontre l'intérêt de ce petit ensemble de critères sur les guides façon Smith et Mosier : deux groupes d'experts ont été amenés à évaluer une interface utilisateur. Seul le premier groupe avait à sa disposition

les critères de Scapin. Le premier groupe a découvert un nombre de problèmes significativement plus important que le second groupe.

Toutefois, l'utilisation de tels principes n'est possible qu'à condition de pouvoir leur associer des modèles, savoir en quoi un message d'erreur sera plus informatif qu'un autre, ou pouvoir être à même d'estimer, voire de construire le modèle mental de l'utilisateur.

En résumé, les méthodes de type “recommandations” requièrent une expérience préalable importante dans la mesure où elles ne fournissent pas de moyens pratiques pour les mettre en œuvre. Nous présentons maintenant quelques techniques qui font un premier pas dans ce sens.

3.1.2. Utilisation de métriques

Dans l'optique de fournir des outils quantitatifs pour aider à l'évaluation, Whiteside et ses co-auteurs, Dowell et Long, utilisent comme support des mesures objectives de la performance de l'utilisateur [Whiteside 88, Dowell 89].

3.1.2.1. Les métriques de Whiteside

La mesure proposée est la suivante : **S = (C / T) x P** où

S = taux d'achèvement de la tâche (pourcentage de réussite de l'utilisateur par rapport à l'expert),

T = Temps passé sur la tâche par l'utilisateur,

P = pourcentage effectué de la tâche (par l'utilisateur),

C = temps minimal nécessaire pour exécuter la tâche par un expert.

L'évaluateur, via des enregistrements audiovisuels, des interviews et des questionnaires, vient compléter, par une évaluation qualitative, l'évaluation quantitative précédente. Cette évaluation reste, malgré tout, pour une grande part, le produit d'une évaluation artisanale (au sens de Long et Dowell).

3.1.2.2. Les métriques de Colbert

De manière plus rigoureuse, Colbert et ses co-auteurs présentent, dans le cadre d'un système de commande et de contrôle pour la planification de débarquement de personnel militaire [Colbert 92], une évaluation d'interface suivant le modèle général décrit dans [Dowell 89]. Ce modèle intègre les approches liées aux techniques du génie logiciel et de l'ergonomie, et permet de régler les problèmes de conflit entre deux évaluations distinctes, l’une centrée sur l'utilisateur, l'autre sur le logiciel.

Colbert définit le domaine d'application pour la planification selon le schéma donné de la figure 11. Ce schéma résume l'interaction des utilisateurs (que nous appelons des planificateurs) avec le système d'aide à la planification pour produire des plans.

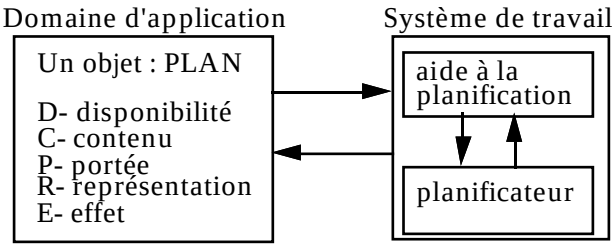


Figure 11 : Schéma de relations pour la planification.

Le domaine est modélisé sous forme de liste d'objets et d'attributs où un objet suggère les aspects qui doivent être pris en considération à l'évaluation. Dans l'exemple traité, les cinq attributs de l'objet PLAN permettent de définir pleinement un plan. La Disponibilité exprime l'opportunité pour autrui (autre que les planificateurs) d'accès au plan et sa mise en application ; le Contenu est ce qui est spécifié par le plan ; la Portée définit et délimite le plan c'est-à-dire les objets impliqués et la durée ; la Représentation est la forme symbolique par laquelle le contenu est communiqué; et l'Effet souligne la valeur ajoutée du plan (par exemple, dans le cas d'un plan militaire, "plus de puissance de feu qu'autrement").

Dans le domaine de la planification, la mesure de Qualité pour chacun des attributs de l'objet PLAN (D, C, P, R, E) est représentée par une équation. Cette équation exprime que la performance du système de planification ($P_{Planning}$) est fonction de la qualité de chacun des attributs du plan ($SQ_{Planning}$) et des coûts du système de travail ($SK_{Planning}$).

$$P_{Planning} = f^n (SQ_{Planning}, SK_{Planning}) \text{ où}$$
$$SK_{Planning} = KU_s + KU_b + KO_s + KO_b,$$
$$\text{et} \quad SQ_{Planning} = QD + QC + QP + QR + QE.$$

Les coûts sont définis par la somme des coûts structuraux ($K\text{-}_s$) et comportementaux ($K\text{-}_b$) engendrés par l'utilisateur (U) et par l'ordinateur (O) (cf. chapitre I, paragraphe 2.2.2 où nous précisons ce qu'entendent Dowell et Long par coûts structuraux et comportementaux). :

KU_s = durée moyenne d'exploration du logiciel ;

KU_b = taux de chargement moyen ;
KO_s = lignes de code, nombre d'objets de l'interface ;
KO_b = temps moyen pour produire un plan.

SQ énonce dans quelle mesure le système réalise ses buts. La Qualité de chacun des attributs du système de planification est mesurée de la manière suivante :

QD = proportion moyenne du plan réalisée dans le temps imparti,
QC = nombre moyen d'hommes débarqués par heure
QP = nombre d'erreurs commises (à certains points sensibles du système)
QR = non défini
QE = non défini

Les paramètres ci-dessus sont estimés lors de la spécification de l'interface. Ces estimations seront comparées avec les résultats réels obtenus avec l'utilisateur.

3.1.3. Une méthode formelle : Le Cognitive Walkthrough

Un avantage certain des méthodes formelles est de détecter des anomalies de conception avant la mise en œuvre qui, malgré l'avancée des outils de prototypage, reste coûteuse.

La méthode du Cognitive Walkthrough [Lewis 90] est construite de la manière suivante : tout d'abord le concepteur spécifie une série de tâches qui seront utilisées pour évaluer la conception de l'interface ; ensuite les séquences d'actions que l'utilisateur peut effectuer pour résoudre chacune des tâches sont listées, et, en relation avec chacune de ces actions, le concepteur remplit un questionnaire. Ce questionnaire, dont la structure est donnée à la figure 12, s'appuie sur la théorie CE+.

CE+ est une théorie cognitive de l'apprentissage par l'exploration. Cette théorie s'inscrit dans la continuité de la Théorie de la Complexité Cognitive [Kieras 85] (que nous présentons au paragraphe 3.3.1.1). CE+ décrit l'interprétation que l'utilisateur élabore sur les objets de l'interface afin de l'assister dans la formation de buts et la sélection des actions possibles. L'application de CE+ à la technique du "Cognitive Walkthrough" permet de souligner les aspects de l'interface qui faciliteront la résolution de problèmes ainsi que le processus d'apprentissage.

CE+ Design Walkthrough Evaluateur _____ Date _____
Interface _____ Tâche _____ Etape # _____

Les actions / choix doivent être notés en accord avec le pourcentage d'utilisateur potentiel qui sera succeptible d'avoir un problème: 0= aucun; 1= quelques un; 2 = plus de la moitié; 3 = la plupart.

1. Description du but immédiat de l'utilisateur :
2. La première action atomique / L'action atomique suivante que l'utilisateur doit faire:
   2a. Est-il évident que cette action est disponible? Pourquoi / Pourquoi pas?
   2b. Est-il évident que cette action est appropriée au but? Pourquoi / Pourquoi pas?
3. Comment l'utilisateur va-t-il accéder à la description de l'action?
   3a. Problème d'accès? Pourquoi / Pourquoi pas?
4. Comment l'utilisateur va-t-il associer une description à l'action?
   4a. Problème d'association? Pourquoi / Pourquoi pas?
5. Toutes les autres actions moins appropriées? Pour chacunes d'elles, pourquoi / pourquoi pas?
6. Comment l'utilisateur va-t-il exécuter l'action?
   6a. Problèmes? Pourquoi / Pourquoi pas?
7. S'il y a des timeouts, temps donné à l'utilisateur avant le timeout?
   Pourquoi / Pourquoi pas?
8. Exécution de l'action. Description de la réponse du système:
   8a. Des progrés évidents ont-ils été fait vers le but? Pourquoi / Pourquoi pas?
   8b. L'utilisateur peut-il trouver les informations dont il a besoin dans les réponses du système? Pourquoi / Pourquoi pas?
9. Description des modifications du but, s'il y en a:
   9a. Est-il évident que le but doit changer? Pourquoi / Pourquoi pas?
   9b. Si la tâche est complète, est-ce évident? Pourquoi / Pourquoi pas?

Figure 12 : Le questionnaire pour l'évaluation d'une action simple.

Un questionnaire comme celui de la figure 12 est construit de la manière suivante :

1. Le concepteur donne le but actuel de l'utilisateur et l'action correcte à effectuer (questions 1 et 2).
2. Le concepteur doit ensuite évaluer la facilité avec laquelle l'utilisateur va être amené à choisir cette action et l'exécuter (questions 2 à 7).
3. Le concepteur décrit ensuite la réponse du système et juge sa pertinence (question8).
4. La dernière question (question 9) permet au concepteur d'évaluer la capacité de l'utilisateur à formuler le but suivant ou à détecter si sa tâche est finie.

Cette technique fournit une simulation manuelle de l'activité cognitive de l'utilisateur.

3.1.4. Discussion

Que ce soit l'utilisation des recommandations de Smith et Mosier, les principes de Scapin ou l'interprétation des résultats du Cognitive Walkthrough, l'expérience en la matière de telles interprétations reste la clef de voûte du

diagnostic final. Une connaissance pointue en ergonomie est souvent nécessaire. Bien que ces techniques paraissent simples à mettre en œuvre, elles sont toutefois coûteuses car elles requièrent la présence et la disponibilité d'évaluateurs compétents. Nous rappelons les résultats de Pollier et Nielsen [Pollier 91, Nielsen 90a]. Selon Pollier, un ergonome seul trouve en moyenne 42% des problèmes d'utilisabilité, le regroupement des évaluations de deux ergonomes environ 67% et celui de trois ergonomes environ 85%.

Certaines techniques tentent de sortir d'une évaluation artisanale (au sens de [Long 89]) en proposant un cadre plus rigoureux comme l'utilisation de métriques ou de modèles formels. Mais chaque médaille a son revers. L'utilisation de critères qui mesurent la facilité d'utilisation implique une définition claire et d'y associer des métriques pertinentes. Cette association reste, à l'heure actuelle, un problème difficile pour lequel il n'existe pas de guide performant (cf. technique décrite dans [Colbert 92]). L'approche quantitative permet, par exemple, de calculer des taux d'achèvement de tâche. Ces informations sont utiles pour comparer des performances réalisées avec des interfaces différentes, mais elles ne permettent pas toujours d'expliquer, voire situer, les problèmes rencontrés par l'utilisateur. Quant aux méthodes formelles, spécifier les données nécessaires au modèle requiert une préparation souvent longue et coûteuse ; nous reviendrons sur ce point au paragraphe 3.3.3 lorsque nous présenterons la discussion sur les méthodes formelles d'analyse automatique.

Cette analyse milite en faveur de l'automatisation de l'évaluation. Un premier pas dans cette direction a été franchi par l'automatisation de la capture.

3.2. Techniques avec capture automatique

Nous avons retenu deux systèmes qui effectuent la capture de données comportementales au moyen d'un dispositif extérieur au logiciel testé : Playback et le système d'Hammontree.

3.2.1. Playback

Le système Playback a été développé au début des années 80 dans le service "Human Factor" des laboratoires IBM [Neal 83]. Comme le montre la figure 13, ce système capture les actions clavier au moyen d'un dispositif physique ("lab computer") situé entre le clavier et son système hôte. En mode enregistrement, le "lab computer" détecte, transmet, date et sauvegarde sur disque toutes les

actions de l'utilisateur sur le clavier. En mode "replay", il les transmet au serveur pour reproduire la session de travail originale.

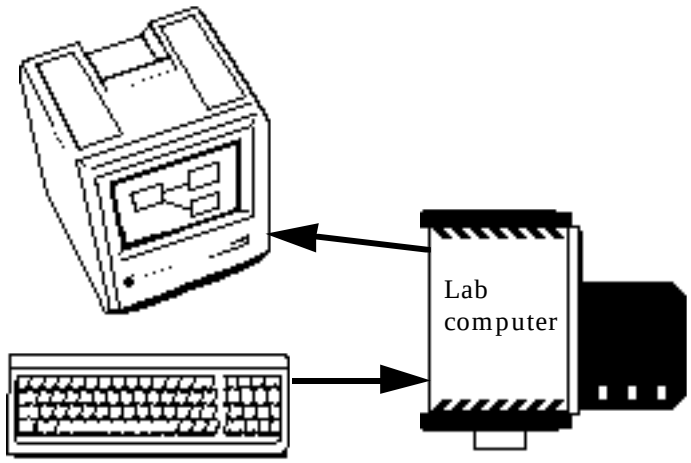


Figure 13 : Configuration du système Playback.

Ce montage a l'avantage de supprimer l'intrusion de la caméra et fournit un fichier de données fiables. Ce fichier peut ensuite alimenter un analyseur statistique simple sur les séquences d'actions : par exemple, nombre de sollicitations de la touche "aide" ou "détruire".

Playback est un système pionnier mais conçu pour des applications limitées à des entrées clavier. Contrairement au système présenté ci-dessous, il ne gère pas les interfaces graphiques à manipulation directe.

3.2.2. Le système de Hammontree

Le système de Hammontree développé sur Macintosh enregistre les actions clavier-souris et complète ces données numériques par un enregistrement audiovisuel analogique [Hammontree 92]. Il comprend :

- un programme de capture d'événements estampillés d'une date. Il s'agit des événements fournis par la boîte à outils du Macintosh.

- un programme de filtrage d'événements qui permet de construire des fichiers de capture personnalisés. Par exemple, un filtre qui extrait toutes les sélections de commande, permet d'élaborer un fichier plus concis (voir en 2.5, la propriété de concision des résultats d'une technique d'évaluation).

- un analyseur multimédia de données qui effectue le lien entre événements et enregistrements audiovisuels par le biais de l'estampille. L'évaluateur localise

et sélectionne un événement et demande à visionner la séquence qui s'y réfère.

- un enregistreur d'annotations verbales utilisable avec l'analyseur multimédia; il numérise la parole et l'associe à des segments de la bande audiovisuelle et à l'événement correspondant.

L'outil fournit une source multiple de données, de traitement et d'analyse pour
1) comparer et évaluer des applications logicielles ainsi que des prototypes,
2) évaluer la documentation de logiciels et le matériel d'instruction,
3) évaluer en ligne les programmes d'entraînement.

3.2.3. Discussion

Les captures externes analogiques (enregistrements audiovisuels) nécessitent un expert humain pour le dépouillement des résultats ; de plus les données brutes, qu'elles proviennent d'enregistrements audiovisuels ou d'une capture numérique, sont difficiles à analyser car souvent trop abondantes. Par exemple, on rapporte que HIMS (Human Interface Monitoring System [Theaker 89]), un outil proposant, à peu de chose près, les mêmes service que celui de Hammontree 92, capture 200 MO d'information à la seconde! Dans ce cas précis, l'image de l'écran n'est pas enregistrée via une caméra audiovisuelle, mais via les signaux vidéos reçus par l'écran.

Ces constatations militent en faveur du traitement automatique de l'analyse du recueil de données. Cette technique est présentée au paragraphe suivant.

3.3. Techniques d'évaluation et analyse automatique

L'automatisation de l'analyse s'envisage selon deux approches complémentaires : l'utilisation de méthodes formelles théoriques et l'approche expérimentale fondée sur la capture logicielle de données comportementales. Dans la première catégorie, nous présenterons CCT, PUM et ICS. MRP viendra illustrer la deuxième tendance.

3.3.1. Méthodes théoriques et formalisées

Les méthodes théoriques et formelles liées à l'automatisation de l'évaluation ergonomique s'appuient toutes sur une représentation de l'activité cognitive de l'utilisateur.

3.3.1.1. La Théorie de la Complexité Cognitive : CCT

CCT se situe dans la lignée de GOMS. Mais contrairement à GOMS qui ne modélise que l'activité experte, CCT permet de comparer les mérites relatifs de plusieurs conceptions en termes de transfert de connaissances et de difficulté d'apprentissage [Kieras 85]. Pour ce faire, Kieras et Polson utilisent, pour une interface donnée, un modèle de la tâche et un modèle formel du processus mental auquel l'utilisateur fait appel dans l'utilisation de cette interface.

- Le modèle du processus mental reprend le modèle de Newell et Simon [Newell 72] qui s'appuie sur un système de production de règles de la forme :

  IF (condition) THEN (action).

- Le modèle de la tâche inspiré de GOMS utilise une décomposition hiérarchique des tâches.

La complexité de l'interface est ici fonction de la quantité, du contenu et de la structure des connaissances requises pour utiliser efficacement l'interface. Elle est évaluée au moyen de métriques tels que le nombre total de règles de production nécessaires à la modélisation de la tâche, le nombre de productions déclenchées pour atteindre un but, etc. Ces métriques sont des mesures prédictives de la difficulté d'apprentissage et du transfert de connaissances.

3.3.1.2. Programmable User Model : PUM

PUM (Programmable User Model) relève du même esprit que la Théorie de la Complexité Cognitive. PUM inclut une architecture cognitive qui permet de prédire l'utilisabilité d'un système en terme de résolution de plan [Young 90]. Pour utiliser PUM, l'évaluateur spécifie ce que l'utilisateur doit savoir sur la tâche et sur l'interface. Cette description est ensuite compilée en règles exécutables par PUM. L'exécution des règles fournit une simulation du comportement cognitif de l'utilisateur en train d'exécuter la tâche avec l'interface spécifiée. Si le simulateur ne trouve pas de plan de résolution pour la tâche donnée, alors l'évaluateur doit déduire l'existence potentielle d'un défaut d'utilisabilité ou une difficulté d'apprentissage.

3.3.1.3. Interacting Cognitive Subsystems : ICS

ICS (Interacting Cognitive Subsystems) peut se voir comme un affinement du Modèle du Processeur Humain [Card 83] qui serait axé sur l'étude du comportement cognitif [Barnard 87]. Dans ICS, l'utilisabilité se mesure, pour l'essentiel, en terme de coût cognitif (alors que les métriques issues du Modèle du Processeur Humain visent plutôt des performances de bas niveau telles la

capacité des mémoires des processeurs, la persistence des informations en mémoire, et les cycles de base des processeurs).

Dans ICS, le sujet humain est modélisé sous forme de neuf sous-systèmes qui fonctionnent en parallèle et qui communiquent via un bus commun de données (voir figure 14). Chaque sous-système est spécialisé dans le traitement d'un type d'information et tous répondent à la même organisation : a) une mémoire locale dans laquelle est recopiée l'information reçue en entrée et qui peut être réutilisée par le sous-système comme donnée d'entrée (ce tampon sert en particulier à la temporisation des informations) ; b) des procédures de traitement qui produisent en sortie des informations de types donnés et qui sont transmises sur le bus commun vers d'autres sous-systèmes spécialisés.

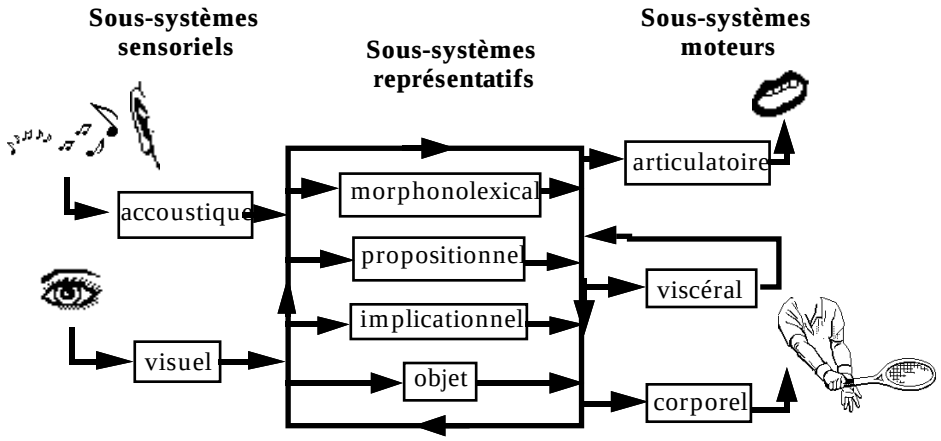


Figure 14 : Schéma du fonctionnement cognitif de l'individu, selon ICS.

On convient de distinguer les sous-systèmes périphériques des sous-systèmes centraux. Les premiers sont de nature sensorielle (acoustique, visuel, tactile) ou bien corporels (articulatoire, visceral, corporel). Les sous-systèmes centraux reflètent des niveaux de traitements cognitifs : les sous-systèmes "objet" et "morphonolexical" modélisent respectivement la structuration de scènes visuelles et d'expressions langagières. Les sous-systèmes propositionel et implicationnel raisonnent au plus haut niveau d'abstraction (représentation de la connaissance et résolution de problème). Le tableau de la figure 8 indique le rôle de chacun des sous-systèmes de ICS.

Sous-système	Description
acoustique	Les données d'entrée de ce sous-système sont les sons. Elles sont ensuite transformées en information pour les sous-systèmes morphonolexical ou implicationnel.
visuel	Les données d'entrée de ce sous-système sont les informations visuelles perçues (caractéristiques physiques d'une scène). Ces données sont ensuite transformées en information pour les sous-systèmes objet ou implicationnel.
morphonolexical	Ce sous-système est spécialisé dans la description structurelle de l'information langagière. Les données de ce sous-système proviennent des sous-systèmes acoustique, propositionnel ou objet. Ces données sont ensuite transformées en données pour les sous-systèmes articulatoire ou propositionnel.
propositionnel	Ce sous-système est spécialisé dans la description sémantique des référents (propriétés et relations). Les données de ce sous-système proviennent des sous-systèmes morphonolexical, implicationnel ou objet. Ces données sont ensuite transformées en données pour ces même sous-systèmes (morphonolexical, implicationnel ou objet).
implicationnel	Ce sous-système est spécialisé dans la composition de schémas de résolution. Les données de ce sous-système proviennent des sous-systèmes acoustique, visuel, propositionnel et physique. Ces données sont ensuite transformées en données pour le sous-système propositionnel et les sous-systèmes corporels.
objet	Ce sous-système est spécialisé dans la représentation spatiale de l'information visuelle perçue. Les données de ce sous-système proviennent des sous-systèmes visuel et propositionnel. Ces données sont ensuite transformées en données pour les sous-systèmes morphonolexical, propositionnel ou corporel.
articulatoire	Ce sous-système est chargé de synthétiser la parole et l'écrit chez l'individu. Les données de ce sous-système proviennent du sous-système morphonolexical, mais aussi du sous-système corporel.
visceral	Ce sous-système est chargé d'interpréter les données provenant de l'état physique intérieur de l'organisme. Les données de ce sous-système proviennent, de plus, du sous-système implicationnel. Ces données sont ensuite transformées en données pour les sous-systèmes articulatoire, corporel et implicationnel.
corporel	Ce sous-système coordonne les mouvements du corps. Les données de ce sous-système proviennent du sous-système objet, mais aussi du sous-système visceral.

Figure 15 : Rôle des sous-systèmes d'ICS.

La théorie ICS et son modèle permettent d'évaluer la charge cognitive d'un individu pour accomplir une tâche au moyen d'une interface donnée. Le principe est le suivant : identifier les sous-systèmes impliqués en notant le suivi des informations dans l'architecture ICS. Par exemple, la lecture d'une phrase sur l'écran met en jeu les sous-systèmes visuel, puis objet, puis morphonolexical, puis propositionnel, puis implicationnel qui décide, par exemple, de l'élimination de la phrase. Le chemin se complexifie si les caractères sont trop petits (plusieurs boucles sont peut être nécessaires pour lire la phrase), si la structure syntaxique est erronée et si la phrase ne fait pas sens. Dès lors, surgissent des boucles de traitement entre les sous-systèmes. Un nombre important de boucles est une façon de prédire une surcharge cognitive.

3.3.2. Où capture et analyse automatique cohabitent : Maximal Repeating Pattern (MRP)

Le principe d'une analyse MRP (Maximal Repeating Pattern) est simple : il s'agit de détecter des répétitions de schémas d'actions [Siochi 91]. Ici, la capture nécessite d'instrumenter le programme source pour capter les actions de l'utilisateur sur les dispositifs d'entrée. Ces actions sont repérées à un bas niveau d'abstraction : par exemple, la sélection d'un bouton ou la saisie d'un champ textuel.

La répétition d'actions peut être considérée comme un révélateur de problème. Par exemple, l'usage fréquent d'une même suite d'actions peut indiquer l'inadéquation de la granularité des commandes et suggérer le besoin d'une macrocommande.

Comme l'indique Siochi dans [Siochi 91], une analyse MRP fournit des enseignements plus riches lorsqu'il s'agit de sujets experts : les intentions sont plus faciles à identifier que dans le cas d'utilisateurs novices dont le comportement au niveau "keystroke" est exploratoire.

3.3.3. Discussion

L'intérêt premier des méthodes formelles d'évaluation tient à leur applicabilité dès l'étape de conception. Les méthodes qui, en plus, sont prolongées d'outils automatiques introduisent dans l'évaluation une rigueur supplémentaire par rapport à l'approche expérimentale. L'inconvénient majeur est, pour l'essentiel, la difficulté d'accès : ICS, PUM, CCT s'adressent à des spécialistes, pas à des implémentateurs. En particulier, il est difficile de s'assurer que le modèle construit est bien le reflet de la réalité.

L'évaluation façon MRP, tout comme EMA, notre propre contribution, s'appuient non plus sur des supposés, mais sur des données réelles. Ces techniques d'évaluation posent des jalons mais n'engendrent pas d'actions correctives. La dernière étape de notre réflexion sur l'état de l'art examine l'automatisation des propositions d'aménagement.

3.4. Techniques avec critique automatique

Les deux systèmes que nous connaissons, KRI et SYNOP, sont des systèmes experts qui offrent une évaluation critique et automatique de la qualité ergonomique d'interfaces Homme-Machine.

3.4.1. Knowledge-based Review of user Interfaces : KRI

KRI (Knowledge-based Review of user Interfaces) s'appuie sur un ensemble de règles (principes ergonomiques) qui, appliquées à la description formelle d'une interface, produisent des commentaires et suggestions de modification [Löwgren 90]. Ce système est organisé comme suit :

- une base de règles modélise le savoir du système en évaluation ergonomique. Ce savoir est construit à partir des connaissances subjectives d'experts ergonomes dont certaines relèvent des recommandations de type Smith et Mosier (cf. paragraphe 3.1.1.1). Pour l'instant, seules les connaissances sur l'organisation spatiale des informations et la syntaxe de l'interface ont été codées ;

- une base de données contient, sous forme textuelle, la retranscription des connaissances codées dans la base de règles. Ce texte est utilisé par KRI pour exprimer son diagnostic ;

- une taxonomie des techniques d'interaction, comme celle de la figure 16, permet au système de structurer de sa base de connaissance en fonction du type d'interface évaluée. Par exemple, la figure 16 présente les choix offerts sous la rubrique "type de dialogue". Le choix, par le concepteur, d'un ou plusieurs des éléments de cette taxonomie permet à KRI de diriger plus efficacement son plan de résolution.

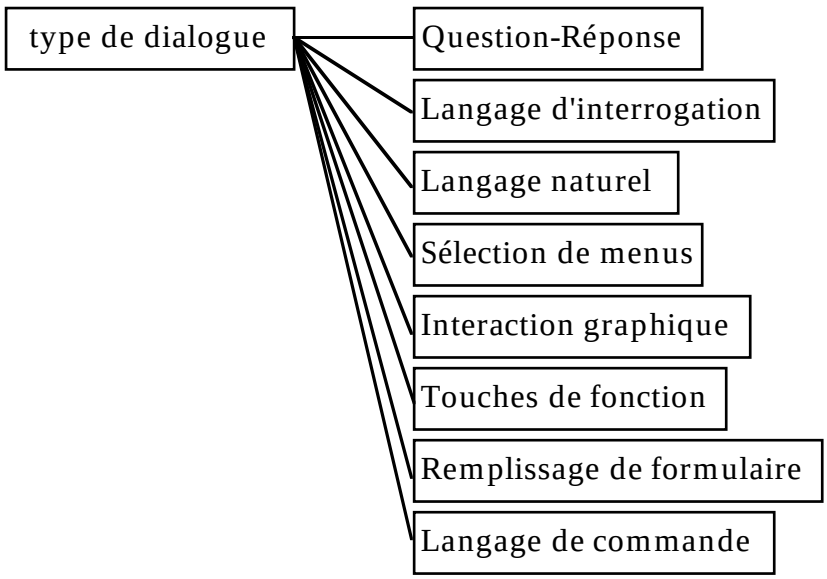


Figure 16 : Extrait de la taxonomie des aspects de l'interface utilisateur.

Une session d'évaluation se déroule en trois temps : chargement de la représentation formelle de l'interface utilisateur, construction du plan d'évaluation en choisissant, dans la taxonomie, les aspects pertinents pour le système étudié, puis diagnostic.

Si l'approche est séduisante, la portée de KRI reste limitée à la critique des interfaces de type remplissage de formulaires et activation de menus via des touches de fonction. Le diagnostic est simpliste, de bas niveau, et ne traite pas la dynamique de l'interaction. Par exemple, "le bouton B1 est trop près du bouton B2" ou "L'ordre proposé ne suit pas l'ordre lexicographique". Dans ces conditions, nous dirons que ce système n'apporte que des éléments d'évaluation figés.

Bien qu'il adopte un fonctionnement différent de KRI, SYNOP [Kolski 89] offre des services similaires.

3.4.2. Le système expert SYNOP

Le système expert SYNOP développé au Laboratoire d'Automatique Industrielle et Humaine de Valenciennes, contribue à l'évaluation statique de vues graphiques pour des systèmes de contrôle. Par évaluation statique, Kolski considère les techniques qui visent à améliorer la présentation de l'information à l'écran : évaluations heuristiques, utilisation de guides d'ergonomie ou enfin, l'approche experte des systèmes à base de connaissance comme SYNOP.

SYNOP est écrit en langage LISP et intègre, dans sa base, des connaissances ergonomiques statiques de présentation d'information. Ces connaissances sont formalisées en règles de production exploitables par un moteur d'inférence. Elles permettent d'évaluer et de modifier automatiquement les vues graphiques. La chaîne logicielle dans laquelle s'intègre SYNOP est représentée schématiquement à la figure 17.

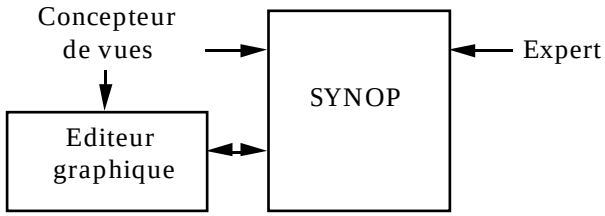


Figure 17 : Chaîne logicielle dans SYNOP.

Le principe d'évaluation et d'amélioration ergonomique d'une vue graphique par SYNOP se décompose en trois phases :

Phase 1 :

Interprétation des fichiers graphiques de la vue à traiter et création d'un réseau sémantique d'objets structurés LISP descriptif ;

Phase 2 :

Evaluation ergonomique des objets du réseau. A cet effet, les règles contenues des sous-bases sélectionnées sont activées. Par exemple, une sous-base, concernant l'ergonomie de présentation des histogrammes, regroupe l'ensemble des règles relatives à ce sous problème. De même, une autre sous-base peut concerner la lisibilité des caractères, les contrastes colorés, etc.

Exemple de règles de la sous-base "les pavés colorés" :

SI	(objet-de-couleur-rouge	x1)
ET SI	(objet-non-utilisé-pour-alarme	x1)
ALORS	(changer-couleur-de-l'objet	x1)
SI	(à-droite-de	x1 x2)
ET SI	(trop-près-de	x1 x2)
ALORS	(écarter-vers-la-droite	x1)

Phase 3 :

Reconstruction des fichiers graphiques correspondant à la vue améliorée et affichage à l'écran de la nouvelle vue.

3.4.3. Discussion

KRI et SYNOP sont, à notre connaissance, les seuls systèmes capables de proposer des actions correctives. Hélas, leur portée est extrêmement ciblée : KRI n'admet que des interfaces de type remplissage de formulaires, et SYNOP est dédié aux applications de type systèmes de contrôle.

De fait, la modélisation de la tâche, ou plus généralement toute activité cognitive d'ordre supérieur, sont ignorés. Les techniques KRI ou SYNOP évaluent l'image de l'interface mais excluent l'utilisateur et les remarques constructives qui découlent de l'observation de ce dernier.

4. Conclusion

Peut-on parler, aujourd'hui, d'automatisation de l'évaluation des interfaces Homme-Machine? La majorité des systèmes actuels ne fournit qu'une aide

partielle : analyse prédictive des performances avec GOMS, analyse “in vivo” des actions avec MRP, simulation cognitive et mini-planification avec PUM, surcharge cognitive prédite ou expliquée avec ICS. Pour tous ces systèmes, la détermination du remède revient au spécialiste, etc. Peu de systèmes sont capables de fournir des critiques constructives à l’exception de KRI et SYNOP qui restent, néanmoins très ciblés dans leur compétence.

Pour clore notre analyse sur l’état de l'art nous présentons, à la figure 18, un synoptique des systèmes étudiés structuré selon les dimensions de notre taxonomie simplifiée. Cette figure souligne l'importance du savoir heuristique dans les méthodes d'évaluation existantes. Il convient également de remarquer que sur les huit méthodes ou techniques automatisées, seul MRP nécessite la présence de l'utilisateur dans le processus d'évaluation.

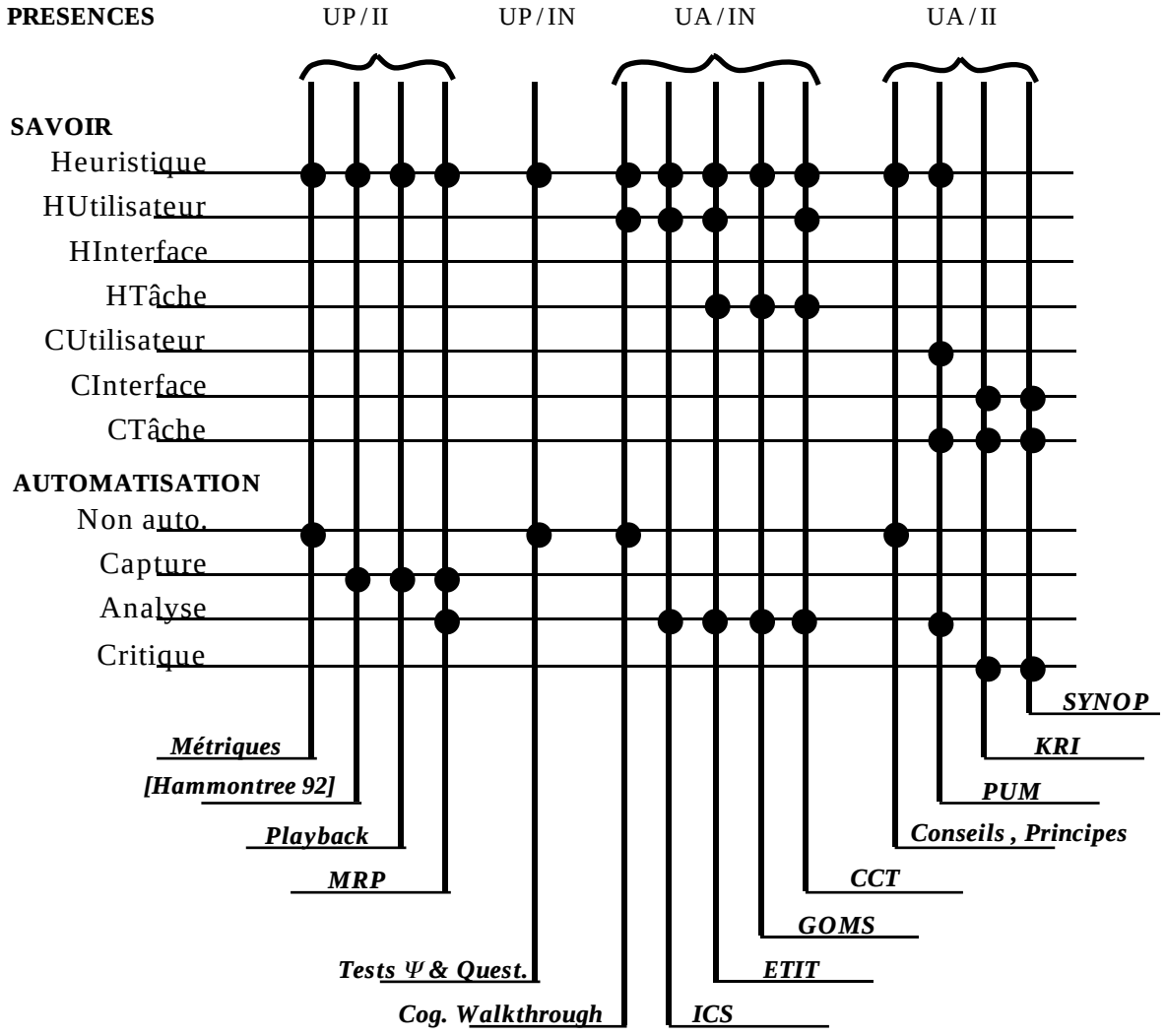


Figure 18 : Vue d'ensemble des systèmes et techniques étudiés. Sur l'axe des présences, le U est utilisé pour “Utilisateur”, le “P” pour “Présent”, le “A” pour “Absent”, le premier “I” pour “Interface”, le “N” pour “Non implémentée” et le second “I” pour “Implémentée”. Sur l'axe du savoir le “H” rappelle “Humain” et le “C” rappelle “Codé”.

Les chapitres II et III que nous venons de présenter nous ont permis de réaliser un tour d'horizon des méthodes et techniques d'évaluation ergonomique des IHM et de préciser le rôle de l'analyse de tâche dans le processus d'évaluation. Nous sommes maintenant en mesure de présenter notre contribution à l'évaluation ergonomique des IHM : ÉMA. Comme le montre la figure 19, ÉMA s'appuie sur une observation directe de l'utilisateur et sur une modélisation de la tâche. Le chapitre suivant présente ce travail en détail.

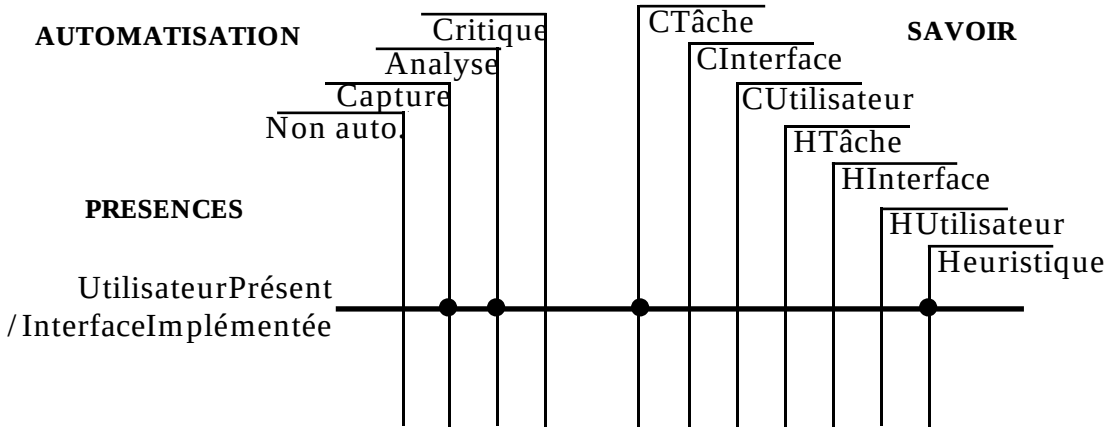
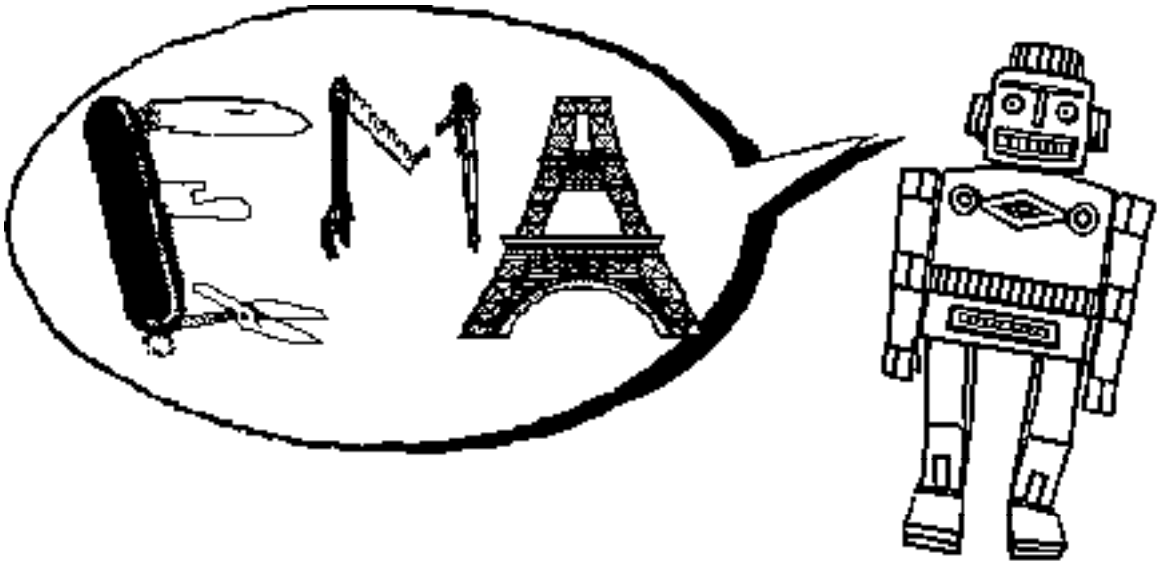


Figure 19 : ÉMA dans notre taxonomie simplifiée.



Partie 2

CONTRIBUTION

Chapitre IV

ÉMA : un mécanisme d'analyse automatique pour l'évaluation ergonomique des interfaces utilisateur

Ce chapitre présente ÉMA, le système que nous avons réalisé comme premier pas vers l'automatisation de l'évaluation ergonomique des interfaces utilisateur. Avant de pénétrer dans les détails de la présentation technique, nous situons ÉMA dans l'espace taxonomique du chapitre III. Nous concluons la discussion en montrant les relations de correspondance entre les capacités d'ÉMA et les principes d'ergonomie de Scapin. Au chapitre V nous montrons comment nous avons validé ÉMA par son application à trois plates-formes pilotes.

1. Positionnement d'ÉMA

ÉMA (Évaluation par Mécanisme Automatique) est un outil qui produit automatiquement une évaluation ergonomique à partir d'une capture de données comportementales numérisées. L'analyse utilise trois sources d'information :

- une base de profils[12] de comportement modélisés sous forme de règles,
- une représentation formelle de l'espace des tâches réalisables avec le logiciel testé,
- les données comportementales enregistrées de manière automatique au cours des sessions d'utilisation du logiciel testé.

Alors que la base de profils est réutilisable, l'espace des tâches est propre à chaque logiciel et les données comportementales dépendent avant tout des sujets observés et des scénarios expérimentaux. L'algorithme d'analyse recherche des profils de comportement dans les données capturées, utilise le modèle de tâche prévue et met en correspondance profil et anomalie. Les anomalies détectées sont issues des principes ergonomiques de Scapin. La figure 1 montre l'enchaînement des différents services d'ÉMA.

[12]En anglais, "pattern"

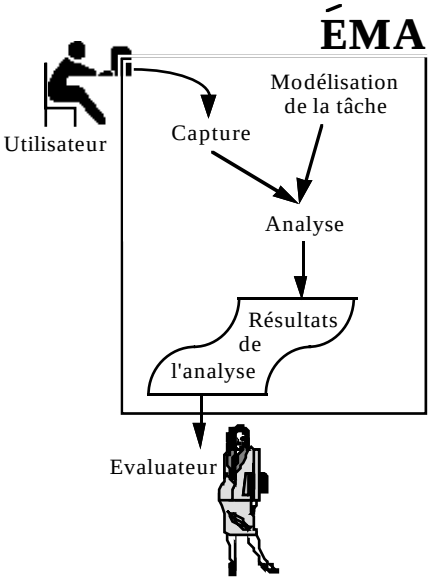


Figure 1 : Fonctionnement d'ÉMA.

La figure 2 situe ÉMA dans le cadre général de Senach [Senach 90] évoqué au chapitre I (cf. chapitre I, figure 11).

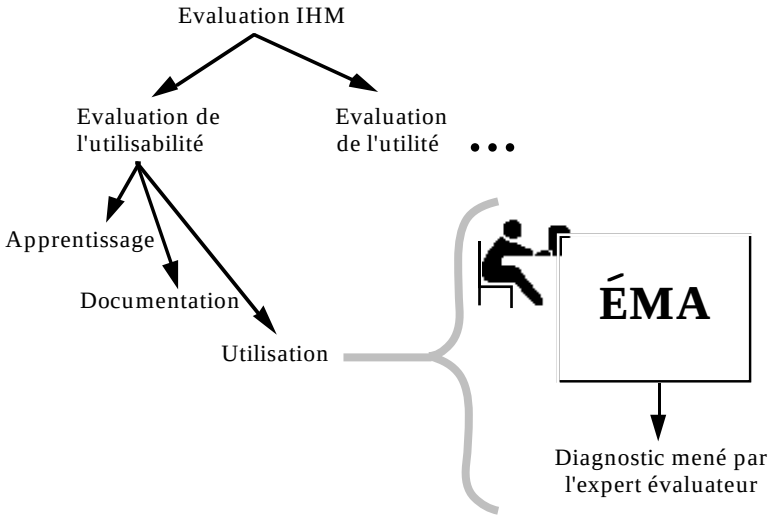


Figure 2 : Positionnement d'ÉMA dans l'évaluation IHM selon [Senach 90].

Après cette présentation introductive, nous étudions les caractéristiques d'ÉMA en nous référant aux cinq dimensions de notre taxonomie (cf. chapitre III) : moyens humains, ressources matérielles, connaissances requises, facteurs situationnels et résultats fournis.

1.1. Moyens humains

Les moyens humains engagés dans l'utilisation d'ÉMA concernent trois catégories d'acteurs : les sujets, les concepteurs, les implémenteurs de l'interface et les ergonomes.

ÉMA est un outil qui relève des techniques d'évaluation expérimentales : il requiert une utilisation effective de l'interface. Les caractéristiques des sujets (l'origine, l'expertise, utilisateur effectif ou représentatif) sont laissées à la discrétion de l'évaluateur.

Le concepteur a la charge de spécifier le modèle de tâche. Par modèle, il faut entendre ici la représentation de la tâche prévue. Nous verrons plus loin dans la discussion technique sur ÉMA, les formalismes adoptés.

L'implémenteur doit instrumenter l'interface à tester. Pour cela, il insère dans son logiciel natif les instructions qui permettront la capture automatique. On relèvera ici une première limitation d'ÉMA : notre système ne fournit pas d'interface logicielle normalisée qui servirait de protocole de connexion universel entre les logiciels à tester et le mécanisme de capture. De fait, l'instrumentation dépend des outils de mise en œuvre de l'interface[13].

L'ergonome est l'évaluateur expert. Il lui revient d'interpréter les résultats d'ÉMA et de les compléter par ses propres analyses heuristiques.

1.2. Ressources matérielles

ÉMA vise le test de logiciels. L'objet de l'évaluation est donc une maquette, un prototype ou un produit. La capture des données nécessite une station de travail dotée de mémoires secondaires. Typiquement, on utilisera la station qui sert de support d'exécution au logiciel testé. Le niveau d'abstraction des informations captées est celui des procédures élémentaires définies au chapitre II (paragraphe 1). Nous reviendrons sur ce point au paragraphe 2.

1.3. Connaissances requises

Nous distinguons deux grandes classes de connaissances : les descriptions nécessaires en entrée au bon fonctionnement de l'outil et l'utilisabilité de l'outil.

En entrée, ÉMA requiert la description du modèle de tâche et demande d'instrumenter le logiciel à tester. La spécification du modèle de tâche et l'instrumentation resteront incontournables tant que les interfaces seront

[13]Cette étude sera entreprise par l'équipe IHM du LGI-IMAG dans le cadre du projet NEIMO [Coutaz 93].

produites au moyen d'outils de bas niveau. On rappelle que les générateurs d'interfaces qui travaillent à partir d'une description conceptuelle à haut niveau d'abstraction disposent des informations nécessaires pour produire automatiquement un modèle de tâche à-la-ÉMA et pour instrumenter le code généré.

Nous n'avons pas mesuré de manière formelle l'utilisabilité d'ÉMA. En l'état, notre outil est une maquette qui vise à cerner l'intérêt et les limites de l'automatisation dans l'évaluation ergonomique des interfaces. Le critère d'utilisabilité de l'outil a été ignoré.

Bien qu'ÉMA vise l'analyse automatique, sa portée, on le verra au paragraphe 2, reste limitée. Dans ces conditions, le travail d'ÉMA doit être complété par l'analyse d'un expert humain. De fait, ÉMA requiert une formation en ergonomie de la même importance que celle utilisée pour les évaluations heuristiques. Une évaluation heuristique fiable repose, on le sait, sur l'intervention de trois à cinq experts. Il serait intéressant de mesurer l'impact de l'analyse fournie par ÉMA sur cette métrique.

1.4. Facteurs situationnels

Les facteurs situationnels recouvrent l'étape dans le processus de développement, le lieu de mise en œuvre de l'évaluation, la typologie des applications et des techniques d'interaction.

La figure 3 illustre l'intervention d'ÉMA dans le processus de développement. Comme dans la figure 13 du chapitre I, nous conservons la distinction entre espace "IHM" et espace "logiciel". ÉMA s'utilise dans l'espace IHM et suppose une approche itérative. Ce processus peut s'inscrire dans un modèle en spirale ou un modèle en V avec retours arrière.

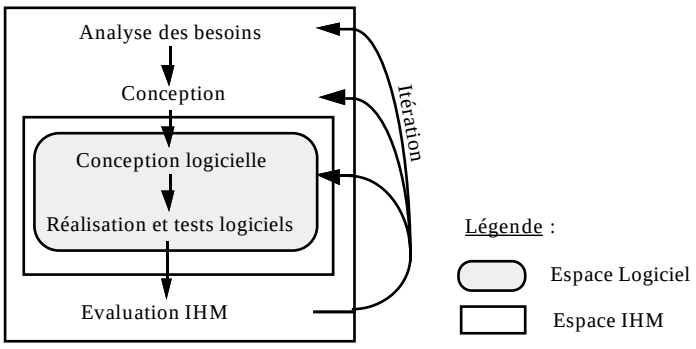


Figure 3 : Cycle de développement adopté.

La capture s'effectue par voie logicielle. Il est donc possible de faire des tests dans l'environnement naturel d'utilisation de l'interface ou en milieu artificiel, c'est-à-dire sur les lieux de développement ou dans un laboratoire d'ergonomie.

Les types d'applications observables par ÉMA se définissent ainsi : les tâches permises peuvent aussi bien relever de la CAO (tâches à structures variables) que du contrôle de procédés (tâches opératives). Toutefois, nous n'avons pas étudié le cas des collecticiels ni celui des interfaces multimodales. Si l'observation des comportements en situation multi-utilisateur exige des techniques nouvelles d'observation, nous estimons qu'ÉMA peut être aisément étendu au cas de l'interaction multimodale.

1.5. Résultats fournis

Les résultats fournis par ÉMA sont des annotations insérées dans le fichier de capture et dans le modèle de la tâche. Le support de restitution est donc informatique. Le niveau d'abstraction considéré est celui de la tâche élémentaire (au sens du chapitre II). Les annotations traduisent des ruptures de cheminement dans l'espace des tâches élémentaires, des annulations ou des répétitions de tâche élémentaire. Ce sont donc des informations objectives et qualitatives (au sens du chapitre III). Elles ne sont pas correctives mais les annotations placées dans l'espace des tâches permettent à l'analyste d'expliquer la cause du comportement.

Ayant présenté les attributs taxonomiques d'ÉMA, nous présentons maintenant ses principes et qualités techniques.

2. ÉMA, présentation technique

Comme le montre la figure 4, le mécanisme d'analyse d'ÉMA s'appuie sur deux composantes : l'une est liée à la tâche, l'autre a trait à l'utilisateur. Dans ce qui suit, nous reprenons en détail chacune des trois facettes d'ÉMA : tâche, utilisateur, analyse.

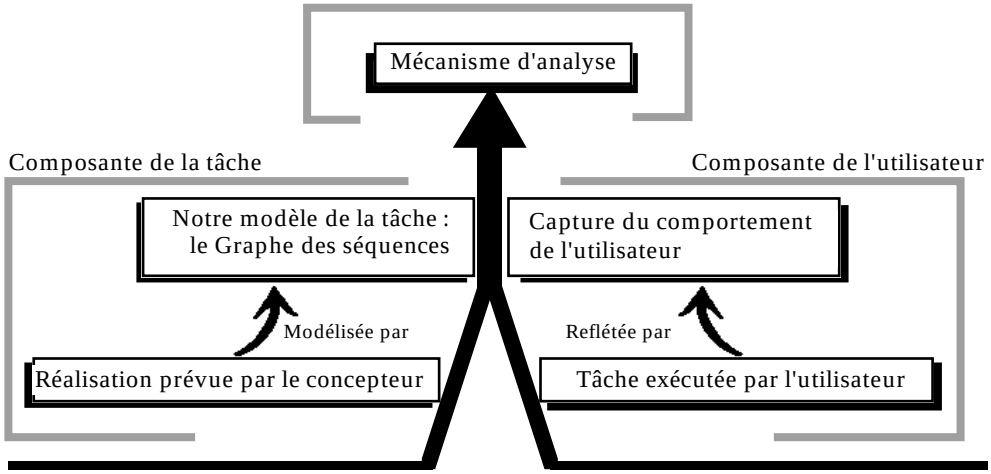


Figure 4 : Les composantes du système ÉMA.

2.1. La composante de la tâche

2.1.1. Les principes

Comme nous l'avons souligné au chapitre II, une tâche se décompose en sous-tâches et une sous-tâche se décompose soit en sous-tâches, soit en tâches élémentaires. Cette décomposition est reprise dans la figure 5.

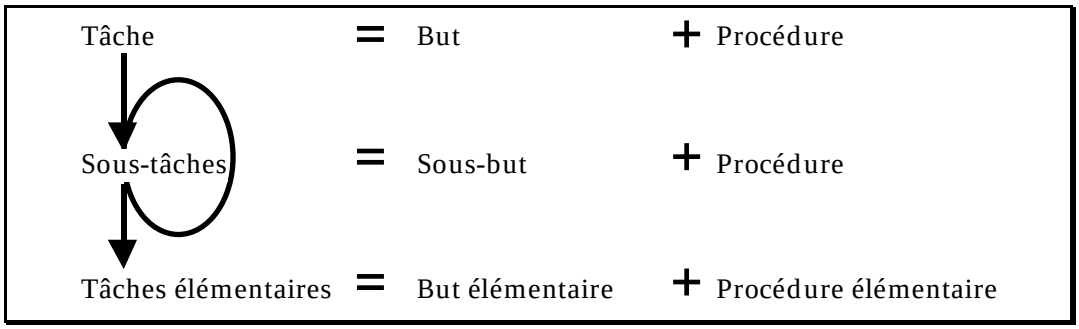


Figure 5 : Décomposition d'une tâche. Les flèches expriment la relation “se décompose en”.

Dans notre analyse de l’état de l’art sur les tâches, nous avons relevé que la plupart des modèles s’appuyait sur une décomposition hiérarchique. Dans ÉMA, nous avons adopté une organisation plate : celle d’un graphe orienté dont les nœuds sont des tâches élémentaires et les arcs des relations d’enchaînement entre tâches. Ce graphe, appelé ***graphe des séquences***, traduit la logique de fonctionnement du système du point de vue du concepteur : il exprime les possibilités de navigation de l’utilisateur dans l’espace des tâches élémentaires.

Une tâche, on le rappelle, est une procédure qui permet d’atteindre un but donné. Dans le cas d’une tâche élémentaire, le but est terminal et la procédure est constituée d’actions. Une action est une opération terminale pour un niveau

d'abstraction donné. Dans ÉMA, le niveau d'abstraction considéré est laissé libre. Il est fixé en concertation avec les évaluateurs et instancié par l'implémenteur chargé de l'instrumentation. Nous verrons au travers des exemples du chapitre V que l'instrumentation peut également être contrainte par les outils de mise en œuvre de l'interface.

Parmi les tâches élémentaires du possible, certaines sont détectées par le mécanisme de capture, d'autres pas. Par exemple, si nous nous plaçons au niveau d'abstraction le plus bas, une action physique résulte de l'utilisation de dispositifs physiques d'entrée. Elle peut se manifester aussi par des actes de parole, des expressions faciales, des gestes, etc. En l'absence de microphone et de caméra, ces actions ne peuvent être perçues par le système d'acquisition. Par exemple, pour l'éditeur de texte vi sous Unix, seules les actions clavier ont une incidence sur l'état du logiciel. En dehors de ces actions, l'utilisateur peut parler, pointer du doigt ou à la souris une zone de l'écran sans effet aucun sur le comportement du logiciel.

La figure 6 montre comment le graphe des séquences modélise l'espace des tâches élémentaires perçues.

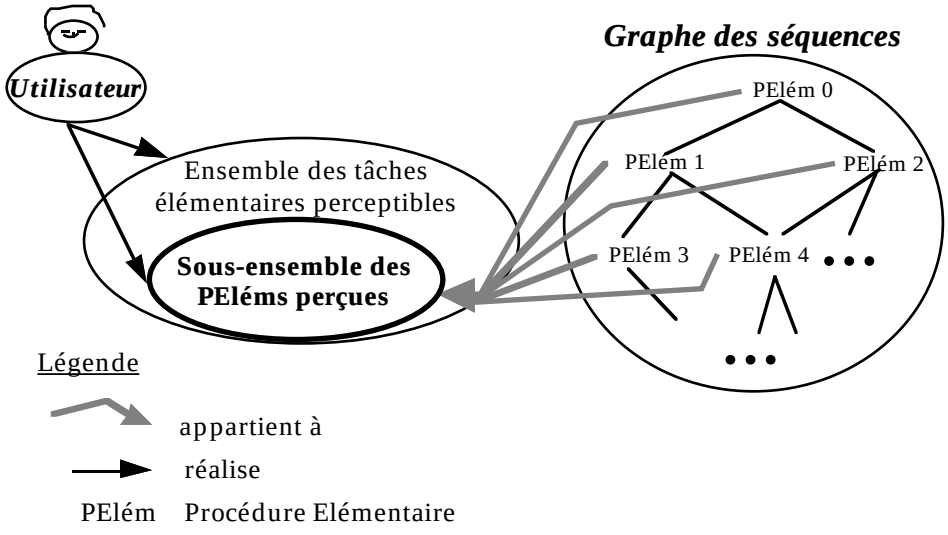


Figure 6 : Procédures concernées par le graphe des séquences.

Nous venons de considérer la nature des nœuds du graphe. Nous en venons aux arcs qui traduisent les relations d'enchaînement entre les tâches élémentaires perçues. Nous avons adopté deux formalismes sémantiquement équivalents : l'un graphique pour la lisibilité, le second textuel proche de l'interprétation machine.

2.1.2. Notation graphique

Le graphe des séquences se représente naturellement par un graphe orienté. L'étiquette d'un nœud désigne une procédure élémentaire ou bien un sous-graphe (cet identificateur sera alors placé dans un rectangle afin de le différencier d'une procédure élémentaire). Il existe deux étiquettes réservées : DEBUT et FIN

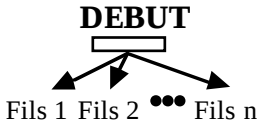
Le nœud **DEBUT** désigne la racine du graphe. La racine encapsule la procédure de lancement du logiciel ou bien la procédure qui donne accès à l'espace des tâches que l'on désire évaluer. L'étiquette **FIN** dénote la sortie du logiciel ou de l'espace des tâches considérées.

Le tableau de la figure 7 présente une description sommaire des relations possibles entre deux nœuds d'un graphe des séquences.

Symbole	Nom	Description
A → **B**	Séquence	il suffit que **A** ait été exécutée une fois pour que ses fils (ici **B**) soient à tout moment accessibles. **A** reste potentiellement accessible.
A ○→ **B**	Obligation	une fois **A** exécutée, la procédure élémentaire suivante est nécessairement l'un des fils de **A** (ici **B**). La relation d'obligation dénote une préemption de la part du système comme dans le cas des boîtes de dialogue modales). **A** reste potentiellement accessible.
A |→ **B**	Restriction	une fois **A** exécutée, ses fils (ici **B**) seront accessibles, mais il ne sera plus possible de revenir sur **A**.

Figure 7 : Relations possibles entre deux tâches élémentaires **A** et **B**.

D'après nos définitions, la racine d'un graphe des séquences est toujours de la forme :

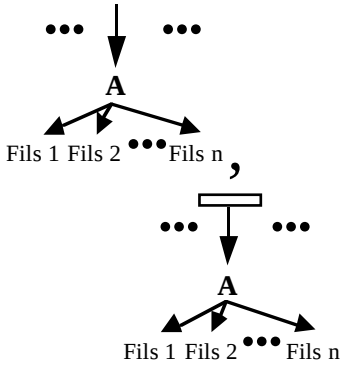
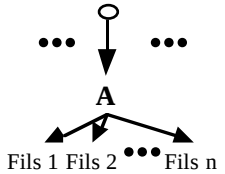


Après une présentation intuitive de la sémantique de nos relations, il convient d'en fournir une définition plus formelle. Pour cela, nous utilisons les notions d'ensemble de procédures élémentaires accessibles (PA) et d'ensemble de procédures élémentaires mémorisées (PM). Soient :

- PAi, l'ensemble des procédures élémentaires que l'utilisateur peut exécuter une fois la procédure élémentaire i exécutée,

- PM_i, l'ensemble des procédures mémorisées pour la procédure élémentaire **i**. Cet ensemble est non vide lorsque **i** est en relation d'obligation avec ses filles. Il regroupe les procédures élémentaires inaccessibles à l'utilisateur après l'exécution de **i** mais qui seront disponibles après l'exécution d'une procédure fille en relation d'obligation avec **i**.

La figure 8 exprime de manière formelle l'évolution de $PA_{\mathbf{A}}$ et $PM_{\mathbf{A}}$ pour chacune des relations possibles de la procédure élémentaire **A** avec son père et ses fils. La première colonne du tableau désigne la relation entre **A** et ses fils tandis que la colonne "Relations avec le parent" montre de manière graphique les combinaisons de relations entre **A** et son père et de **A** avec sa descendance. Nous présenterons ensuite un exemple concret.

Nom de la relation avec la descendance	Relations avec le parent	Evolution de $PA_{\mathbf{A}}$ et $PM_{\mathbf{A}}$
Séquence		$PA_{\mathbf{A}} = PA_{Père} +$ {Fils 1, Fils 2, ..., Fils n} $PM_{\mathbf{A}} = \{\}$
		$PA_{\mathbf{A}} = PM_{Père} +$ {Fils 1, Fils 2, ..., Fils n} $PM_{\mathbf{A}} = \{\}$

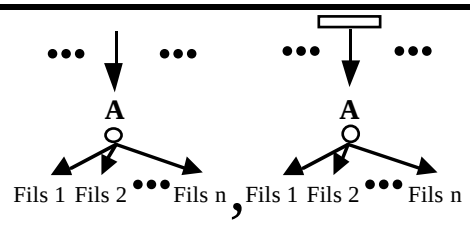
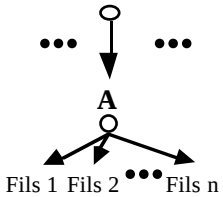
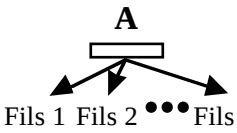
Obligation		$PA_{\mathbf{A}}$ ={Fils 1, Fils 2,..., Fils n} PMA = $PA_{Père}$
		$PA_{\mathbf{A}}$ ={Fils 1, Fils 2,..., Fils n} $PM_{\mathbf{A}}$ = $PM_{Père}$
Restriction		$PA_{\mathbf{A}}$ ={Fils 1, Fils 2,..., Fils n} $PM_{\mathbf{A}}$ = {}

Figure 8 : Implication des relations sur les ensembles PA et PM d'une procédure élémentaire.

Nous illustrons maintenant notre propos au moyen d'un exemple tiré de l'interface utilisateur de Compo (l'une des trois plates-formes de validation d'ÉMA). Cet extrait couvre le choix de la langue de travail ainsi que la procédure de sortie. Au démarrage, Compo demande à l'utilisateur de choisir entre l'anglais, le français et l'allemand (voir la figure 9). La figure 10 illustre l'interface et son message de confirmation dans le cas de la procédure de sortie.

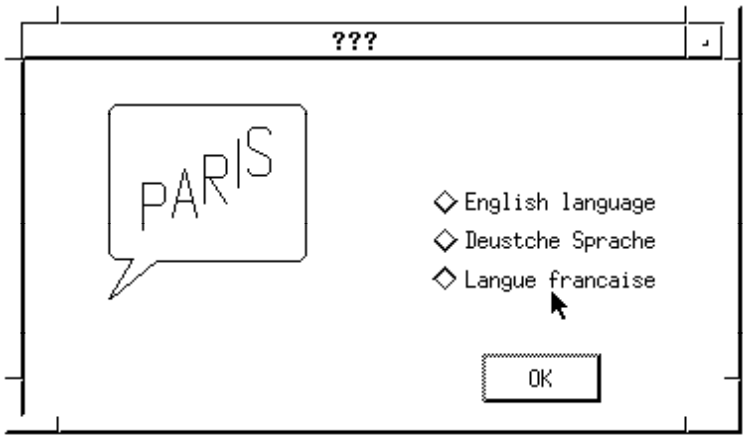


Figure 9 : Compo: sélection de la langue de travail.

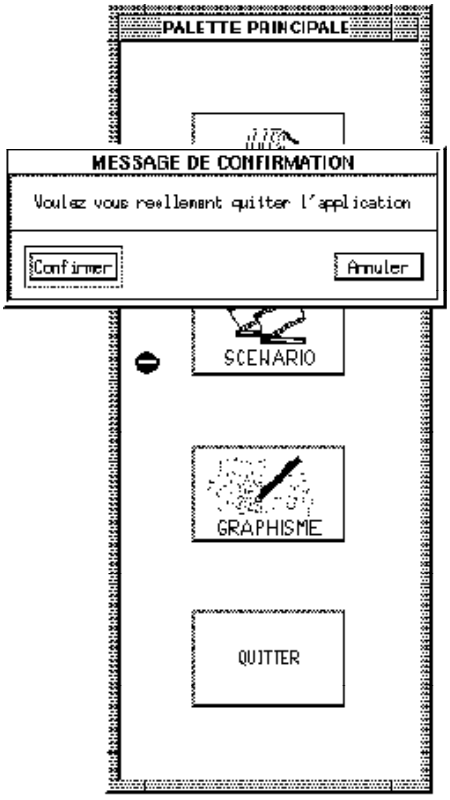


Figure 10 : Compo : demande de confirmation de sortie.

La figure 11 montre le graphe des séquences correspondant à cet exemple. La première procédure élémentaire possible est la sélection d'une langue suivie par la sélection du bouton “OK”. Tant que ce bouton n'est pas sélectionné, l'utilisateur peut modifier son choix ; cette liberté est représentée par la relation de séquence vers la tâche élémentaire “selectionner_OK” (symbole :→). Après la désignation du bouton “OK”, il n’est plus possible de revenir sur le choix de la langue. Cette contrainte se traduit par la relation de restriction entre “selectionner_OK” et sa descendance (symbole :⊢→).

Une fois la langue choisie, l'utilisateur voit apparaître la palette d’icônes de la figure 10. La sélection de l’icône “QUITTER” permet de sortir du logiciel. Les autres icônes donnent accès à de nouveaux espaces de tâches représentés respectivement par les sous-graphes “Palette_APPLICATION”, “Palette_SCENARIO” et “Palette_GRAPHISME”. Si l'utilisateur choisit de quitter le logiciel, il devra répondre à la question “Voulez-vous réellement quitter l'application?” et ne faire que cela (voir la figure 10). Cette contrainte est modélisée dans la figure 11 par une relation d'obligation (symbole :○→). Lorsque l'utilisateur aura confirmé son choix de quitter l'application en sélectionnant “Confirmer”, il ne lui sera plus possible de revenir en arrière (nous retrouvons ici une relation de restriction).

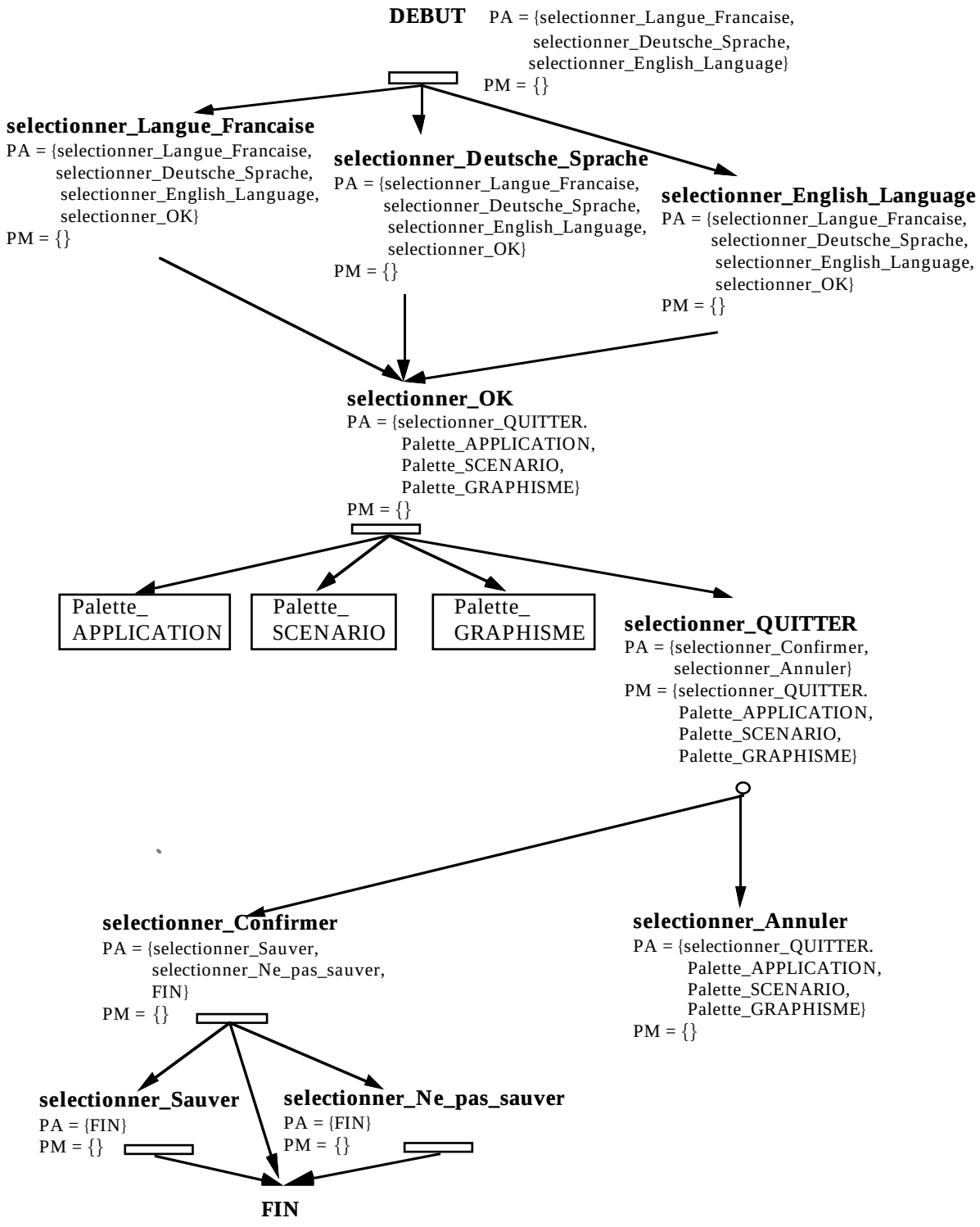
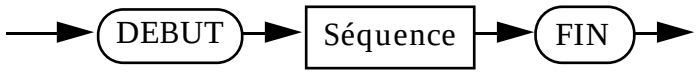


Figure 11 : Graphe des séquences pour l'exemple des figures 9 et 10 décoré en chaque nœud par la valeur de l'ensemble des procédures élémentaires accessibles (PA) et de l'ensemble des procédures élémentaire mémorisées (PM).

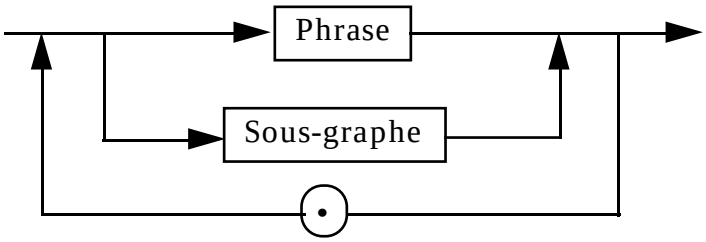
2.1.3. Notation textuelle

Le formalisme textuel permet de traduire une représentation graphique donnée en une expression sémantiquement équivalente et interprétable en machine. Ce formalisme est décrit par les diagrammes syntaxiques ci-dessous. Nous présentons en Annexe I un résumé détaillant l'utilisation de ces diagrammes.

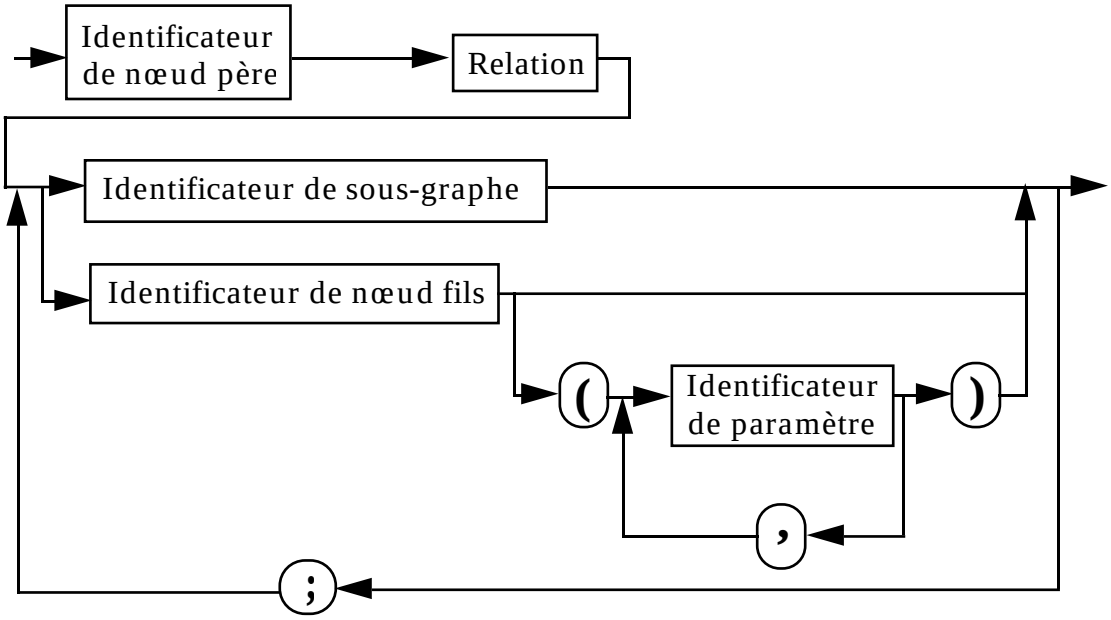
Graphe des séquences ::



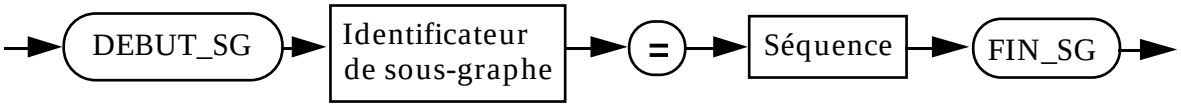
Séquence ::



Phrase ::



Sous-graphe ::



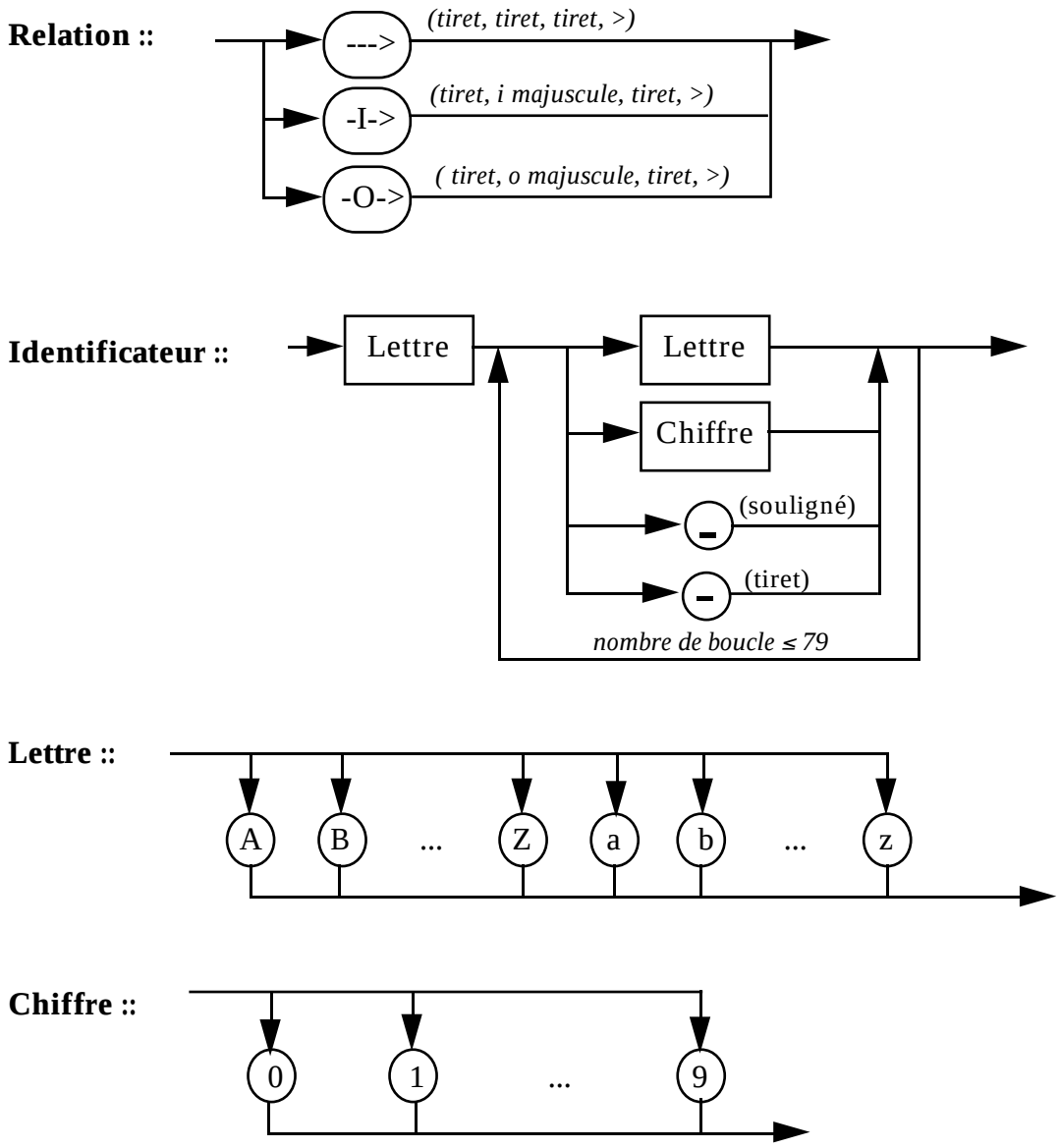


Figure 12 : Diagrammes syntaxiques du graphe des séquences.

Les identificateurs de nœud sont des identificateurs de procédures élémentaires. Une procédure élémentaire est définie par un identificateur unique. Tout identificateur est limité à 80 caractères. En outre, il n'est pas fait de différence entre les majuscules et les minuscules. Ainsi les identificateurs “nom_ident” et “Nom_IDENT” représentent la même procédure élémentaire.

Outre cela, une procédure élémentaire peut être paramétrée. Ce service autorise l'introduction dynamique de valeurs dans le fichier de capture (dont nous présentons l'organisation au paragraphe suivant), lors de l'enregistrement effectué au cours de l'interaction. Un paramètre n'a pas de sémantique propre à ÉMA mais il peut servir de repère pour l'ergonome dans le fichier de capture une fois enrichi des résultats d'analyse d’ÉMA. Par exemple, la procédure

d'ouverture d'un fichier peut être spécifiée dans un graphe des séquences par: “ouverture_fichier (nom)” ou par “ouverture_fichier (chaîne_de_caractères)”, etc. Dans ces conditions, dans le fichier de capture, le nom du fichier effectivement ouvert par le sujet observé apparaîtra en clair.

Les relations de séquence, d'obligation ou de restriction s’expriment ainsi :

- relation de séquence : ---> (tiret, tiret, tiret, >)
- relation de restriction : -I-> (tiret, i majuscule, tiret, >)
- relation d'obligation : -O-> (tiret, o majuscule, tiret, >)

ÉMA accepte des commentaires : toute chaîne de caractères entre (* et *). Les commentaires peuvent être introduits par le concepteur à la spécification du graphe des séquences ou par l’ergonome analyste une fois l’analyse d’ÉMA effectuée.

La figure 13 fournit la représentation textuelle du graphe des séquences de la figure 11.

```
DEBUT          (*commande Unix : poste_de_composition *)
                                  -I->    selectionner_Langue_Francaise;
                                          selectionner_Deutsche_Sprache;
                                          selectionner_English_Langage .

(* procédures élémentaires liées à la sélection de la langue *)
selectionner_Langue_Francaise     --->    selectionner_OK .
selectionner_Deutsche_Sprache     --->    selectionner_OK .
selectionner_English_Language     --->    selectionner_OK .
selectionner_OK                   -I->    (* ss-graphes *)
                                          Palette_APPLICATION ;
                                          Palette_SCENARIO;
                                          Palette_GRAPHISME,

                                          (* proc. élémentaire *)
                                          selectionner_QUITTER .

(* procédures élémentaires liées à la sortie du logiciel *)
selectionner_QUITTER              -O->    selectionner_Confirmer;
                                          selectionner_Annuler.
selectionner_Confirmer            -I->    selectionner_Sauver;
                                          selectionner_Ne_pas_sauver;
                                  FIN .
selectionner_Ne_pas_sauver  -I->  FIN .
```

Figure 13 : Extrait de la représentation textuelle du graphe des séquences de Compo : description du lancement et de la sortie du logiciel.

Dans sa version présente, ÉMA ne fournit pas d'éditeur de spécification des représentations graphiques. Seuls les graphes des séquences entrés sous forme textuelle sont interprétables par ÉMA.

En résumé, le graphe des séquences modélise l'espace des tâches prévues pour un système donné. L'exploration effective de cet espace par le sujet se déduit des procédures élémentaires perçues. C'est à la capture de ces procédures que nous nous intéressons maintenant avec la composante utilisateur d'ÉMA.

2.2. La composante de l'utilisateur

La figure 14 met en évidence le rôle de la composante de l'utilisateur dans le système ÉMA.

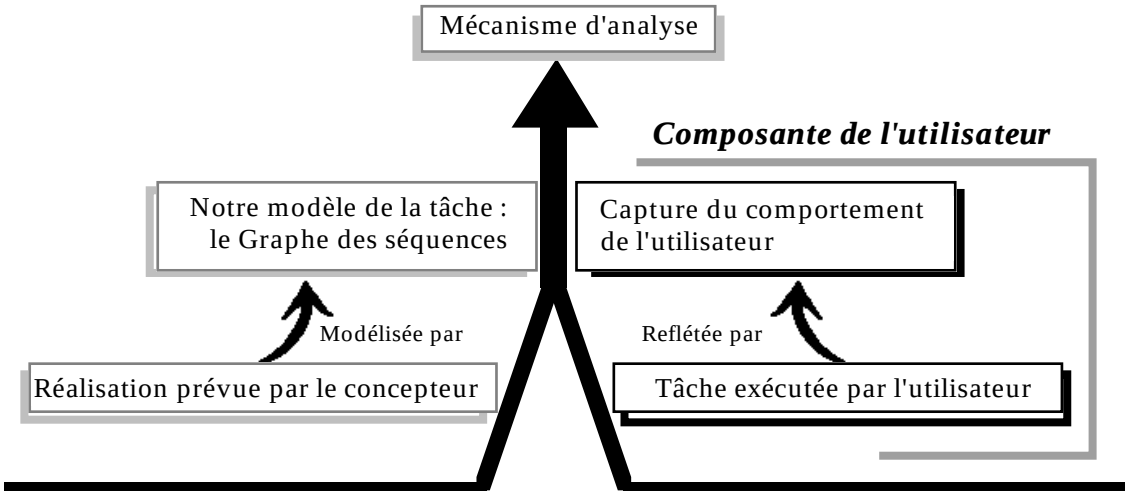


Figure 14 : La composante de l'utilisateur dans ÉMA.

La tâche effective de l'utilisateur trouve son reflet dans l'enregistrement des procédures élémentaires perçues. Puisque ces procédures doivent être observées, nous devons leur trouver un modèle informatique : l'événement. Cette notion fait l'objet du paragraphe suivant. Nous serons alors en mesure de présenter le dispositif technique : le fichier de capture.

2.2.1. Notion d'événement

M.F. Barthet définit un événement comme "un fait dont l'apparition est de nature à déclencher l'exécution de traitement" [Barthet 88]. Nous adoptons cette définition et nous organisons les événements en fonction de leur effet sur les composants logiciels d'un système interactif.

La décomposition première d'un système interactif repose sur la distinction entre le noyau fonctionnel et son interface utilisateur : le noyau fonctionnel

implémente les concepts et fonctions du domaine d'application tandis que l'interface donne l'accès et présente ces concepts et fonctions à l'utilisateur. Cette structuration nous permet de distinguer deux classes d'événement : **[a]** les événements utilisateur et **[b]** les événements internes. Cette représentation est certes schématique mais elle suffit à nos besoins.

[a]

Les événements utilisateur modélisent les actions physiques que l'utilisateur applique sur les dispositifs d'entrée. Ils sont traités par le composant interface utilisateur. Ils se répartissent à leur tour en deux sous-classes :

a.1.

Certains événements ont une incidence uniquement sur l'état de l'interface utilisateur. Ceux-ci correspondent, dans la terminologie de Harrison, à la notion de "commande passive" [Harrison 87]. Par exemple, dans les interfaces graphiques, le déplacement d'une fenêtre est une commande passive : elle est neutre vis-à-vis du noyau fonctionnel.

a.2.

D'autres événements utilisateur entraînent un changement d'état du noyau fonctionnel. Ils traduisent alors l'exécution par l'utilisateur de procédures élémentaires liées au domaine. Par exemple, le double-clic sur une icône pour lancer une application ou le déplacement d'un fichier entre deux dossiers.

[b]

Les événements internes au logiciel interactif circulent entre l'interface utilisateur et le noyau fonctionnel.

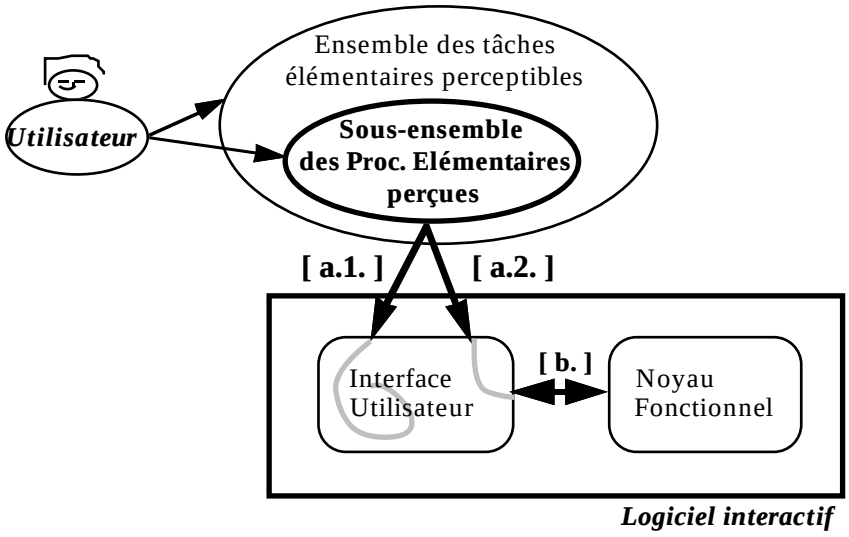


Figure 15 : Relation entre procédures élémentaires, classes d'événements et composants logiciels.

La figure 15 montre la source des événements et leurs effets sur les composants logiciels. En particulier, on notera comment les procédures élémentaires perçues donnent naissance à des événements de classe **[a]**. Ces événements viennent alimenter le fichier de capture.

2.2.2. Structure du fichier de capture

Le fichier de capture obéit à une structure conventionnelle imposée par les traitements d'analyse d'ÉMA. Il est constitué d'une suite ordonnée d'enregistrements. Comme le montre la figure 16, un enregistrement est soit une trace soit un commentaire, c'est-à-dire une chaîne de caractères préfixées par "(*" et suffixées par "*)" et ne comprenant pas de retour chariot.

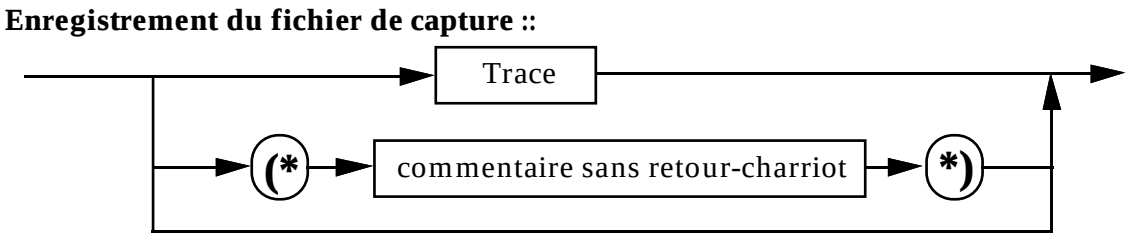


Figure 16 : Diagramme syntaxique d'un enregistrement du fichier de capture.

Une trace est le reflet de l'exécution d'une procédure élémentaire. Une trace est une information estampillée. La figure 17 en explicite la structure :

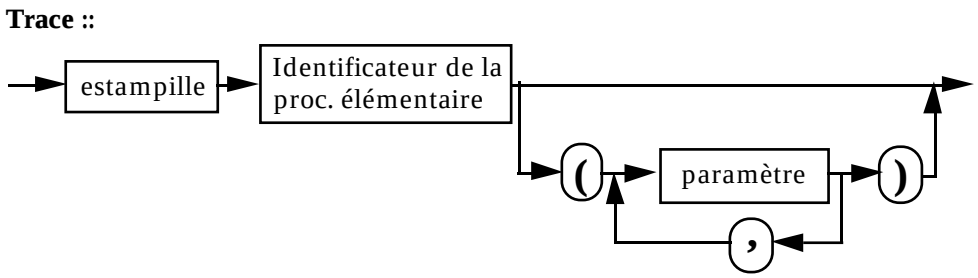


Figure 17 : Diagramme syntaxique d'une trace du fichier de capture.

L'estampille est définie par l'heure absolue d'occurrence de l'événement représentatif de la procédure élémentaire. La figure 18 définit la forme admise.

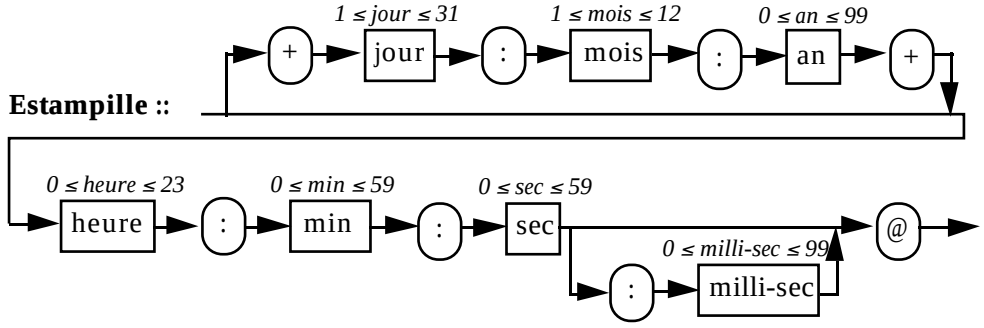


Figure 18 : Diagramme syntaxique d'une estampille du fichier de capture.

La figure 19 présente un extrait du fichier de capture produit par l'utilisation de Compo. Il reflète le cheminement effectif suivant dans l'espace des tâches élémentaires : choix de la langue anglaise, puis choix de la langue française, validation de ce dernier choix, puis choix de l'option Quitter suivi d'une confirmation.

```
+16:06:93+15:02:28@ DEBUT
+16:06:93+15:02:31@ selectionner_English_Language
+16:06:93+15:02:33@ selectionner_Langue_Francaise
+16:06:93+15:02:40@ selectionner_OK
+16:06:93+15:02:53@ selectionner_QUITTER
+16:06:93+15:02:57@ selectionner_Confirmer
+16:06:93+15:03:00@ FIN
```

Figure 19 : Extrait d'un fichier de capture pour le logiciel Compo.

Nous avons défini la structure d'un fichier de capture ainsi que le graphe des séquences. Nous présentons maintenant le mécanisme clef qui autorise l'analyse automatique.

2.3. Le mécanisme d'analyse

La figure 20 situe le mécanisme d'analyse dans le système ÉMA.

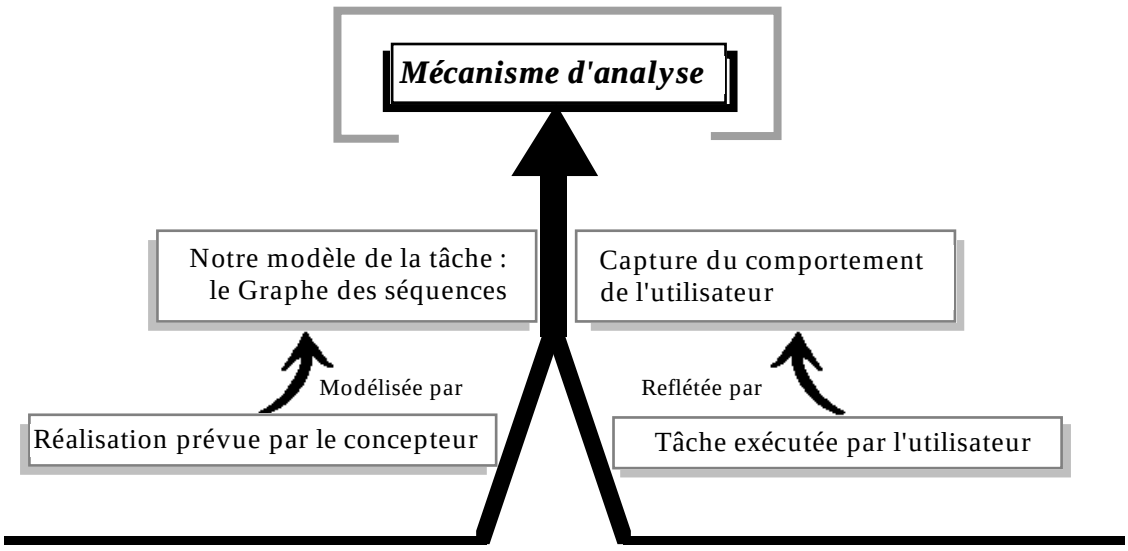


Figure 20 : Le mécanisme d'analyse dans ÉMA.

L'analyseur d'ÉMA utilise en entrée le graphe des séquences et le fichier de capture et produit en sortie un graphe des séquences annoté et un fichier de capture annoté (se reporter à la figure 21).

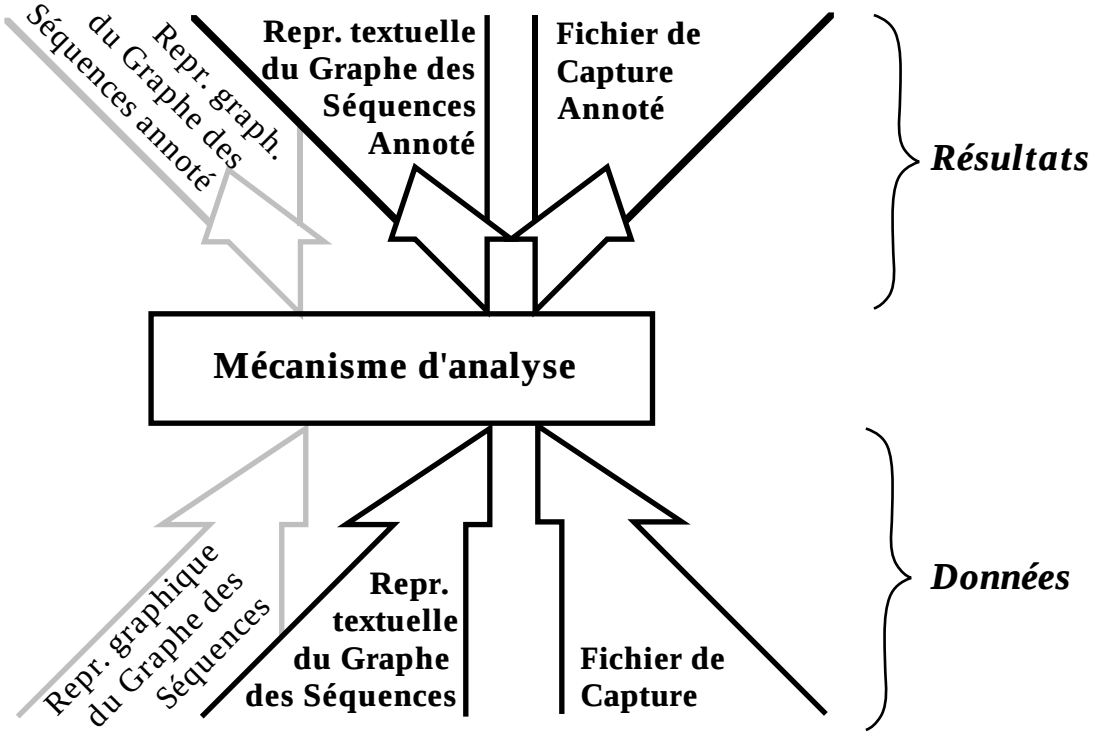


Figure 21 : Les entrées et les sorties du mécanisme d'analyse d'ÉMA. En noir : les implémentations réalisées ; en grisé : celles à venir.

L'analyse s'articule autour de profils de comportement et de règles de détection d'anomalie. Les profils ont été identifiés empiriquement en étudiant des fichiers de capture et en analysant nos propres expériences. A ces profils nous avons fait correspondre des règles heuristiques de détection d'anomalie. Les anomalies sont notées automatiquement à la fois dans le graphe des séquences et dans le fichier de capture. Les annotations, qui sont des commentaires, ont l'avantage d'enrichir les sources d'information de l'analyse tout en restant computationnellement neutres. Nous présentons, en annexes 3 et 4 les fichiers annotés produits pour nos plates-formes d'expérimentation dont nous reviendrons sur le détail au chapitre suivant.

Les règles de détection d'anomalie sont au nombre de quatre : rupture, annulation immédiate, répétition, et invalidité. Nous les présentons successivement.

2.3.1. Règle 1 : Détection de rupture

Nous appelons rupture un changement de direction dans la progression prévue dans le graphe des séquences. Elle traduit une interruption dans la continuité normale de la tâche entreprise.

Une rupture se détecte au moyen du profil de comportement suivant : se trouver dans un nœud autre que feuille et effectuer un retour arrière ou un changement de branche dans le graphe des séquences.

- Dans la représentation graphique du graphe des séquences, une rupture se concrétiserait par une flèche dont l'origine serait sur l'identificateur de la procédure élémentaire qui précède la rupture et la pointe sur le premier identificateur de la procédure élémentaire qui suit la rupture (cf. figure 24). A cette flèche, on associerait un menu déroulant qui montrerait toutes les ruptures survenues en ce point.
- Dans la représentation textuelle du graphe des séquences, une annotation équivalente s'exprime par l'insertion, au niveau du nœud de rupture, du commentaire **(* Rupture *)** et des dates d'occurrence avec en regard l'identification des nœuds suivants.
- Dans le fichier de capture, l'annotation est un simple ajout du commentaire "**(* Rupture *)**" qui signale l'endroit où la détection de rupture est observée (cf. figure 23).

Exemple de détection de rupture

Nous considérons l'utilisation des raccourcis clavier de MSWord dans l'environnement Macintosh. Dans ce logiciel, pour une même commande, les raccourcis par défaut diffèrent en fonction de la langue. Dans ce cadre, nous choisissons l'affichage de la règle de formatage de la figure 22. En version française, il convient d'utiliser la combinaison de touches "⌘=" pour contrôler l'apparition et la disparition de la règle à l'écran. En version anglaise, le raccourci par défaut est "⌘R".

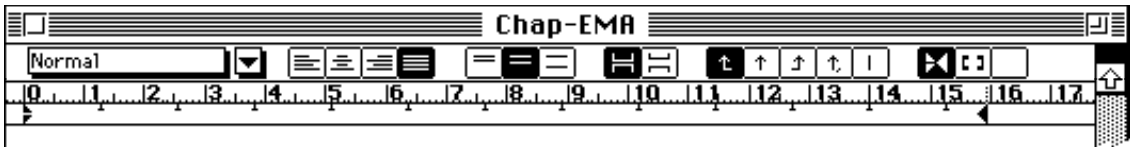


Figure 22 : La règle de formatage de MSWord dans l'environnement Macintosh.

Prenons le cas d'un sujet expert dans l'usage de MSWord en langue anglaise et demandons-lui de contrôler l'affichage de la règle avec MSWord en version française. En raison de son niveau "habilité" au sens de Rasmussen, on peut prévoir qu'il sollicitera les touches "⌘R" (transfert de compétence). Dans la version française cette combinaison active le formulaire de recherche d'une chaîne de caractères. L'utilisateur est alors en situation inattendue : il est forcée à

commettre une rupture L'exemple des figures 23 et 24 illustre ce cas entre la sélection du formulaire de recherche et l'affichage de la règle.

```
14:28:12    selection_formulaire_recherche
14:28:21    selection_affichage_regle
                                          (*Rupture*)
14:28:23    selection_double_interligne
```

Figure 23 : Exemple d'une détection de rupture, au sein du fichier de capture.

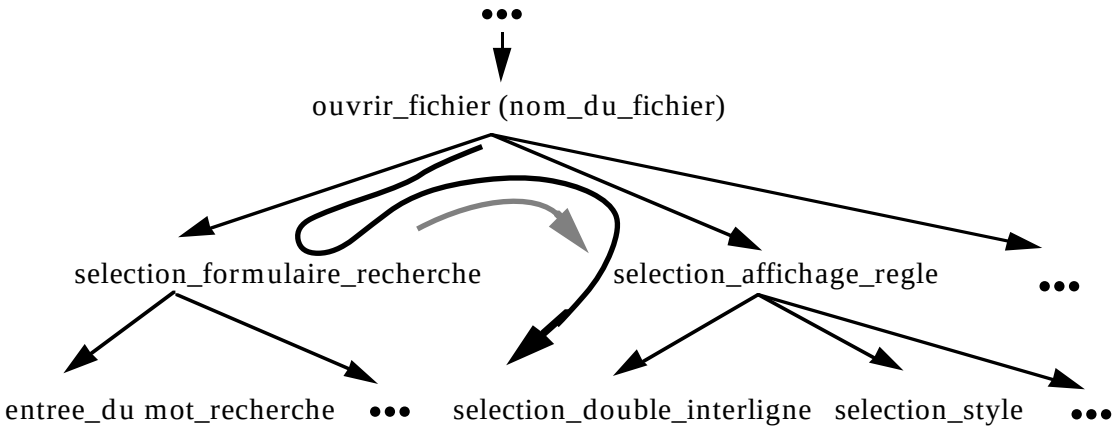


Figure 24 : Exemple de détection d'une rupture dans le graphe des séquences. La flèche en grisé dénote la rupture. La courbe fléchée en trait plein concrétise le cheminement de l'utilisateur dans le graphe.

2.3.2. Règle 2 : Annulation immédiate

L'annulation immédiate est un cas particulier de rupture à la différence près que la tâche élémentaire qui suit la rupture correspond à défaire la tâche élémentaire qui la précède. L'annulation immédiate ne peut être détectée que si le système ÉMA connaît explicitement les procédures élémentaires annulatrices. Nous offrons deux moyens de déclaration de ces procédures :

- La première façon consiste à augmenter la déclaration d'une procédure élémentaire avec la déclaration de la procédure annulatrice correspondante. La figure 25 précise la syntaxe qui devra être employée.

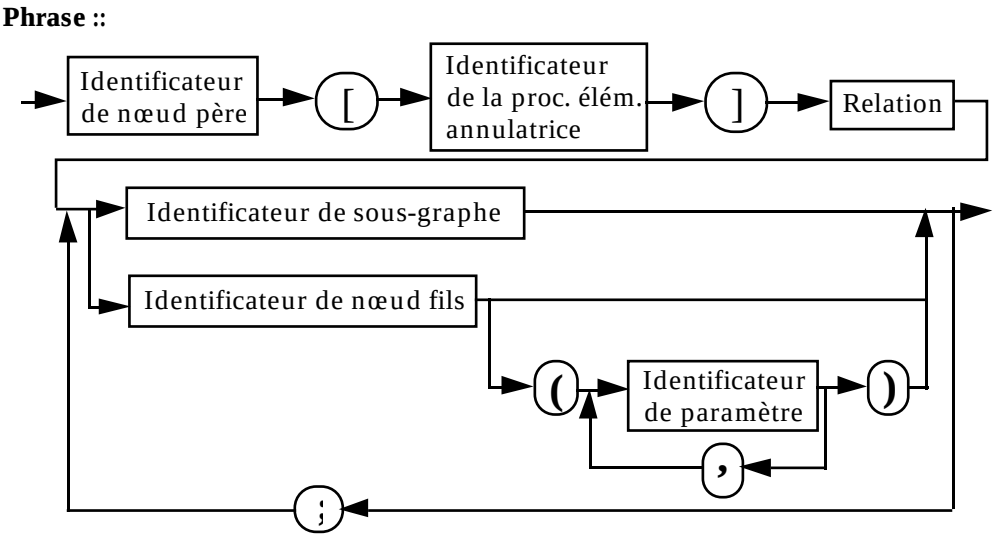


Figure 25 : Diagramme syntaxique permettant la déclaration de procédure élémentaire annulatrice.

- Le second moyen pour déclarer une procédure élémentaire d'annulation est de respecter un format d'identificateur. Un identificateur qui contiendrait une chaîne de caractères conventionnelle, par exemple "Annul" suffirait à désigner une procédure d'annulation immédiate. Il revient alors au système ÉMA de vérifier que les relations de cette procédure avec son parent et sa descendance est compatible avec sa fonction : en particulier, une telle procédure ne peut être engagée dans une relation d'obligation.

La détection d'une annulation immédiate entraîne l'ajout d'une annotation dans le graphe des séquences tout comme dans le fichier de capture :

- Dans la représentation graphique du graphe des séquences, une annulation immédiate pourrait être mise en évidence par la présence du bouton "***Annulation***" à côté de l'identificateur de la procédure annulatrice ; un menu déroulant associé au bouton indiquerait l'heure d'occurrence de chacune des annulations et préciserait pour chacune d'elles l'identificateur de la procédure élémentaire suivante.
- Dans la représentation textuelle du graphe des séquences, une annulation immédiate est dénotée par l'ajout d'un commentaire : **Annulation** complété des dates d'occurrence des annulations avec, en regard, chacun des identificateurs des procédures élémentaires suivant l'annulation (cf. figure 27).
- Dans le fichier de capture, l'annotation est le simple ajout du commentaire "**(* Annulation *)**" qui signale l'endroit où l'annulation immédiate a été observée.

Exemple d'annulation immédiate

Toujours dans l'environnement du Macintosh, nous retenons l'exemple de l'impression d'un document avec changement d'imprimante. Pour cela, il est possible de sélectionner l'option "Imprimer" du menu déroulant "Fichier". La fenêtre modale de la figure 26 est alors disponible.

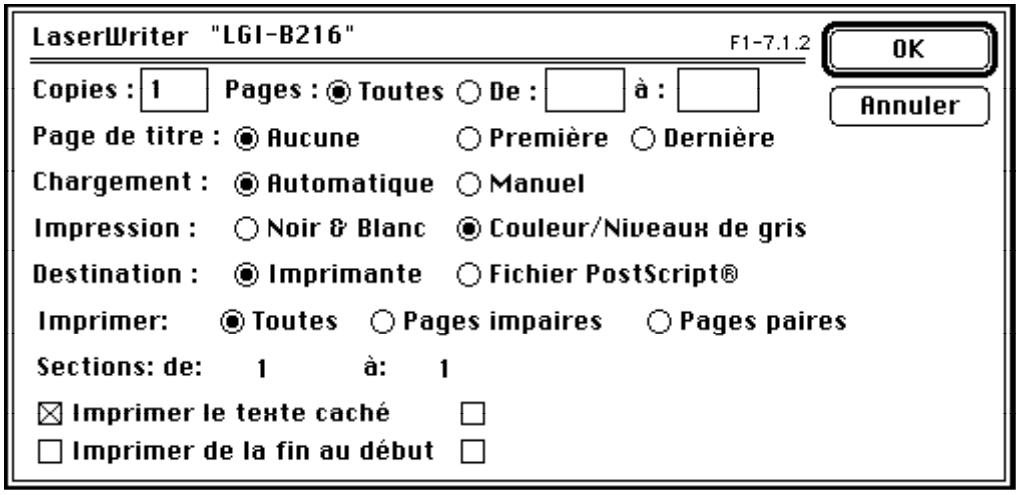


Figure 26 : Illustration d'une annulation immédiate.

L'utilisateur constate que l'imprimante par défaut (ici LGI-B216) n'est pas celle qui convient. La seule façon de spécifier une nouvelle imprimante est d'annuler la demande d'impression. La figure 27 illustre la tâche d'annulation et sa détection dans le graphe des séquences.

```
DEBUT                    -I-> ouvrir_nouveau_fichier,
                              ouvrir_fichier (nom),
                              …
                              selection_menu_pomme_selecteur,
                              … .
ouvrir_fichier           --->  …
                              selection_imprimer,
                              … .
selection_imprimer       --->  selection_chargement (type),
                              selection_page_debut(numero),
                              …
                              selection_annuler
                                 (* Annulation :
                                       08:47:12, suivie de :
                                             08:47:21@ selection_menu_pomme_selecteur
                                       09:01:57, suivie de :
                                             09:02:07@ selection_menu_pomme_selecteur
                                 *)
```

Figure 27 : Exemple d'annotation du graphe des séquences dans le cas d'une annulation immédiate.

2.3.3. Règle 3 : Répétition

L'exécution répétée de procédures élémentaires s'observe dans le fichier de capture. Comme pour les autres anomalies, sa détection serait notée dans la représentation graphique par un bouton **Repetition** dont la sélection ferait apparaître le menu déroulant des dates d'occurrence avec les procédures suivantes. Sa détection est actuellement notée par un commentaire sémantiquement équivalent dans la représentation textuelle, et un simple commentaire **(* Repetition *)** dans le fichier de capture.

Exemple de répétition

Nous prendrons l'exemple de l'éditeur de texte aXe disponible sous OpenWindows. Comme le montre la figure 28, une fois l'éditeur lancé, une petite fenêtre de la taille d'une icône, jaillit à l'écran. Pour accéder au fichier à éditer, il faut cliquer une fois sur le bouton "Edit" pour ouvrir un espace de travail, puis charger le fichier désiré dans cet espace.

Figure 28 : Première fenêtre de l'éditeur aXe, sous l'environnement OpenWindows.

Dans l'environnement OpenWindows, le simple clic est peu utilisé et le double-clic exprime l'ouverture de fichier ou de répertoire, ou encore le lancement d'application. Dans ces conditions, un sujet expert de l'environnement OpenWindows, aura tendance avec l'éditeur aXe d'appliquer un double-clic au bouton-icône "Edit" de la fenêtre initiale. Ce faisant, il ouvrira deux zones de travail. On sera amené à observer une répétition (suivie de la fermeture d'une des zones de travail, qui peut-être définie comme une annulation immédiate).

La figure 29 donne l'extrait de la représentation graphique du graphe des séquences annoté du bouton de répétition et de son menu déroulant.

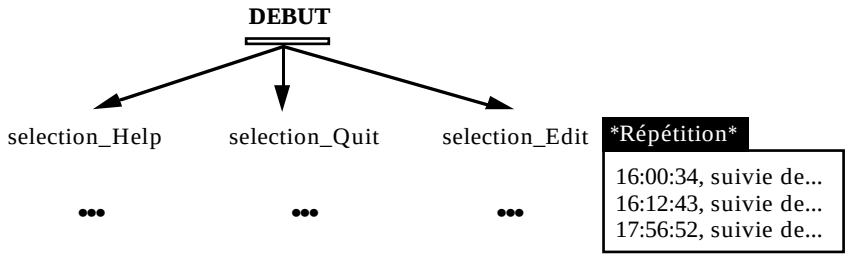


Figure 29 : Exemple de détection d'une répétition dans un graphe des séquences.

2.3..4. Règle 4 : Invalidité

Une procédure élémentaire invalide est une procédure qui ne correspond, au sein du graphe des séquences, a aucune des procédures élémentaires susceptibles d'être exécutées. Ce type de comportement a essentiellement été observé dans des interfaces utilisateur pour lesquelles la structure du dialogue est plutôt dirigée par la machine. Nous verrons un cas concret de ce type d'interaction au chapitre suivant. Les annotations dans le fichier de capture et le graphe des séquences sont semblables aux cas étudiés jusqu'ici.

Exemple d'invalidité

Nous présentons ici une interface qui permet l'accès aux informations d'une base de données médicale : PRIME (Prototype Recherche d'Informations MEdicales) [Cuzin 92]. Cette interface a été générée automatiquement à partir d'un langage de spécification interprété par Look [Deux 92] de l'environnement O_2 [Deux 91]. La fenêtre qui nous intéresse (cf. figure 30) affiche les informations concernant le dossier médical d'un patient. Elle comprend les informations suivantes : deux boutons (la gomme, en haut à gauche, et l'accès aux informations des examens subis par le client : D.M.S. pour Dossier Médical de Spécialité), des champs dans lesquels l'utilisateur peut saisir du texte (carrés à droite de Maladies Chroniques, Allergies, Antécédents et Autres, ainsi que les Données de Venues).

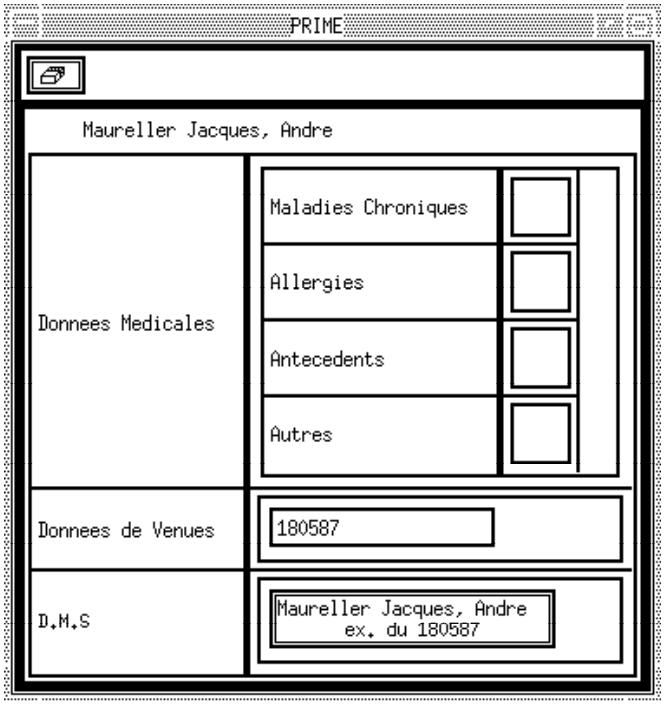


Figure 30 : Affichage du dossier d'un patient dans PRIME.

Un problème observé couramment avec cette interface est l'utilisation du bouton gauche de la souris pour accéder au D.M.S. L'utilisateur attend vainement une réaction du système. En réalité, l'accès au D.M.S. doit se faire via le bouton de droite de la souris. Cette convention vient en contradiction avec la sélection des autres champs qui se pratique toujours avec le bouton gauche de la souris.

Nous observons un ensemble d'invalidités à chaque fois que l'utilisateur cherche à sélectionner ce champ avec un bouton de la souris autre que celui effectif. La représentation graphique du graphe des séquences annoté pour cette portion d'interface est fournie par la figure 31. Sur ce graphe sont annotées les invalidités observées lors de l'accès au D.M.S.

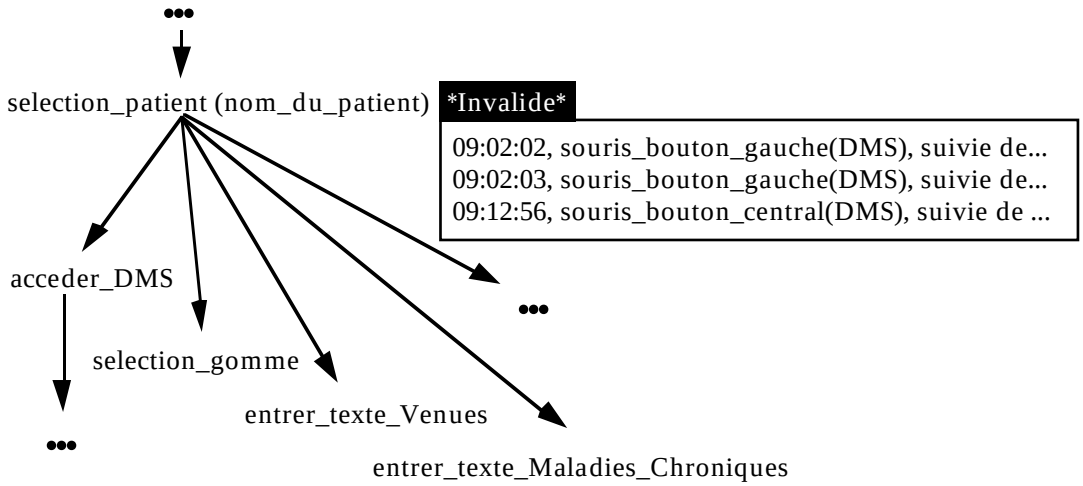


Figure 31 : Exemple de détection, sur la représentation graphique du graphe des séquences, de procédures élémentaires invalides.

3. ÉMA et principes d'ergonomie

Les notions de rupture, d'annulation et de répétition sont des éléments utiles que nous voudrions situer dans un contexte plus large validé par des principes d'ergonomie. Les critères que nous allons considérer sont les recommandations de Scapin [Scapin 90] que nous avons préférées à celles de Nielsen [Nielsen 90b] ou de Gould [Gould 87b] pour leur documentation rigoureuse et explicite.

Dans cette dernière partie, nous identifions le lien entre les valeurs rendues par l'analyse d'ÉMA et le cadre directif de Scapin. Cet exercice ouvre quelques perspectives sur l'intégration dans les services d'ÉMA d'un critique de plus haut niveau d'abstraction motivé par des fondements théoriques de l'ergonomie cognitive.

Les principes directeurs de Scapin s'organisent, on le rappelle, en huit catégories : le guidage, la charge de travail, le contrôle explicite, l'adaptabilité, la gestion des erreurs, l'homogénéité, la signifiance des codes et la compatibilité. Dans ce qui suit, nous éliminerons les tentatives de rapprochement avec l'adaptabilité et le contrôle explicite dans la mesure où l'observation des procédures élémentaires réalisées par l'utilisateur n'apportent pas d'informations suffisantes quant à l'appéciation de ces critères.

3.1. Rupture dans l'espace des tâches

Une rupture indique que l'interface conduit l'utilisateur à une impasse. Cette situation trouve son explication dans le guidage et la compatibilité (au sens de Scapin) :

- Guidage : l'utilisateur s'engage dans une mauvaise voie par manque d'information sur les possibilités qui lui sont offertes (problème d'observabilité de l'état du système [Abowd 92]). Il se retrouve alors dans l'obligation de rebrousser chemin (il pourra revenir à un état antérieur à condition que le système offre des services de réparabilité [Abowd 92]).
- Compatibilité : l'engagement dans une mauvaise voie est dû, ici, à une discordance entre le modèle mental de l'utilisateur et le modèle de tâche du système : l'utilisateur cherche à réutiliser un savoir acquis mais inadapté au cas présent (transfert de connaissance).

Deux autres causes peuvent être rattachées à une détection de rupture, à savoir des problèmes d'homogénéité et de signifiance des codes. En effet, si, pour une même commande, dans deux contextes différents, les procédures à effectuer ne sont plus les mêmes, ou si, les intitulés des commandes ne sont pas clairs, alors ces deux raisons peuvent conduire l'utilisateur à une impasse.

Sur l'exemple qui nous a servi à illustrer la détection de rupture (passage en double interligne dans MSWord), il s'agissait d'un problème de compatibilité entre le modèle mental de l'utilisateur et le modèle d'utilisation de l'interface Homme-Machine. En effet, pour l'utilisateur, l'affichage de la règle s'effectue par le biais du raccourci clavier ⌘R, alors que dans l'interface, c'est la combinaison ⌘= qu'il convient de produire.

3.2. Annulation immédiate

L'annulation immédiate traduit généralement un problème de navigation. Cela peut-être la cause d'une charge de travail importante, d'un problème d'homogénéité, d'une mauvaise signifiance des codes ou d'un problème de compatibilité.

- Charge de travail : si l'utilisateur doit mémoriser un trop grand nombre de procédures élémentaires, cela peut induire l'exécution d'une procédure à la place d'une autre, l'annulation étant alors le moyen de revenir dans l'état précédent du système.
- Homogénéité : une annulation immédiate peut aussi provenir d'un trouble de l'utilisateur induit par la nécessité d'utiliser des moyens différents pour arriver au même résultat.
- Signifiance des codes : une opération mal nommée faussera le choix de l'utilisateur et l'emmènera sur une fausse piste.
- Compatibilité : de même, une confusion induite par un savoir acquis sur un autre système amène à un état du système non souhaité, état dont l'utilisateur se sortira grâce à l'annulation de la dernière procédure exécutée.

Une annulation immédiate peut provenir, dans une moindre mesure, de problèmes de guidage. L'utilisateur n'ayant pas l'information pertinente, il exécute une procédure puis l'annule, le résultat observé n'étant pas celui escompté.

L'exemple que nous avons retenu pour illustrer l'annulation est le choix d'une imprimante dans une tâche d'impression de document (voir paragraphe 2.3.2). Ce problème souligne essentiellement une lacune de guidage : lorsque l'utilisateur édite un document, la variable d'état du système qui mémorise l'identité de l'imprimante attachée au service d'impression n'est pas observable. Il faut fouiner dans l'état du système ou bien comme dans le cas de notre sujet, abandonner la tâche en cours.

3.3. Répétition

L'exécution répétée d'une même procédure élémentaire peut traduire un défaut de granularité couplée à l'absence de macro-commandes ou, comme dans Eager, à l'absence d'un service de détection automatique de tâche répétitive [Cypher 91]. Ici, nous retiendrons la surcharge cognitive du sujet ou un problème dans le retour d'information.

L'absence de retour d'information alors que le sujet s'attend à une réaction de la part du système ou encore l'instabilité du temps de réponse peuvent entraîner des répétitions de tâche élémentaires : l'utilisateur insiste pour "bien faire comprendre à la machine" ce qu'il entend faire. Ici, les répétitions mettent en avant notamment des problèmes de guidage.

Une répétition peut aussi être induite par une mauvaise gestion d'erreurs, une homogénéité mal conduite ou un problème de compatibilité entre le modèle d'utilisation du système et les expériences précédentes d'utilisation d'interfaces Homme-Machine de l'utilisateur. L'exemple de l'éditeur aXe présenté en 2.3.3 illustre ce cas en raison du transfert du double-clic dans un contexte ou le simple clic fait sens pour le système.

3.4. Invalidité

L'observation de procédures élémentaires invalides dénote un problème d'observabilité de l'état du système, ou l'absence de retour d'information proactif [Sellen 92]. Dans l'espace de Scapin, on parlera de problèmes de guidage et d'homogénéité. L'exemple de l'usage de la souris pour le système PRIME est à ce titre révélateur.

Mais aussi, et pour une moindre part, une charge de travail trop importante, des problèmes sur la signifiance des codes ou de compatibilité entre le modèle mental de l'utilisateur et modèle du dialogue de l'interface sont à l'origine d'invalidités observées.

3.5. En synthèse

Le tableau de correspondance de la figure 32 résume nos réflexion sur la correspondance entre les valeurs rendues par notre système et le cadre à fondement ergonomique des principes de Scapin.

- Un ▬ à l'intersection de la ligne i et de la colonne j signifie que la règle de comportement de la ligne i permet ***rarement*** la détection d'un problème relevant du principe ergonomique de la colonne j.
- Un ○ à l'intersection de la ligne i et de la colonne j signifie que la règle de comportement de la ligne i permet ***quelquefois*** la détection d'un problème relevant du principe ergonomique de la colonne j.

- Un + à l'intersection de la ligne i et de la colonne j signifie que la règle de comportement de la ligne i permet ***souvent*** la détection d'un problème relevant du principe ergonomique de la colonne j.

Principes ergonomiques / Règles	Guidage	Charge de travail	Gestion des erreurs	Homogénéité	Signifiance des codes	Compatibilité
Détection de rupture	+	—	—	○	○	+
Annulation immédiate	○	+	—	+	+	+
Répétition	+	—	○	○	—	○
Invalidité	+	○	—	+	○	○

Figure 32 : Correspondance entre règles de comportement et principes ergonomiques.

Il est certain que l'on pourrait trouver une justification pour au moins un ○ dans chacune des cases. Ici, nous reflétons simplement notre interprétation des jeux de critères ergonomiques de [Scapin 90] en fonction des résultats que nous avons observés lors de l'application de nos règles de comportement. Nous présentons ces résultats au chapitre suivant.

4. Conclusion

ÉMA est un premier pas vers l'automatisation de l'évaluation ergonomique des interfaces utilisateur. Si comme Giroux on distingue l'utilité potentielle d'un système de l'usage réel qui en est fait [Giroux 87], ÉMA modélise l'utilité potentielle d'un système par le graphe des séquences et l'usage réel est reflété par le fichier de capture. Notre analyseur automatique s'appuie sur une base de règles de comportement qui autorise la comparaison entre l'utilité potentielle et l'usage réel pour produire des mesures en termes de rupture, de répétition, d'erreur d'activation et d'annulation de tâches élémentaires. A l'évidence, une prochaine étape est la considération de tâches composées.

De manière générale, les techniques d'évaluation qui s'appuient sur une capture automatique du comportement de l'utilisateur ne considèrent que l'usage réel du système. Tel est le cas notamment de MRP (Maximal Repeating Pattern [Siochi 91]) présentée au chapitre III et limitée à la détection des répétitions. Sans prétendre à l'exhaustivité, ÉMA va plus loin en ouvrant le champ d'investigation à d'autres profils de comportement intéressants, comme les ruptures et les annulations.

Dans le chapitre qui suit nous complétons la présentation des principes d'ÉMA par l'exposé de nos pratiques expérimentales.

Chapitre V

ÉMA : Etudes de cas

Ce chapitre présente l'application d'ÉMA aux trois systèmes qui nous ont servi à valider notre contribution technique :

- Compo, un générateur d'interfaces spécialisées,
- les DAB, deux simulations de distributeur automatique de billets de banque, et
- Médiathèque, un logiciel d'interrogation de base de données documentaires.

Pour chacune des plates-formes retenues, nous décrivons de manière informelle l'utilisation de l'interface Homme-Machine ; nous modélisons ensuite chaque interface par son graphe des séquences et nous présentons les résultats de l'analyse d'ÉMA. Dans le cas de Compo, qui est un logiciel de plus de 300 000 lignes et riche sur le plan fonctionnel, le graphe des séquences ne concerne que les tâches dont l'utilisabilité semblait problématique. Le graphe est complet pour les deux autres systèmes.

Nous justifions maintenant le choix de ces trois plates-formes et montrons comment leur complémentarité définit la couverture d'ÉMA sur l'axe "type d'application" du chapitre III.

1. Justification du choix des plates-formes d'expérimentation

Dans la taxonomie du chapitre III sur les méthodes et techniques d'évaluation, nous avions affiné l'axe "facteurs situationnels" en sous-espaces parmi lesquels nous relevions la dimension "type d'application". Nous définissions brièvement cette notion en fonction de la nature des tâches et de la catégorie d'interface utilisateur (cf. paragraphe 2.4 du chapitre III). Nous souhaitons ici gagner en précision en considérant la souplesse admise dans la structure de la tâche, la possession de l'initiative de l'interaction, la typologie de l'interface en terme de langage et de dispositif au sens de Nigay [Nigay 94] :

- La structure de la tâche peut être rigide, souple ou mixte. Une structure souple s'applique à une tâche de conception. Cette dernière se caractérise, en général, par l'absence d'ordre strict et stable des sous-tâches qui la composent. Par opposition aux tâches dont la structure est rigide, une tâche de conception révèle un comportement opportuniste [Hayes-Roth 79] qui se manifeste par l'entrelacement de tâches. Un exemple de tâche structurée est le pilotage d'avion [Hoshstrasser 89] alors que la mise en page d'un journal relève d'une activité de conception.

- L'initiative de l'interaction revient à l'utilisateur ou bien au système ou encore aux deux acteurs de manière alternée. En principe, le choix entre ces possibilités est lié à la structure de la tâche. En pratique, la capacité du contrôle de l'interaction dépend étroitement des outils de mise en œuvre [Sukaviriya 93, Wanner 89]. Résultat : trop de systèmes de recherche d'information imposent des structures de dialogue dirigées par la machine (de type questions/réponses) alors que l'activité de recherche d'information ne se calque pas nécessairement sur une structure rigide.

- La typologie des interfaces est encore mal cernée. Toutefois, on trouvera dans [Nigay 94] un espace taxonomique qui s'appuie sur les notions de langage d'interaction et de dispositif physique. Un langage d'interaction (d'entrée ou de sortie) est un langage que l'utilisateur ou le système emploient pour échanger de l'information. Un langage définit un ensemble d'expressions bien formées, c'est-à-dire un assemblage conventionnel de symboles qui fait sens. Par exemple, un langage naturel, un langage gestuel, un langage graphique. Un dispositif physique (d'entrée ou de sortie), tels une souris, un écran, un microphone, est un capteur ou un émetteur qui permet de produire les expressions d'un ou plusieurs langages d'interaction. Par exemple, via un microphone ou un clavier, il est possible de transmettre les expressions d'un langage naturel.

Le tableau de la figure 1 montre la complémentarité des plates-formes retenues. Nous observons une bonne couverture pour la structuration des tâches et l'initiative de l'interaction : Compo offre une structure souple alors que le DAB et Médiathèque imposent une structure rigide. Dans Compo, l'initiative de l'interaction revient à l'utilisateur tandis que Médiathèque contrôle le dialogue. Comme nous l'avions annoncé au chapitre IV (localisation d'ÉMA dans notre espace taxonomique), les langages d'interaction et les dispositifs d'entrée/sortie de nos plates-formes de test sont usuels : manipulation directe, langage de commande, écran-clavier-souris.

	Structure de la tâche	Dialogue dirigé par	Langage d'interaction en entrée	Médias utilisés en entrée
Compo	Souple	Utilisateur	Manipulation directe	Clavier Souris
DAB	Rigide	Mixte	Manipulation directe	Souris
Médiathèque	Rigide	Système	Langage de commande	Clavier

Figure 1 : Tableau comparatif des systèmes retenus en fonction de la structure de la tâche, de l'initiative de l'interaction, des langages d'interaction et des dispositifs d'entrée. (Par souci simplificateur, les modalités de sortie ont été ignorées).

Nous présentons maintenant chacune de ces trois plates-formes ainsi que les résultats observés.

2. Compo

Le système auteur Compo a été créé voici quatre ans par l'équipe Interface Homme-Machine du Laboratoire de Génie Informatique dans le cadre d'une collaboration avec Renault-Recherche. Ce projet est toujours en cours de développement. Compo a été la première interface utilisateur à laquelle nous nous sommes intéressée dans le cadre de notre étude sur l'évaluation ergonomique des interfaces Homme-Machine.

2.1. Présentation générale

Compo est un environnement auteur pour la conception d'applications interactives embarquées dans des véhicules de tourisme. Ce système s'adresse à des concepteurs d'interface éventuellement non informaticiens (par exemple, des ergonomes).

L'Application que le concepteur construit est un ensemble structuré de concepts. En effet, une Application embarquée est constituée d'un ensemble de Scénarios, par exemple un Scénario "aide à la navigation", un Scénario "réglages", etc. A son tour, un Scénario comprend des Groupements d'Objets, qui correspondent approximativement à la notion d'une page écran. Par exemple, le Scénario "réglages" inclut un Groupement d'Objets pour régler le chauffage, un autre pour régler l'autoradio, etc. Chaque Objet d'un Groupement représente un concept du domaine de la voiture (un thermomètre moteur, une jauge essence, etc.). Cette hiérarchie est reprise figure 2 où sont situés en italique des exemples d'Application, de Scénarios ou de Groupements.

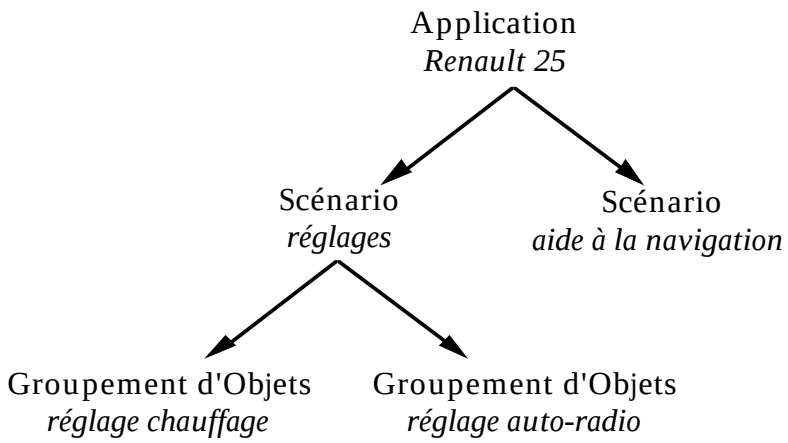


Figure 2 : L'organisation hiérarchique des concepts manipulables dans Compo.

L'interface utilisateur de Compo doit permettre au concepteur d'applications embarquées de manipuler (observer, dessiner, programmer) les concepts d'Application, de Scénario, de Groupement et d'Objet gérées par Compo. Pour ce faire, chacun des concepts est accessible via différentes vues : vue structurelle, vue logique et vue graphique.

- La vue structurelle d'un concept présente, sous forme iconique, les sous-concepts qui la composent. Par exemple, la vue structurelle d'un Groupement montre les Objets qui le constituent (sur la figure 3 : le Groupement "curseur1" est constitué de l'Objet "QuelleJauge"). Cette figure présente en premier plan la vue structurelle du Groupement "curseur1", et en arrière plans, celle du Scénario dans lequel "curseur1" est impliqué : "pap_c1", puis la vue structurelle de l'Application à laquelle "pap_c1" appartient : "DemoPape" elle-même retouvable à partir de la "PALETTE DES APPLICATIONS"existantes. Les relations père-fils entre les objets sont concrétisées par un chemin d'icônes situé dans le bandeau de chaque fenêtre. La vue structurelle d'un Groupement permet d'ajouter ou de retirer un Objet de ce Groupement.

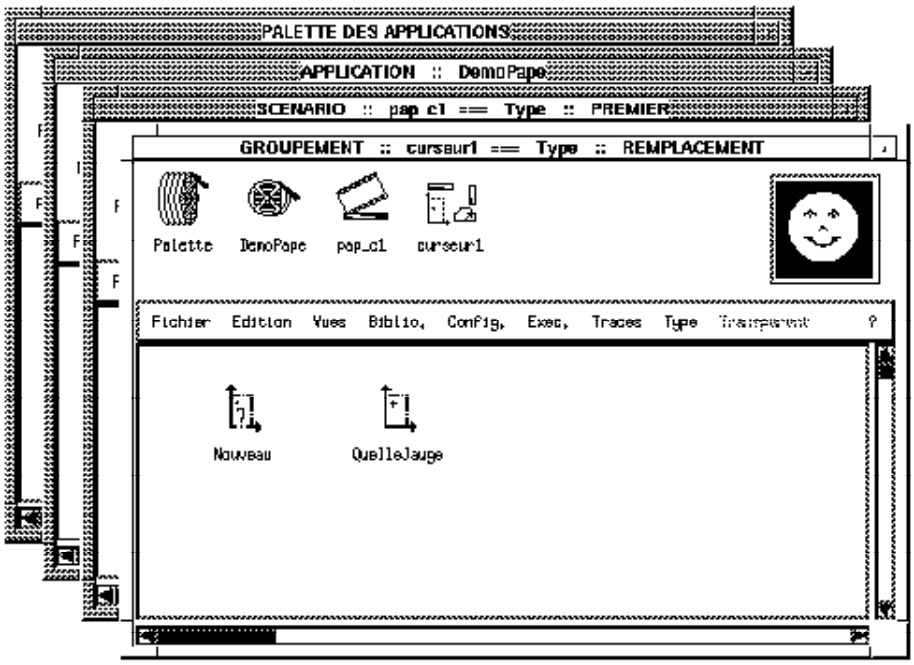


Figure 3 : Vue structurelle d'un Groupement dans Compo. Dans cet exemple, le groupement "curseur1" est constitué d'un objet "QuelleJauge". Le chemin iconique du bandeau de la fenêtre indique que "curseur1" appartient au scénario "pap-c1", lui-même composant de l'application "DemoPape". En cliquant sur l'une de ces icônes, l'utilisateur a accès direct à la vue structurelle correspondante.

- La vue logique montre le script qui spécifie le comportement dynamique du concept. Pour un Groupement, ce sont, par exemple, les enchaînements entre les Groupements ou le changement de couleur d'un objet. A cette fin, un

langage événementiel de type iconique a été conçu [Chabert 91]. La figure 4 présente l'espace de définition de ces scripts. La vue logique permet d'éditer les scripts.

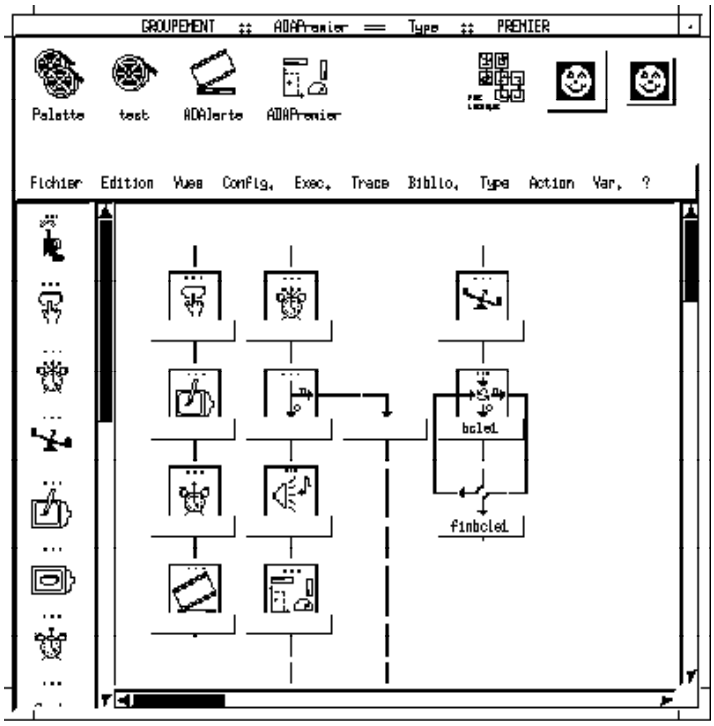


Figure 4 : Vue logique d'un Groupement. A gauche, la palette des instructions iconiques. A droite, un exemple de script. Un script se construit par manipulation directe.

- La vue graphique d'un concept contient sa définition graphique. Par exemple, la vue graphique d'un Groupement montre la page écran telle qu'elle apparaîtra dans le véhicule. Une vue graphique permet de dessiner à façon les formes des Objets et décider des couleurs. Un exemple est donné par la figure 5 qui présente l'ébauche d'une jauge.

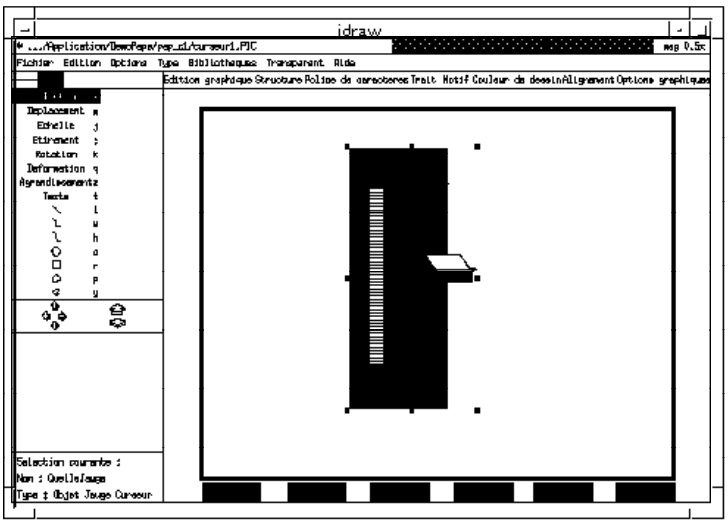


Figure 5 : Vue graphique d'un Groupement.

Compo offre un terrain d'expérience pour l'observation de tâches de conception (ici, la conception d'interfaces embarquées). Pour une description complète de l'interface du logiciel Compo, le lecteur intéressé pourra se reporter à [Collet 93].

2.2. Les procédures élémentaires considérées pour le graphe des séquences

Dans le cas de Compo, la génération du graphe des séquence a été partielle. En effet, l'évaluation menée autour de Compo s'est déroulée en parallèle avec le développement d'ÉMA. La capture automatique y a été mise en œuvre, et l'analyse a été manuelle, suivant une évaluation de type heuristique. Seuls les problèmes détectés ont été l'objet d'une comparaison avec le graphe des séquences. Aussi, seuls les extraits significatifs par rapport aux problèmes qui ont été soulevés par l'analyse sont donnés directement avec les exemples auxquels nous ferons référence au paragraphe 4.4.

Dans Compo, nous captons à la fois des événements de la classe [a.1] et [a.2], c'est-à-dire des événements qui correspondent à des commandes (ou procédures élémentaires) passives ou à des procédures qui changent l'état du noyau fonctionnel (cf. Chapitre IV, paragraphe 2.2.1). Dans notre expérimentation, les procédures élémentaires perçues sont les suivantes :

- Les procédures élémentaires qui n'entraînent aucune modification du noyau fonctionnel de Compo sont essentiellement les modifications des aspects de la présentation des informations (retaillage, iconification, déplacements, etc., des diverses fenêtres).

- Les procédures élémentaires qui entraînent des modifications du noyau fonctionnel sont les sélections d'objet et les choix dans les menus.

La figure 6 donne un exemple de procédures élémentaires perçues pour l'interface utilisateur de Compo.

```
Selectionner_application (nom)
Selection_menu_appli_Fichier-Ouvrir_selection
Selection_menu_appli_Exec-Regler--Vitesse
Double_click_sur (nom)
Scroll_vertical (fenetre)
Deplacement_fenetre (fenetre)
etc.
```

Figure 6 : Exemples de procédures élémentaires de Compo captées par notre mécanisme.

2.3. Mise en œuvre de la capture

Le logiciel Compo se décompose en trois parties : la navigation, l'éditeur de langage iconique et l'éditeur graphique. La navigation concerne les accès aux différentes vues (structurelles, graphiques ou logiques), ainsi qu'aux passages d'un concept à un autre (Application, Scénario, Groupement, etc.). La totalité du code représente quelques 300.000 lignes écrites en langage C (pour la navigation et l'éditeur de langage iconique) et C++ (pour l'éditeur graphique). Aussi, dans un premier temps, nous avons choisi de restreindre la capture aux seules procédures élémentaires qui permette de naviguer dans l'interface.

Compo s'exécute au-dessus de l'environnemnt X Window. Nous sommes concernés par trois des principes directeurs de X :

- les services d'ordre supérieur ;
- la distinction entre les fonctions de gestion (le serveur X) et la présentation des ressources (le "window manager") ;
- la notion de procédure Callback.

• Par définition, les fonctions offertes par le serveur sont élémentaires. Aussi, des objets de présentation d'ordre supérieur, appelés widgets, (menus, barres de défilement, boutons, etc.), sont-ils fournis dans des boîtes à outils développées au-dessus du serveur. Pour Compo, nous avons choisi Motif. Un widget de classe topLevel, désigne une fenêtre reconnue par le window manager. Le contenu d'un topLevel peut être un ensemble de widgets mais ceux-ci ne sont pas reconnus par le window manager. La figure 7 montre un exemple de topLevel décoré par mwm.

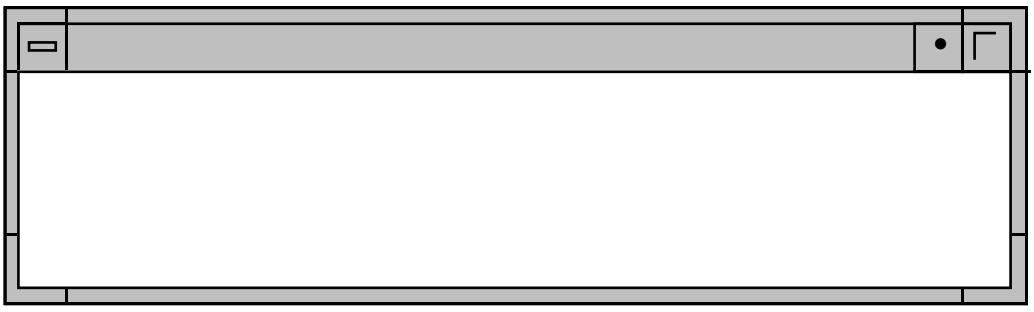


Figure 7 : Un topLevel et ses décorations mwm.

• Concernant la séparation des rôles, le serveur X se charge, par exemple, de régler les recouvrements des fenêtres, tandis que les éléments décoratifs de ces fenêtres (bouton de fermeture, bandeau de titre, etc.) et la façon de les

manipuler sont fixés par le “window manager” choisi par l’utilisateur. Pour Compo, le “window manager” mwm (Motif Window Manager) a été choisi.

- La technique des procédures Callback est une façon de réaliser des systèmes réactifs. A un événement est associé un traitement. Ce traitement est encapsulé dans une procédure déclenchée automatiquement à l’occurrence de l’événement.

2.3.1. Technique utilisée

La capture que nous avons réalisée et dont nous présentons un extrait en annexe 2, ne reflète pas directement la capture telle qu'elle est définie pour ÉMA : elle a été implémentée bien avant la solution actuelle. Mais la description adoptée est très proche du formalisme présent. En vérité, ce formalisme découle de cette première expérience.

L'annexe 2 présente les aspects techniques de l'implémentation de la capture. De manière générale, la dichotomie des fonctions entre serveur ou window manager, et Callback, distingue les procédures élémentaires qui n'entraînent pas de changement dans le noyau fonctionnel de celles qui y répercutent un changement.

- Les procédures élémentaires qui ne modifient pas l’état du noyau fonctionnel de Compo sont identifiées à partir des événements captés par mwm et par certains widgets Motif comme les barres de défilement. Le tableau de la figure 8 explicite l’effet, sur les éléments décoratifs que mwm ajoute à un widget topLevel, de quelques-unes des procédures élémentaires perçues.

Procédure élémentaire	Actions physiques	Effet
Deplacement_fenetre (fenetre)	sur la barre du haut, glissement avec le bouton gauche de la souris enfoncé	déplacement de la fenêtre
Iconification_fenetre (fenetre)	clic souris avec le bouton gauche sur 	iconification
Retaillage_fenetre (fenetre)	clic souris avec le bouton gauche sur 	agrandissement plein écran de la fenêtre OU retour à une fenêtre de taille normale
Selection_fenetre (fenetre)	clic souris avec le bouton gauche dans la fenêtre	passage en avant-plan de la fenêtre

Figure 8 : Effet de quelques procédures élémentaires sur les éléments décoratifs de mwm.

- Chacune des procédures élémentaires qui entraîne une modification du noyau fonctionnel fait appel à une Callback. Aussi c'est à l'entrée de chacune des Callback que s'insèrent quelques lignes de codes permettant la capture.

2.3.2. Le recueil des données

Six utilisateurs ont été amenés à utiliser le logiciel Compo. Le degré d'expertise dans l'utilisation de Compo de ces utilisateurs se répartit comme suit : deux experts, un occasionnel, et trois novices.

Ces six utilisateurs sont des étudiants ou des ingénieurs de recherche travaillant pour l'équipe IHM du LGI. (Si l'on se reporte à notre taxonomie du chapitre III sur les techniques d'évaluation, nous avons choisi des sujets représentatifs d'origine locale et l'expérimentation a eu lieu in situ.)

Nous avons demandé à chaque utilisateur d'accéder à la vue graphique de l'Objet "QuelleJauge" appartenant au Groupement "curseur1" lui-même élément du Scénario "pap-c1" qui compose l'Application "demopape" (cf. figure 9).

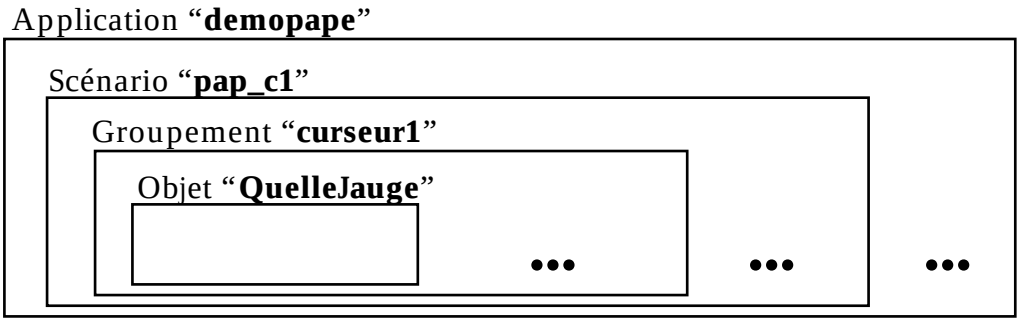


Figure 9 : Localisation de l'Objet "QuelleJauge".

Cette première tâche finie, nous leur avons demandé de créer une nouvelle Application composée de deux Scénarios, composés à leur tour de un ou deux Groupements.

2.4. Analyse selon les schémas d'ÉMA et interprétation des résultats

Nous présentons ici nos réflexions sur les détections de rupture, les annulations et sur les répétitions. Nous ne parlerons pas d'invalidités pour Compo qui met en application le principe du retour d'information pro-actif. DAB et Médiathèque illustreront largement ce cas de figure.

2.4.1. Détections de rupture

Nous présentons figure 10 une description textuelle d'une interaction observée, où figurent également les réactions pertinentes du système. Cette description est directement issue du fichier de capture. La tâche est réalisée ici par un sujet novice et a abouti à l'ouverture de la vue graphique du Groupement de nom "curseur1", alors que le but recherché était l'ouverture de la vue graphique de l'Objet "QuelleJauge". La notation utilisée pour décrire chacune des procédures élémentaires du sujet ainsi que les réactions du système est la suivante :

n:: • (nº de fenêtre), événement : réaction du système (entre parenthèses, les numéros des nouvelles fenêtres créées),

+ signifie l'apparition de la fenêtre dont le numéro est précisé entre parenthèses,

- signifie la disparition de la fenêtre indiquée. Le fichier de capture de cette séquence est donné en annexe 2.

1:: • lancement du logiciel Compo : fenêtre choix de langue +(1)
2:: • (1), sélection du bouton "langue française" : PARIS apparaît dans le cadre
3:: • (1), sélection du bouton "OK" : fenêtre -(1), Palette principale +(2)
4:: • (2), sélection du bouton "Application" : Palette des applications +(3)
5:: • (3), sélection du bouton "demopape" : bouton "demopape" sélectionné
6:: • (3), menu déroulant"Fichier / Ouvrir sélection":APPLICATION::demopape+(4)
7:: • (4), sélection du bouton "pap-c1" : bouton "pap-c1" sélectionné
8:: • (4), menu déroulant "Fichier / Ouvrir sélection" : SCENARIO::pap-c1 +(5)
9:: • (5), sélection du bouton "curseur1" : bouton "curseur1" sélectionné
10:: • (5), menu déroulant "Fichier / Ouvrir sélection":GROUPEMENT::curseur1+(6)
11:: • (6), sélection du bouton "QuelleJauge" : bouton "QuelleJauge" sélectionné
12:: • (6), menu déroulant"Options / Vue Graphique":Vue graphique de curseur1+(7)
13:: • (6), menu déroulant"Fichier / Ouvrir sélection":OBJET::QuelleJauge+(8)
14:: • (8), menu déroulant"Options / Vue Graphique":Vue graphique de QuelleJauge+(9)

Figure 10 : Un exemple de capture réalisée pour Compo.

On remarque que l'ouverture de la vue graphique du Groupement "curseur1" à l'étape 12 est inutile par rapport à l'objectif (qui est l'accès à la vue graphique de l'Objet "QuelleJauge"). En se reportant au graphe des séquences, la procédure élémentaire superflue va de pair avec une détection de rupture et se manifeste comme un cul-de-sac. L'utilisateur ne donne pas suite à l'ouverture de la vue graphique du Groupement "curseur1" (cf. figure 11).

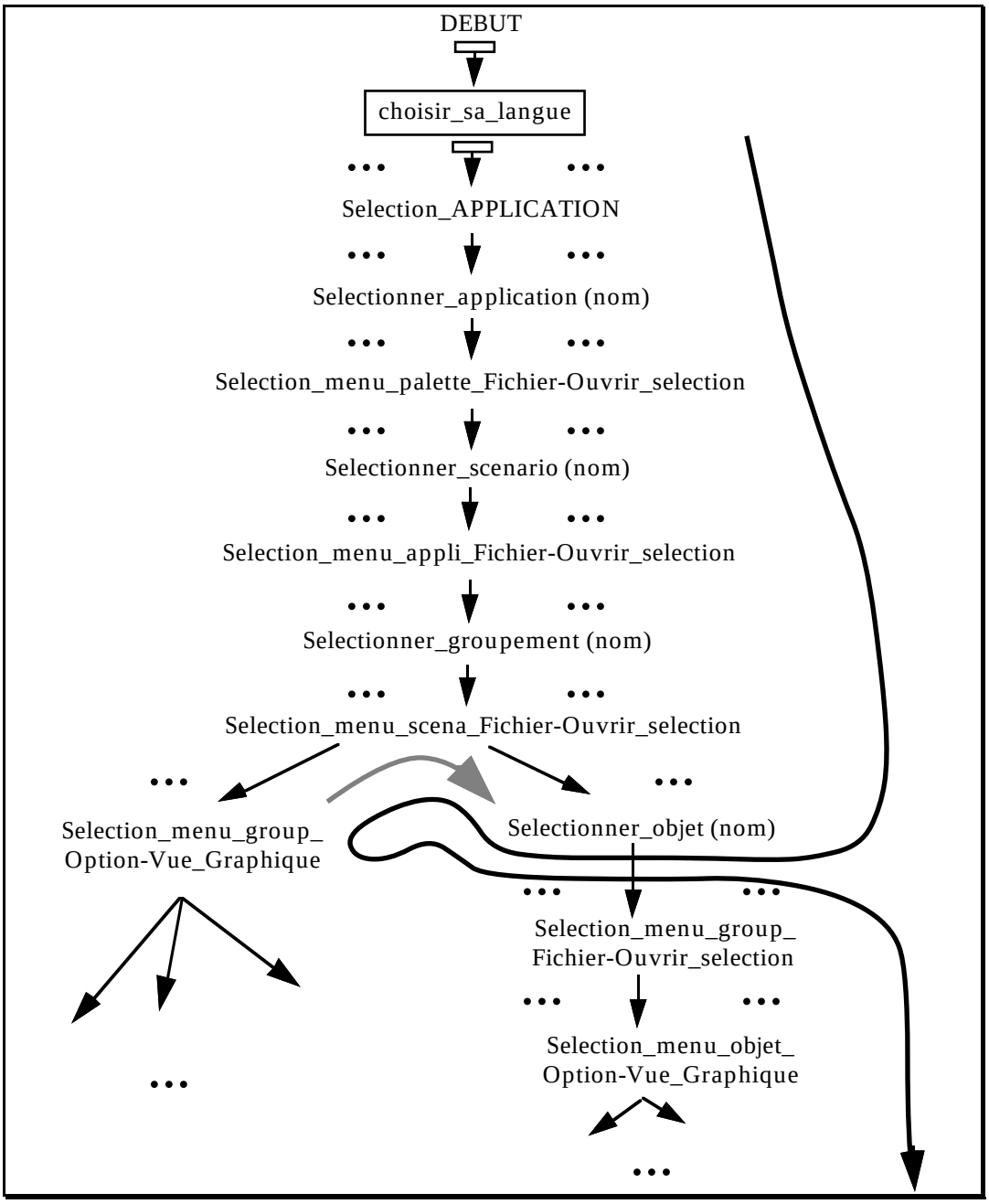


Figure 11 : Compo : observation d'une rupture de séquence. La flèche noire représente le parcours de l'utilisateur. La flèche grise dénote le changement de direction observé. (Le sous-graphe “choisir_sa_langue” a été présenté au cahpitre IV.)

La présence répétée d’un même cul-de-sac indique que l’interface conduit l’utilisateur à une situation d’impasse. Il y a incompatibilité entre le modèle mental et le modèle de tâche du système. Dans le cas de Compo, nous avons constaté que l’incohérence de l’interface était la cause profonde de cette discordance observée chez deux des utilisateurs novices ainsi que chez l'utilisateur occasionnel. En effet, certaines options des menus d'une vue structurelle concernent le sous-concept sélectionné, par exemple l'accès à la vue structurelle de ce dernier, alors que d'autres options concernent directement le

concept lui-même, par exemple, pour un Groupement ou un Objet, l'accès à la vue graphique.

2.4.2. Annulations immédiates

De même que certains utilisateurs après avoir sélectionné un concept, ouvrent la vue graphique du concept père puis laissent cette vue graphique ouverte (cf. problème de rupture présenté ci-dessus où deux des utilisateurs novices ainsi que l'utilisateur occasionnel se trouvent piégés), le troisième utilisateur novice a refermé immédiatement cette vue graphique. L'annulation est alors détectée par la fermeture d'une vue immédiatement après son ouverture. L'observation de cette annulation vient confirmer le problème de cohérence détecté par l'observation des ruptures lors de l'ouverture de la vue graphique d'un Groupement ou d'un Objet .

2.4.3. Répétitions

De nombreuses répétitions s'observent dans l'utilisation de Compo. Nous en avons retenu deux : les répétitions de déplacement de fenêtres et de sélection de concepts (Applications, Scénarios, Groupements ou Objets) précédant l'accès à la vue structurelle du concept sélectionné. La première est mise en relief par l'observation de procédures élémentaires qui n'entraînent pas de modification dans le noyau fonctionnel de Compo, alors que la seconde s'observe à partir de procédures élémentaires en relation directe avec le noyau fonctionnel de Compo.

Déplacements de fenêtres

Les nombreux déplacements de fenêtres observés chez tous nos utilisateurs, à un moment ou à un autre de l'interaction, soulèvent le problème de l'encombrement de l'écran qui contraint l'utilisateur à déplacer, voire à fermer, ses fenêtres très fréquemment.

Observation de clics consécutifs

De nombreux doubles-clics sont observés sur la sélection de concept dans les vues structurelles. Cette observation est systématique pour chacun des utilisateurs novices. Nous ne parlons pas ici du double-clic habituel, mais bien de deux clics consécutifs appliqués à un même objet de l'interface. Si l'on rentre un peu plus dans le détail de l'interface, nous remarquons que le double-clic est utilisable dans la palette des applications sur une icône d'Application pour obtenir la liste des Scénarios qui composent cette Application. De même, si l'on

double-clique sur un des éléments de cette liste, on obtient la liste des groupements composant le scénario sélectionné, etc. (cf. figure 12).

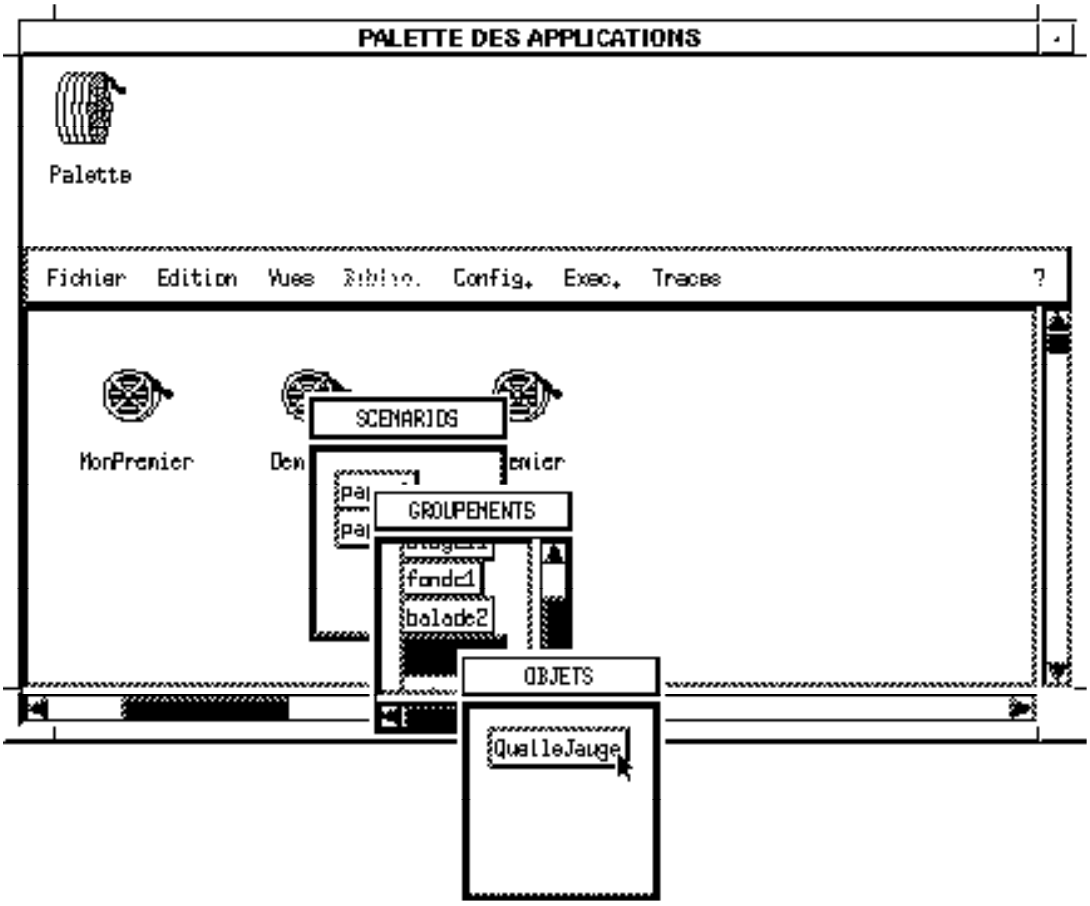


Figure 12 : Compo : utilisation du double-clic dans la palette des Applications.

Or, dans les vues structurelles, le double-clic ne permet pas d'accéder aux sous-concepts du concept sur lequel l'utilisateur a double-cliqué. Nous soulevons ici un problème de cohérence de l'interface, problème qui se retrouve au sein du graphe des séquences, de manière très localisée, en regard des procédures élémentaires de sélections de concepts.

2.5. Conclusion

Pour Compo, les observations effectuées l'ont été de manière “très artisanale”. C'était là nos premiers pas dans le domaine de l'évaluation. Nous n'avons pas suivit de méthode rigoureuse et les résultats portés ci-dessus sont le fruit de notre première expérimentation dans le domaine.

Contrairement à Compo, nous sommes en mesure pour les expérimentations qui suivent (les DAB et Médiathèque), de présenter précisément par rapport à chacune de nos règles, les résultats obtenus.

3. Distributeurs Automatiques de Billets : le DAB initial et le DAB modifié

La seconde expérimentation que nous avons réalisée concerne deux simulations d'interface de distributeur automatiques de billets de banque (DAB). Nous présentons, dans un premier temps, leur fonctionnement. Puis, individuellement pour chacun des DAB, nous exposerons le graphe des séquences qui lui est associé. Ensuite, de manière commune aux deux DAB, nous préciserons la mise en œuvre de la capture. Par ailleurs, ces deux simulations ont fait l'objet d'une évaluation heuristique par un ergonome : Ian Denley, de l'Unité d'Ergonomie de l'Université College, à Londres (UCL). Au vu de ces évaluations, nous mettrons en évidence l'apport de l'analyse automatique fournie par ÉMA.

La première simulation de DAB a été réalisée à l'Unité d'Ergonomie de l'UCL dans un cadre pédagogique. Il s'agissait de faire réaliser, par des étudiants, un prototype d'interface de distributeur automatique de billets de banque. Pour notre expérience, nous avons également testé une deuxième configuration de DAB qui autorise une plus grande souplesse d'utilisation. Ces deux configurations diffèrent par l'existence ou l'absence des touches clavier "Valid" et "Cancel" et par les possibilités, offertes ou non, de retour aux différentes opérations.

3.1. Présentation générale

Pour les besoins de la simulation, une palette matérialise les cartes de crédit de cinq utilisateurs (Eileen, Mina, Clive, Adam et Stephen). Le sujet doit sélectionner sa carte de crédit pour en simuler l'introduction ou le retrait (cf. figure 13, palette de gauche). Lorsqu'une carte de crédit est insérée dans la machine, l'icône qui la représente devient grisée. Le retrait des billets s'effectue en cliquant dans une zone de retrait (cf. figure 13, zone en bas à droite).

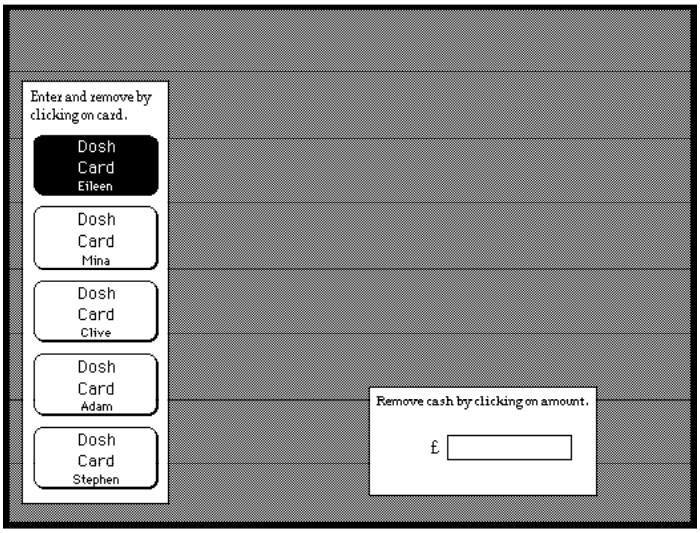


Figure 13 : Eléments de la simulation, pour le DAB.

3.1.1. DAB initial, sans retour sur les choix

La première simulation réalisée est un DAB dont le fonctionnement suit une structure très rigide qui n'autorise aucun retour sur les choix effectués. Cette interface comprend un écran, un clavier numérique et deux boutons situés à droite de l'écran (cf. figure 14).

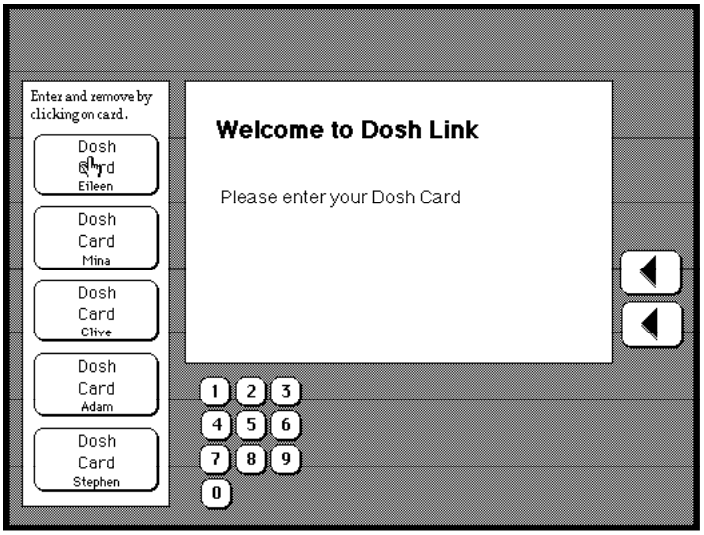


Figure 14 : Ecran d'accueil de le DAB initial.

Une fois la carte introduite, l'écran affiche une demande de saisie du code secret (cf. figure 15).

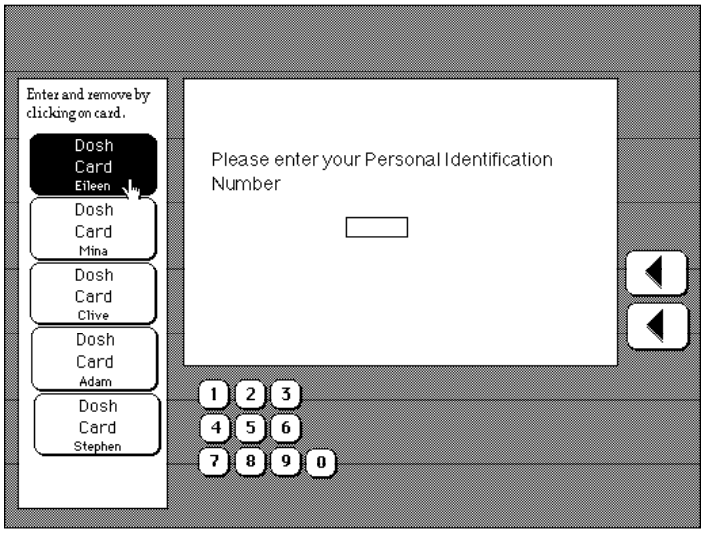


Figure 15 : DAB initial : écran de demande du code secret.

Si l'utilisateur commet une erreur de saisie de son code, il doit retirer sa carte et recommencer l'opération. Une fois les quatre chiffres du code secret correctement entrés, l'écran de la figure 16 apparaît. Cet écran affiche les fonds disponibles sur le compte de l'utilisateur. Ce dernier a alors le choix d'effectuer ou non un retrait. S'il répond “No”, l'opération est terminée et l'utilisateur doit retirer sa carte.

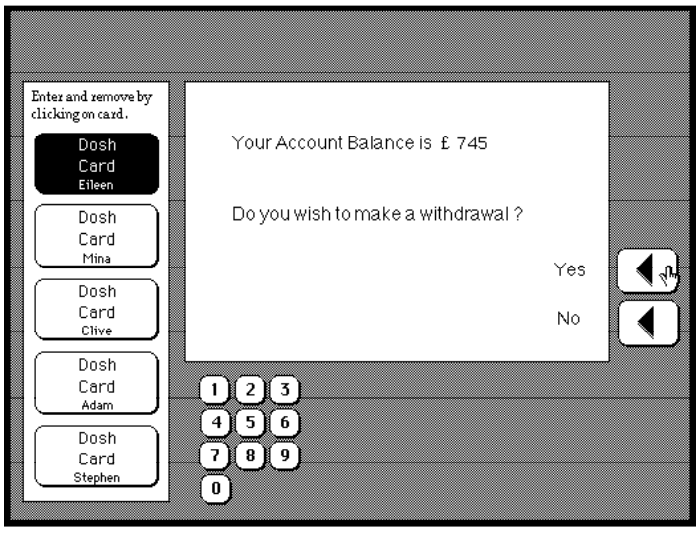


Figure 16 : DAB initial : écran de demande d'opération.

Dans le cas d'une réponse affirmative, on demande au sujet la somme désirée (cf. figure 17). Si son compte est suffisamment approvisionné et si la banque dispose des fonds nécessaires, il retire cet argent et l'opération prend fin ; sinon l'opération prend fin immédiatement après l’affichage du message : “You have insufficient funds for this transaction” ou “Sorry, we are temporarily enable to dispense cash”.

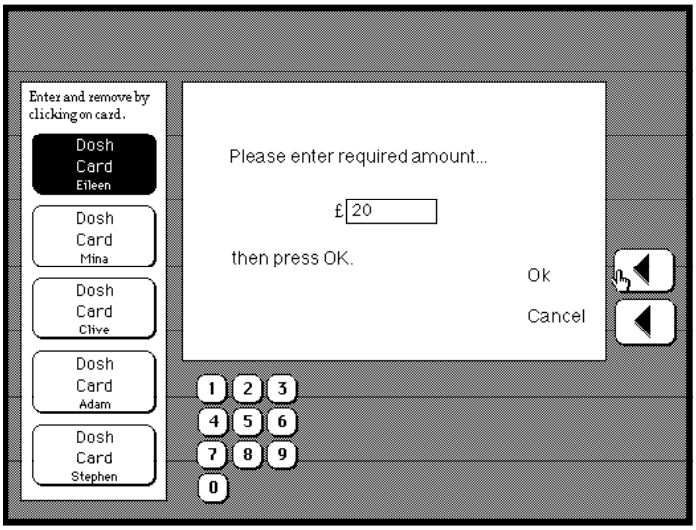


Figure 17 : DAB initial : écran de demande de la somme à retirer.

L'architecture du dialogue est ici très simple, comme le souligne le graphe des séquences que nous détaillons au paragraphe 3.2.2.

3.1.2. DAB modifié avec choix multiples

La seconde interface se veut d'un maniement plus souple que la première en offrant à l'utilisateur la possibilité de revenir sur ses choix. Elle comprend un écran, un clavier numérique enrichi de deux boutons “Valid” et “Cancel”, et de trois boutons situés à droite de l'écran (cf. figure 18).

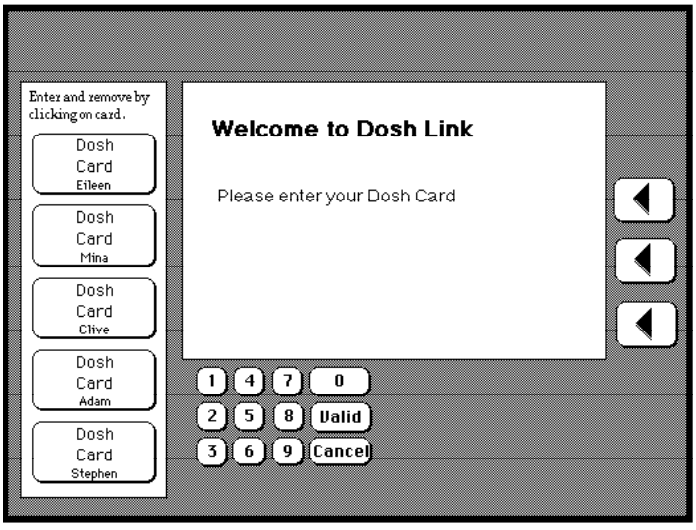


Figure 18 : Ecran d'accueil de le DAB modifié.

Dès que la carte de crédit est introduite, l'écran de demande du code secret apparaît (cf. figure 19). Pendant la saisie du son code, l'utilisateur peut à tout moment décider l'arrêt de cette opération en appuyant sur la touche “Cancel” du clavier numérique. Il lui sera alors demandé de retirer sa carte.

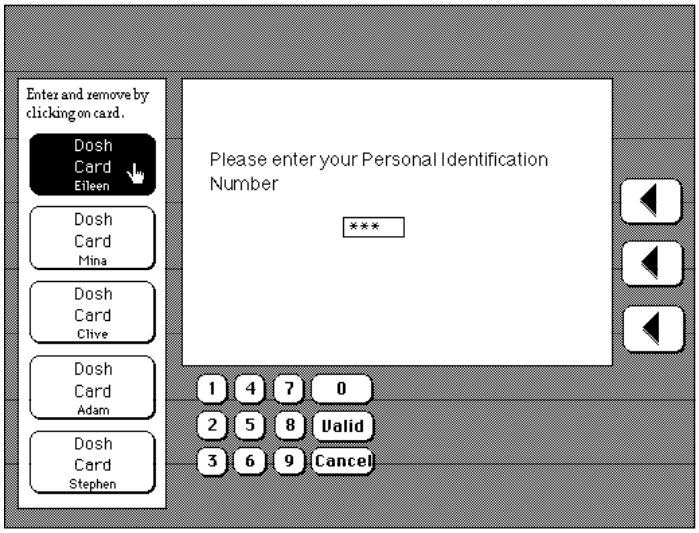


Figure 19 : DAB modifié : écran de demande du code secret.

Lorsque les quatre chiffres du code secret ont correctement été entrés, l'écran de la figure 20 apparaît. Trois opérations s'offrent alors à l'utilisateur : le retrait d'argent, la consultation du solde de son compte ou la fin des opérations.

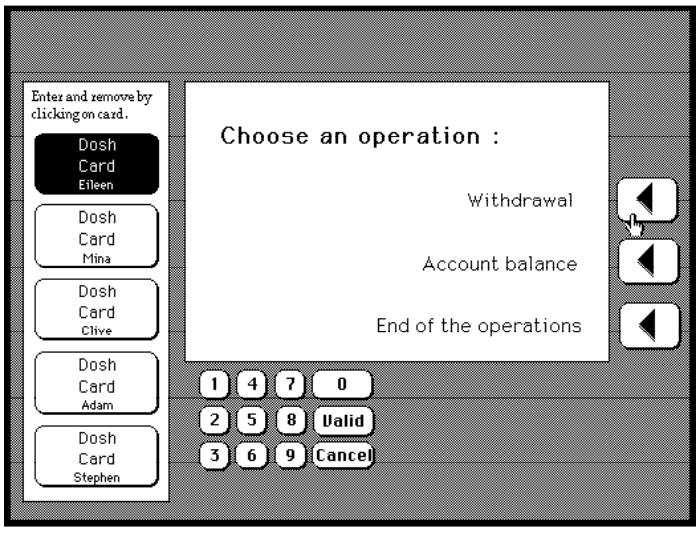


Figure 20 : DAB modifié : écran de demande d'opération.

Lorsque l'utilisateur choisit de visualiser son solde, l'écran de la figure 21 apparaît. L'appui sur la touche “Cancel” a alors pour effet de retourner à l'écran de demande d'opération (figure 20). A tout moment de l'interaction, la touche “Cancel” entraînera un retour à l'écran de demande d'opération.

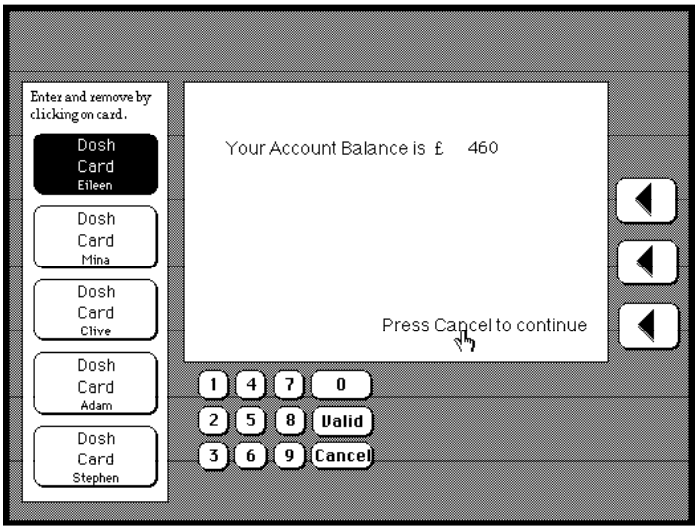


Figure 21 : DAB modifié : visualisation du solde.

Si l'utilisateur désire effectuer un retrait, l'écran de la figure 22 est affiché. Il lui est alors demandé la somme qu'il désire retirer. L'utilisateur peut alors rentrer la somme désirée et cliquer sur “Valid”. Si son compte est suffisamment approvisionné et si la banque dispose des liquidités nécessaires, il peut alors simuler le retrait de la somme demandée, en cliquant dans la zone de retrait (cf. figure 13, zone en bas à droite), sinon un message indiquant la raison de l'impossibilité du retrait apparaît (“You have insufficient funds for this transaction” ou “Sorry, we are temporarily enable to dispense cash”). Une fois l'argent retiré, l'écran de demande d'opération apparaît de nouveau (cf. figure 20).

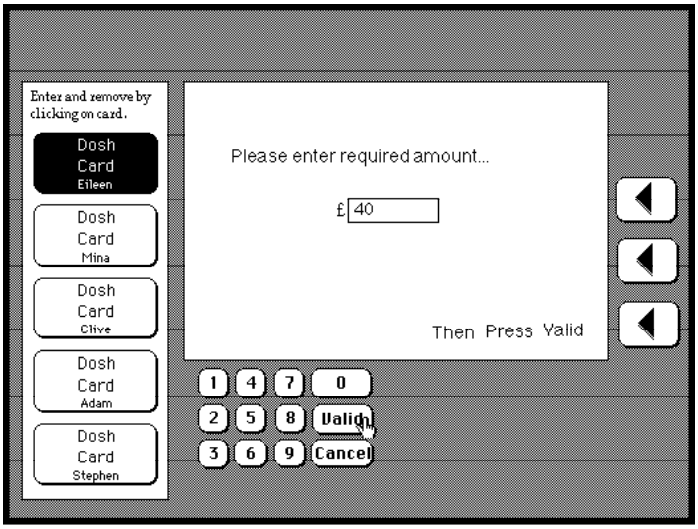


Figure 22 : DAB modifié : écran de demande de la somme à retirer.

Pour retirer sa carte, l'utilisateur doit choisir l'option “End of the operations” sur l'écran de demande des opérations (cf. figure 20).

L'architecture du dialogue pour le DAB modifié est plus complexe que celle de le DAB initial. Elle autorise de multiples opérations.

3.2. Les graphes des séquences

Pour chacun des deux DAB, il a été aisé de construire le graphe des séquences en raison de la relative simplicité de la tâche.

3.2.1. Les procédures élémentaires considérées

Pour chacune de ces deux interfaces, seule la souris permet d'interagir avec le logiciel. Ces interactions sont le résultat de clic souris dans des boutons. Ces boutons sont : les cartes de crédits, la simulation des boutons physiques sur la droite de l'écran, chacune des touches du pavé numérique (touches “0” à “9”, “Valid” et “Cancel”), la zone de retrait de l'argent et la zone de retrait des billets. Aussi ce sont les clics dans chacune de ces zones que nous considérons en tant que procédure élémentaire.

L'ensemble de ces procédures est très réduit. La figure 23 donne l'ensemble de ces procédures pour le DAB initial, sans retour sur les choix effectués, et la figure 24, pour le DAB modifié, avec choix multiples.

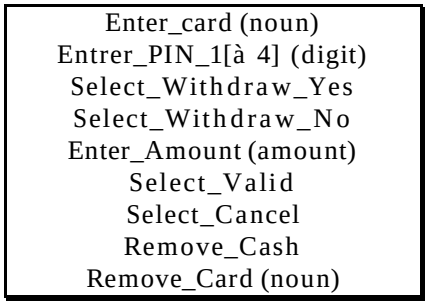


Figure 23 : DAB initial : ensemble des procédures élémentaires perçues.

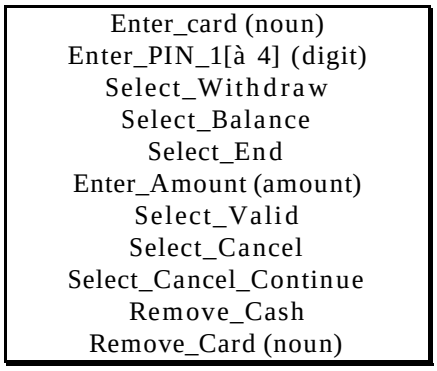


Figure 24 : DAB modifié : ensemble des procédures élémentaires perçues.

3.2.2. DAB Initial : Le graphe des séquences

Le graphe des séquences que nous présentons figure 25 met en évidence l'absence de retour sur les procédures élémentaires effectuées.

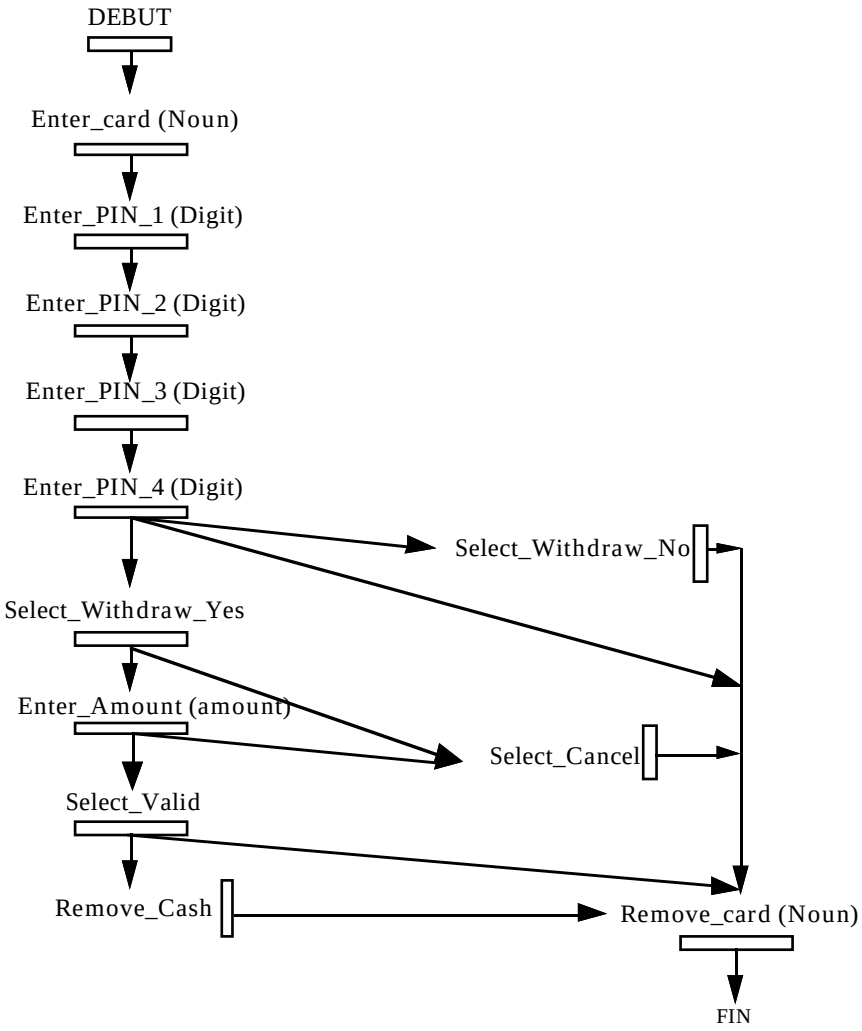


Figure 25 : DAB initial : graphe des séquences complet.

3.2.3. DAB modifié : Le graphe des séquences

Comme pour le DAB initial, le graphe des séquences de le DAB modifiée est très simple. Aussi nous le représentons dans son intégralité ci-dessous, figure 26.

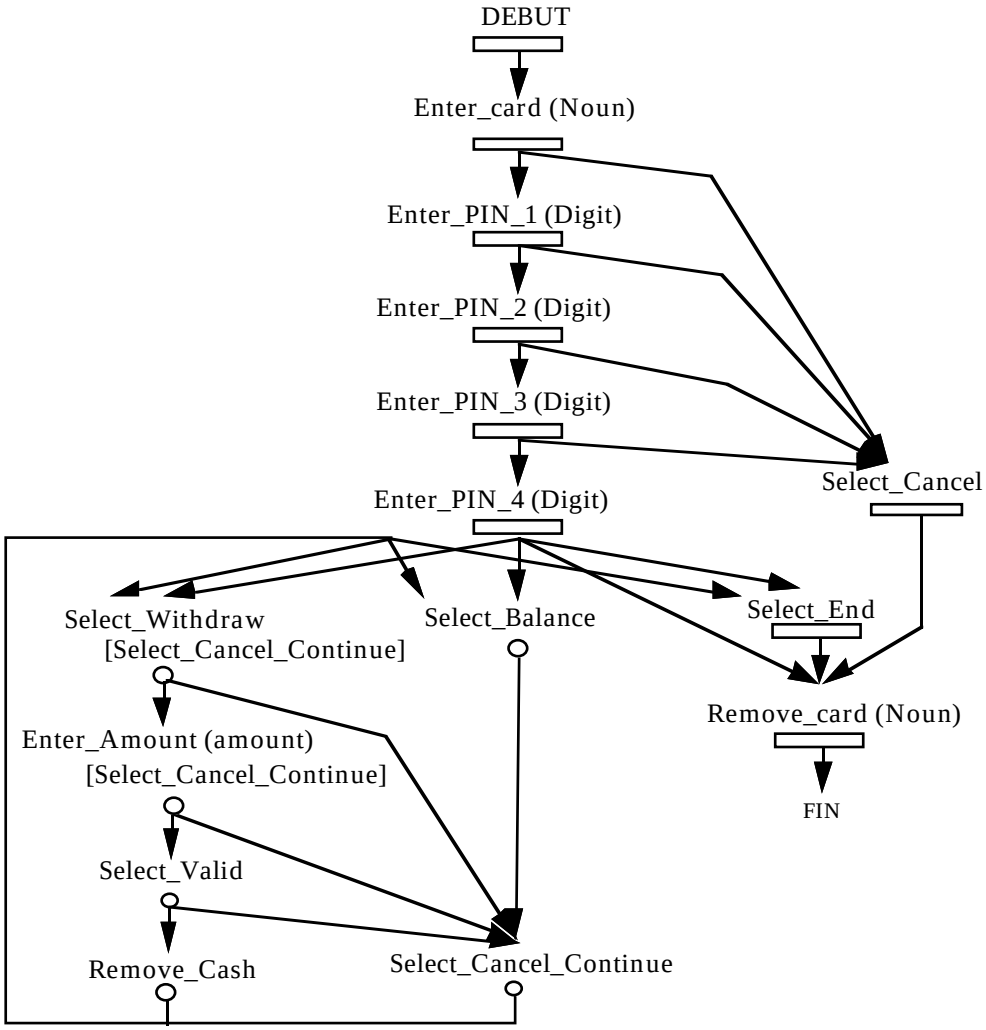


Figure 26 : DAB modifié : graphe des séquences complet.

3.3. Mise en œuvre de la capture

Ces simulations ont été réalisées au moyen d'Hypercard, un logiciel souvent utilisé comme outil de maquettage d'IHM. Hypercard utilise cinq types d'objets : les boutons (par exemple : les boutons "Valid", "Cancel", etc.), les champs (par exemple : la zone de demande de la somme à retirer), les cartes (par exemple : chacune des figures 14 à 22 représente une carte), les fonds (par exemple : le fond est constitué de la palette qui matérialise les cartes de crédits des cinq utilisateurs ainsi que le clavier numérique) et les piles (par exemple : chacune des deux simulations DAB). Une carte est une unité élémentaire d'information. Les cartes sont regroupées en pile, une pile est mémorisée dans un fichier du système

Macintosh. Chaque carte est superposée à son fond, et un fond est généralement partagé par plusieurs cartes. Les boutons et les champs appartiennent soit à un fond, soit à une carte. A chacun des objets d'Hypercard on peut associer un script. Un script est un ensemble de procédures, où chaque procédure est commandée par un événement déclenchant. Par conséquent, si à un bouton est associé le script donné figure 27, l'événement mouseUp (c'est-à-dire le relâchement du bouton de la souris) sur ce bouton fera afficher la carte suivante de la pile.

```
on mouseUp
    go to next card
end mouseUp
```

Figure 27 : Exemple de script sous Hypercard

3.3.1. Technique utilisée

Comme les procédures élémentaires détectées se limitent à des clics souris dans des boutons ou des champs, la capture logicielle s'est faite par simple inclusion de code dans les scripts de chacun des boutons de l'interface. Le lecteur trouvera en annexe 3 un extrait des scripts pour le DAB initial. Nous trouvons, figure 28, le script associé à l'événement mouseUp pour le champ de simulation du retrait des billets.

```
on mouseUp
  -- Sandrine, 9/8/92
        Record ("Remove_Cash")
  -- end Sandrine

  -- goes to card "Return Card"

  go to card "Return Card"
end mouseUp
```

Figure 28 : Script du champ associé au retrait de l'argent, pour le DAB initial

Record, présentée dans la figure 29, est une procédure au sens d'HyperTalk qui retranscrit dans le fichier de capture la procédure élémentaire perçue.

```
on Record Chaine
  global ObsFile

  write the long time & ""@"" & Chaine & return to file ObsFile
end Record
```

Figure 29 : Détail de la fonction Record

Ainsi les procédures élémentaires observées sont les clics souris (repérés grâce à l'événement mouseUp).

Le clavier physique étant présent, malgré son inutilité, nous avons rajouté la capture de son utilisation.

3.3.2. Le recueil des données

Le recueil des données pour cette évaluation s'est effectué dans les conditions suivantes : quatre sujets ont eu à utiliser successivement les deux interfaces pour simuler un retrait d'argent. A chaque sujet, on a demandé de retirer £30, ou le maximum possible, et de prendre connaissance du solde de son compte. L'expérimentateur a pour sa part, demandé aux sujets de commenter, en direct, le déroulement de leur utilisation (technique du "Thinking aloud"). Pendant les sessions, il a observé les utilisateurs, pris des notes et construit son évaluation en direct.

3.4. Analyses heuristiques et apports de l'analyse fournie par ÉMA

Dans un premier temps nous présentons les bases de l'évaluation de l'ergonome. Nous nous servons de ses conclusions pour mettre en relief les apports de l'analyse construite par ÉMA. Ainsi, pour chacun des deux DAB, nous présentons les évaluations de l'ergonome suivies des résultats fournis par ÉMA.

3.4.1. Bases de l'évaluation heuristique de l'expert

L'évaluation menée par l'ergonome se situe, au sein de notre taxonomie, dans la catégorie des techniques d'évaluation non automatique, travaillant sur la base d'un savoir heuristique ; et, dans ce cas précis, l'interface est implémentée et le sujet présent.

Les connaissances utilisées par l'ergonome pour cette évaluation reposent essentiellement sur sa propre expérience, mais aussi sur un ensemble de principes connus qu'il tente de vérifier. Ces principes sont du même ordre que ceux présentés dans [Scapin 90] (cf. chapitre III, paragraphe 3.1.1.3). Le tableau de la figure 30 décrit brièvement ces 12 principes, et pour chacun d'entre eux, nous précisons le critère de Scapin le plus proche.

Principe	Description	Critères de [Scapin 90]
Compatibilité	L'utilisateur doit être en mesure de réutiliser un savoir précédemment acquis en dehors du système.	Compatibilité
Cohérence	L'utilisateur doit être en mesure de généraliser l'expérience acquise au sein d'une des parties du système, aux autres parties.	Homogénéité
Simplicité	Le nombre et la complexité des actions nécessaires doivent être réduits.	Charge de travail
Redondance	L'utilisateur doit pouvoir accéder de différentes manières au résumé des informations.	Guidage
Pertinence	Les informations critiques doivent être présentées à l'utilisateur de manière pertinente.	Signifiance des codes
Transparence	Les informations sur les tâches exécutables, aussi bien que sur l'état de la machine doivent être formulées de manière non ambiguës.	Guidage
Complétude	Les informations sur les tâches exécutables, aussi bien que sur l'état de la machine doivent être complètes, c'est-à-dire que toutes les options doivent être présentes simultanément.	Guidage
Support à l'orientation	Si l'information à présenter ne peut pas l'être dans sa totalité en une fois, l'utilisateur doit être aidé dans sa recherche.	Guidage
Retour d'information	L'utilisateur doit être informé des conséquences de ses actions.	Contrôle explicite
Réversibilité	L'utilisateur doit avoir la possibilité de retourner à des états antérieurs du système.	Gestion des erreurs
Contrôle	L'utilisateur doit avoir la possibilité de contrôler les actions de la machine.	Contrôle explicite
Souplesse	L 'utilisateur doit pouvoir choisir les modalités qui lui conviennent en fonction des modalités présentes dans le système et de son expérience.	Adaptabilité

Figure 30 : Principes ergonomiques utilisés par Ian Denley.

Comme nous le décrivons ci-dessous, les résultats fournis par ÉMA confirment, pour la plupart, les analyses de l'ergonome. La retranscription complète en anglais du texte des deux évaluations produites par l'ergonome est fournie en annexe3.

3.4.2. DAB initial sans retour sur les choix

3.4.2.1. Résultats de l'évaluation de l'expert

Dans son analyse, l'ergonome a relevé deux types d'anomalie : les problèmes liés à la mise en œuvre de la simulation et ceux relevant de l'interface.

Les problèmes liés à la mise en œuvre de la simulation

Les problèmes liés à la mise en œuvre de la simulation concernent les arrangements nécessaires pour implémenter cette simulation, et non directement l'interface utilisateur de le DAB. Les bases de la simulation étant identiques pour les deux DAB, nous avons regroupé ici toutes les remarques liées à la situation de simulation :

1 Un utilisateur a tenté de placer le curseur de la souris dans le rectangle de l'écran de demande du code secret (cf. figure 19) avant de rentrer son code. Il n'y a alors eu aucune réaction de l'interface : ni message d'erreur, ni signal sonore.

2 Pour saisir leur code secret, trois utilisateurs sur quatre ont utilisé le clavier numérique physique avant celui du simulateur. Cette confusion n'apparaît qu'à l'entrée du code secret. Pour les autres demandes, le sujet n'utilise que le clavier simulé. L'apprentissage est ici très rapide et cette confusion n'est jamais intervenue une seconde fois pendant l'interaction.

3 Un utilisateur a double-cliqué pour retirer sa carte, ce qui a entraîné une confusion car la carte a aussitôt été réinsérée, alors qu'il désirait simplement la retirer. Pour pallier ce problème, il est nécessaire, lors du retrait de la carte, de supprimer la mémorisation des clics souris lorsque ces derniers sont proches dans le temps.

Pour chacune de ces trois observations, il est clair que l'utilisation de la souris et la disponibilité du clavier physique ont introduit un biais. Pour se rapprocher de la réalité, l'utilisation d'un écran tactile aurait permis d'éviter ces trois problèmes de retour d'information.

Les problèmes liés à l'interface utilisateur

Les problèmes directement en liaison avec l'interface de le DAB initial sont les suivants:

b1 L'utilisateur n'a pas la possibilité d'annuler le code secret déjà tapé. Par exemple, si l'utilisateur tape un chiffre erronné dès le premier chiffre de son code secret, il doit entrer les trois autres chiffres. De plus, il ne lui sera pas donné une seconde chance. Il devra retirer sa carte et recommencer l'opération. La gestion de l'erreur, à ce niveau, est inexistante.

b2 Si le code secret est incomplet et que l'utilisateur effectue d'autres commandes (un utilisateur ne s'était pas rendu compte qu'il manquait un chiffre et a tenté de valider son code secret en cliquant dans les boutons à droite de l'écran de le DAB), aucun retour d'information n'est donné pour indiquer que la procédure élémentaire engagée est invalide. Cette observation soulève aussi un problème de compatibilité avec l'utilisation classique des DAB, où une fois le code secret entré, il est souvent nécessaire de le valider.

b3 Trois utilisateurs n'ont pas vu le dialogue les informant du montant disponible sur leur compte. Ils ont été dans l'obligation de réinsérer leur carte afin de retrouver cette information. La pertinence du dialogue n'est pas suffisante ; ce dialogue nécessite d'être reconçu.

3.4.2.2. Analyse fournie par ÉMA

La structure de l'interaction est très rigide. En conséquence, il n'y a pas de ruptures possibles, ni d'annulations, ni de répétitions. Restent les invalidités.

Les problèmes liés à la mise en œuvre de la simulation

Nous avons traqué toutes les procédures élémentaires décrites dans la figure 23 ainsi que les utilisations du clavier physique. Ce choix nous a permis de détecter dans le fichier de capture (cf. figure 31) les invalidités liées à l'utilisation du clavier physique, mettant ainsi en avant les problèmes énoncés aux points -1.- et -2.- de l'évaluation de l'expert.

```
Beginning of the session at : 10:42:12
10:45:33@ Bkgnd_keyDown_on ()
                                        (*Invalide*)
10:45:34 @ Bkgnd_keyDown_on ()
                                        (*Invalide*)
10:45:39@ Enter_card (Stephen)
10:45:45 @ Bkgnd_keyDown_on (5)
                                        (*Invalide*)
10:45:46 @Bkgnd_keyDown_on (0)
                                        (*Invalide*)
10:45:47@ Bkgnd_keyDown_on (0)
                                        (*Invalide*)
10:45:54@ Enter_PIN_1 (5)
...
```

Figure 31 : DAB initial : fichier de capture annoté par ÉMA.

La figure 32 montre une représentation textuelle du graphe des séquences annoté par ÉMA. Cette annotation permet de souligner la redondance des invalidités pour une même procédure élémentaire. Dans le cas présenté dans la figure 32, il s'agit principalement du problème observé lors de la première utilisation du clavier simulé pour la saisie du code secret.

```
DEBUT
            (* Invalide : 10:45:33@ Bkgnd_keyDown_on (),
                    suivie de : 10:45:34@ Bkgnd_keyDown_on ()     *)
            (* Invalide : 10:45:34 @ Bkgnd_keyDown_on (,
                    suivie de : 10:45:39@ Enter_card (Stephen)     *)

                    -|->    Enter_Card (Noun) .
Enter_Card
            (* Invalide : 10:45:45@ Bkgnd_keyDown_on (5) ,
                    suivie de : 10:45:46@ Bkgnd_keyDown_on (0)    *)
            (* Invalide : 10:45:46@ Bkgnd_keyDown_on (0) ,
                    suivie de :  10:45:47@ Bkgnd_keyDown_on (0) *)
            (* Invalide : 10:45:47@ Bkgnd_keyDown_on (0) ,
                    suivie de :  10:45:54@ Enter_PIN_1 (5)   *)

                    -|->    Enter_PIN_1 (Digit) .
    Enter_PIN_1     -|->    Enter_PIN_2 (Digit) .
    Enter_PIN_2     -|->    Enter_PIN_3 (Digit) .
    Enter_PIN_3     -|->    Enter_PIN_4 (Digit) .
    Enter_PIN_4     -|->    Select_Withdraw_Yes ;
                            Select_Withdraw_No ;
                            Remove_Card (Noun) .
    ...
```

Figure 32 : DAB initial : représentation textuelle annotée par ÉMA du graphe des séquences.

Le problème soulevé par le point -3-, c'est-à-dire l'utilisation du double-clic pour le retrait de la carte de crédit, n'est pas pointé directement par l'une de nos quatre règles de comportement. Nous introduisons ici l'utilité de fournir à l'ergonome un ensemble de services de type indexation ou statistique. Ainsi, dans ce cas précis, l'observation du nombre de redémarrages de l'interrogation peut devenir intéressante. Le service à fournir est, pour ce cas particulier, la mise en évidence d'une procédure élémentaire particulière. De plus, ce service peut s'appliquer à de nombreuses autres requêtes de l'évaluateur, comme, par exemple, le nombre d'exécution d'une procédure élémentaire donnée.

Les problèmes liés à l'interface utilisateur

Encore une fois, l'observation des invalidités attire l'attention de l'évaluateur sur un point critique : le problème soulevé en -b_1.-, (mauvais retour d'information). Ce problème porte sur la recherche de validation du code secret de l'utilisateur alors que seuls trois des numéros de ce code avaient été rentrés.

Nous présentons figure 33, l'extrait de la représentation graphique annotée par ÉMA, du graphe des séquences, soulignant cette anomalie. Sur cette figure, le menu déroulant associé au bouton *Invalide* est déroulé.

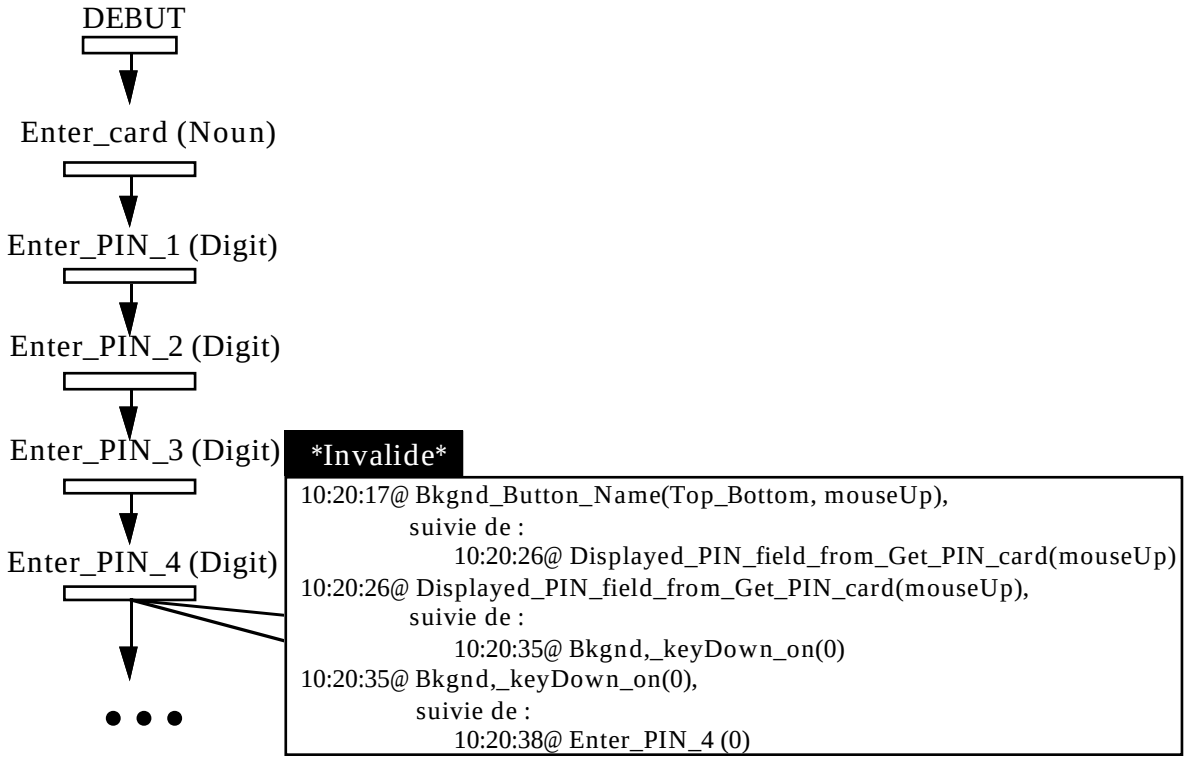


Figure 33 : DAB initial : représentation graphique, annotée par ÉMA, du graphe des séquences.

Le point -c_1.-, qui souligne la non pertinence du dialogue annonçant le solde du compte de l'utilisateur, implique un redémarrage quasi-systématique (trois utilisateurs sur quatre). Il est clair que la simple observation du redémarrage ne suffit pas à comprendre les raisons qui ayant conduit l'utilisateur à réintroduire sa carte de crédit. Par contre, cette observation peut servir d'indicateur et permettre à l'évaluateur d'orienter ses recherches autour du redémarrage, voire d'affiner sur ce point les questions qu'il compte poser à l'utilisateur.

3.4.3. DAB modifié avec choix multiple

3.4.3.1. Résultats de l'évaluation de l'expert

L'expert a détecté cinq anomalies de gravité différente :

- un problème grave,
- des problèmes mineurs.

Un problème grave est un problème qui induit des perturbations chez les utilisateurs. Un problème mineur est issu de réflexions des utilisateurs, des

préférences qu'ils aimeraient voir se réaliser, voire d'observations personnelles de l'ergonome.

Un problème grave

En fait, nous n'avons noté qu'un seul problème grave :

a_2.L'utilisation de la touche “Valid”, qui permet de confirmer le montant du retrait désiré, a posé un problème à chacun des utilisateurs. Cette touche est trop loin du dialogue et les utilisateurs ont du mal à la trouver (cf. écran de la figure 21). Les touches “Valid” et “Cancel”, intégrées au clavier numérique simulé, sont difficiles à repérer. Nous soulevons ici un problème de transparence et de pertinence du dialogue énoncé.

Des problèmes mineurs

Les problèmes mineurs sont au nombre de quatre :

b_2.La phrase “Cancel to continue” est à reformuler. En effet, les notions d'annulation et de continuation sont antagonistes, la transparence du dialogue n'est ici pas satisfaisante.

c_2.Un utilisateur aurait souhaité que le choix de “End of the operations” éjecte automatiquement la carte (exemple de migration de tâche vers le système). La simplicité des actions à effectuer en serait améliorée.

d_2.Aucun retour d'information n'indique comment procéder pour abandonner l'opération en cours. Par exemple, si l'utilisateur commence un retrait, puis désire voir l'état de son compte avant de prendre la décision de la somme à retirer, l’interface ne montre pas l'utilisation possible de la touche “Cancel”.

e_2.Le dialogue oral qui invite l'utilisateur à retirer sa carte se poursuit quelques secondes après le retrait. Cette prolongation du retour d'information est superflue.

Chacune de ces remarques est soit issue de commentaires oraux des utilisateurs (pour les points -b_2.- et -c_2.-), soit d'observations soulevées directement par l’expert (pour les points -d_2.- et -e_2.-).

3.4.3.2. Analyse fournie par ÉMA

Comme pour le DAB initial, nous observons qu'il n'est possible de détecter ni rupture, ni répétition dans cette interface. En effet, une fois encore, le dialogue est très dirigé (lorsque ce ne sont pas des relations de restriction (-I->), cette interface est fortement conduite par des relations d'obligation (-O->)). Par contre, l'observation de l'action Select_Cancel_Continue qui suit les actions “Select_Withdraw” et “Enter_Amount” traduit une annulation comme le précise le graphe des séquences (cf. figure 26, notification des annulations d'action entre “[” et “]”).

Un problème grave

Le problème soulevé par l'ergonome au point -a_2.- qui relate la difficulté rencontrée par les utilisateurs pour localiser la touche “Valid” sur le clavier simulé, s’est traduit par de nombreuses invalidités ainsi que des annulations. La figure 34 où nous présentons le fichier de capture annoté par ÉMA souligne ce constat.

```
10:22:55@ Select_Withdraw
10:23:03@Enter_Amount (30)
10:23:10@ Bkgnd_Button_Name(Bottom_Button, mouseUp)
                                                    (* Invalide *)
10:23:16@ Bkgnd_Button_Name(Bottom_Button, mouseUp)
                                                    (* Invalide *)
10:23:26@Amount_field_from_Get_Amount_card(mouseUp)
                                                    (* Invalide *)
10:23:33@Bkgnd_Button_Name(Bottom_Button, mouseUp)
                                                    (* Invalide *)
10:23:35@ Bkgnd_Button_Name(Top_Bottom, mouseUp)
                                                    (* Invalide *)
10:23:38@ Bkgnd_Button_Name(Bottom_Button, mouseUp)
                                                    (* Invalide *)
10:23:49@ Select_Cancel_Continue
                                                    (*Annulation*)
10:23:59@ Select_Balance
10:24:11@ Select_Cancel_Continue
10:24:16@Select_Withdraw
10:24:22@ Enter_Amount (30)
10:24:32@ Bkgnd_Button_Name(Bottom_Button, mouseUp)
                                                    (* Invalide *)
10:24:34@ Bkgnd_Button_Name(Top_Bottom, mouseUp)
                                                    (* Invalide *)
10:24:40@ Bkgnd_Button_Name(Middle_Bottom, mouseUp)
                                                    (* Invalide *)
10:25:21@ Select_Valid
10:25:32@ Remove_Cash
```

Figure 34 : DAB modifié, fichier de capture annoté par ÉMA.

Grâce au graphe des séquences (cf. figure 35), nous localisons avec précision l'occurrence de ces erreurs de manipulation.

```
Select_Withdraw      [Select_Cancel_Continue]
                              -O->   Enter_Amount (amount) ;
                                     Select_Cancel_Continue.
Select_Balance
                              -O->   Select_Cancel_Continue .
Enter_Amount                  [Select_Cancel_Continue]
    (* Invalide : 10:23:10@ Bkgnd_Button_Name(Bottom_Button, mouseUp),
        suivie de : 10:23:16@ Bkgnd_Button_Name(Bottom_Button, mouseUp) *)
    (* Invalide : 10:23:16@ Bkgnd_Button_Name(Bottom_Button, mouseUp),
        suivie de : 10:23:26@ Amount field from Get Amount card, mouseUp)    *)
    ...
    (*Annulation : 10:23:49@ Select_Cancel_Continue
        suivie de :  10:23:59@ Select_Balance                               *)
    (* Invalide : 10:24:32@ Bkgnd_Button_Name(Bottom_Button, mouseUp)
        suivie de : 10:24:34@ Bkgnd_Button_Name(Top_Bottom, mouseUp)    *)
    (* Invalide : 10:24:34@ Bkgnd_Button_Name(Top_Bottom, mouseUp)
        suivie de : 10:24:40@ Bkgnd_Button_Name(Middle_Bottom, mouseUp)  *)
    (* Invalide : 10:24:40@ Bkgnd_Button_Name(Middle_Bottom, mouseUp)
    suivie de : 10:25:21@ Select_Valid*)
                              -O->   Select_Valid ;
                                     Select_Cancel_Continue .
    ...
```

Figure 35 : DAB modifié, représentation textuelle annotée par ÉMA du graphe des séquences.

Des problèmes mineurs

Contrairement à le DAB initial pour lequel les problèmes soulevés par l'expert découlaient de ses observations, les problèmes mineurs mentionnés ici relèvent uniquement du jugement subjectif des sujets ou de l'expert. Ainsi ces problèmes ne sont pas mis en évidence par ÉMA puisqu'ils n'ont entraînés aucun dysfonctionnement lors de la manipulation de l'interface. Par exemple, la phrase "Cancel to continue", bien qu'elle relève une incompatibilité de termes, est comprise par les utilisateurs dans le contexte d'utilisation de le DAB. Nous soulevons ici le point faible de notre système. En effet nous ne sommes pas en mesure de repérer les anomalies auxquelles l'utilisateur s'adapte, sans que cela le perturbe dans l'utilisation de l'interface.

3.5. Conclusion

En résumé, pour chacun des utilisateurs, l'analyse fournie par ÉMA a soulevé:

DAB initial	Répétitions	Invalidités	Annulations
Utilisateur no 1	1	4	0
Utilisateur no 2	1	4	0
Utilisateur no 3	2	0	0
Utilisateur no 4	2	7	0
TOTAL DAB initial	**6**	**15**	**0**
DAB modifié			
Utilisateur no 1	2	11	2
Utilisateur no 2	1	0	1
Utilisateur no 3	1	4	1
Utilisateur no 4	3	5	3
TOTAL DAB modifié	**7**	**20**	**7**

Ces résultats sont directement extraits des fichiers annotés que nous avons joint à l'annexe 3. Les répétitions découlent toutes d'une utilisation normale du logiciel ; par contre les 35 invalidités observées et 2 des annulations concernent les problèmes présentés au paragraphe précédent.

4. Médiathèque

Nous présentons maintenant notre dernière expérimentation : Médiathèque.

Le logiciel Médiathèque permet l'interrogation de la base de données de la médiathèque de l'IMAG (institut d'Informatique et de Mathématiques Appliquées de Grenoble). Cette bibliothèque regroupe quelques 15500 titres d'ouvrages.

4.1. Présentation générale

L'interrogation de la base de données, via Médiathèque, repose sur la construction de questions à base d'opérateurs logiques (ET, OU et SAUF). La figure 36 présente l'écran d'accueil. L'utilisateur peut orienter sa recherche sur les ouvrages, les rapports, les périodiques ou les films audiovisuels disponibles à la médiathèque.

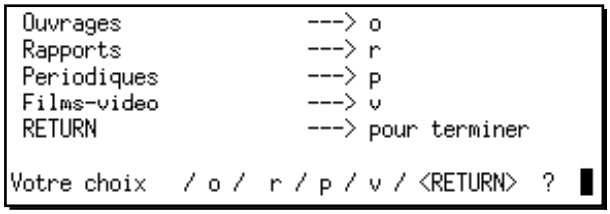


Figure 36 : Ecran d'accueil du logiciel Médiathèque.

Les écrans suivants montrent la construction d'une requête. La figure 37 présente l'écran qui s'affiche dès que l'utilisateur a orienté sa recherche sur les ouvrages disponibles. De manière générale, l'utilisateur construit progressivement sa requête selon les étapes imposées par le système (voir les figures 37 et 38).

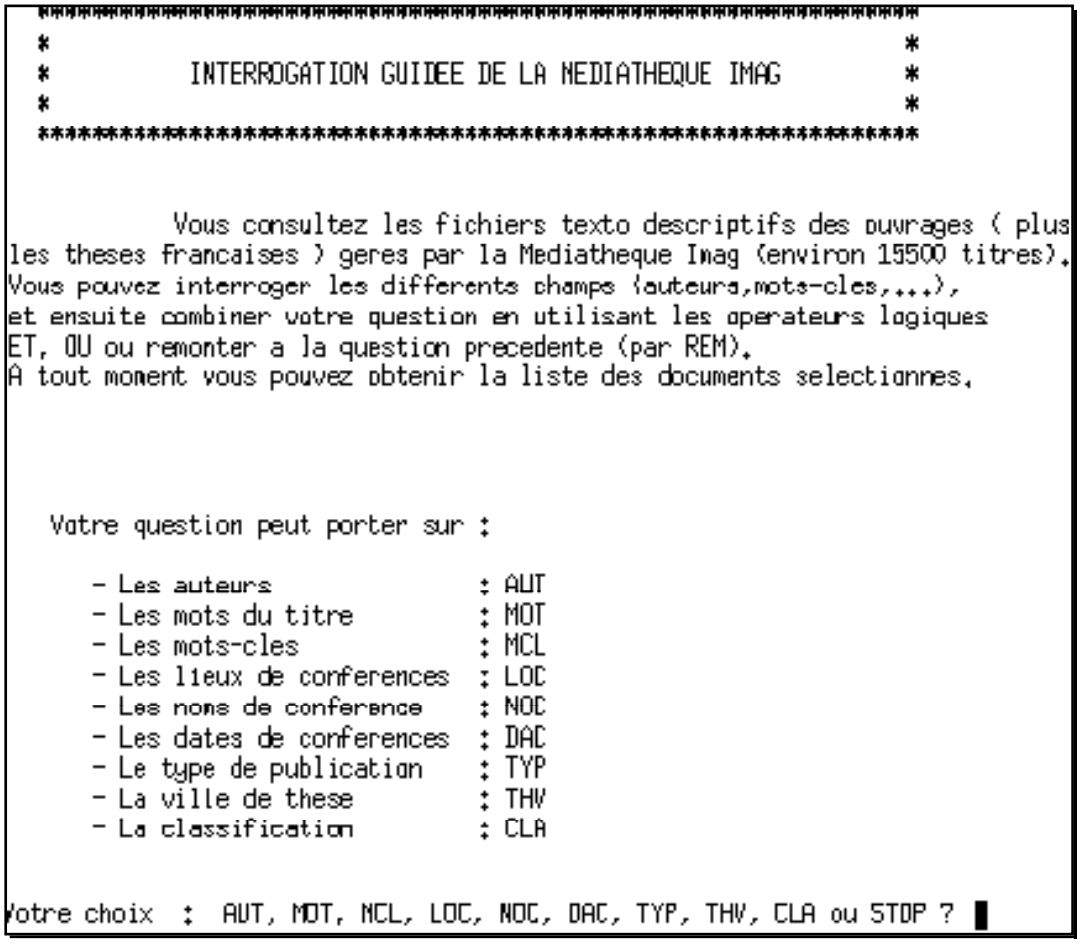


Figure 37 : Construction de requête avec le logiciel Médiathèque.

La requête de la figure 37 permet de construire chacun des termes de la question, et la figure 38 présente l'écran qui permet de spécifier le lien entre les termes ou de demander l'édition des résultats.

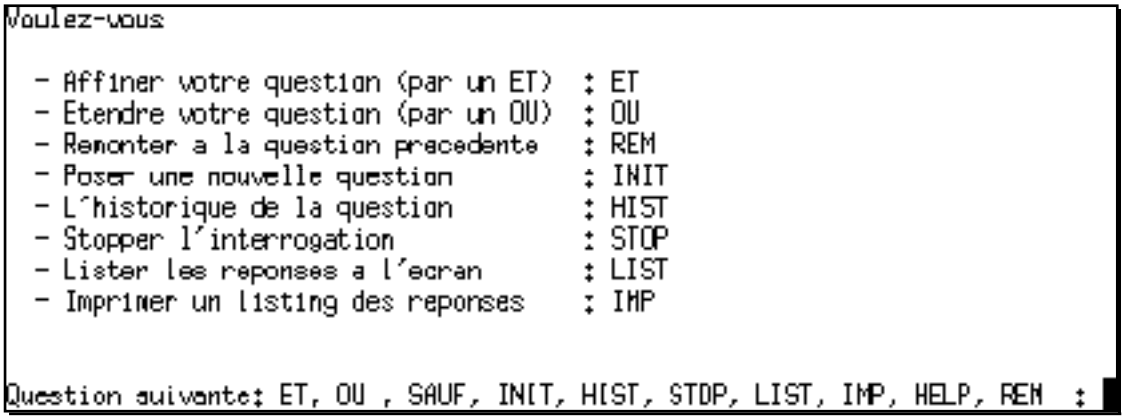


Figure 38: Construction de requête dans le logiciel Médiathèque.

Le dialogue est entièrement dirigé par l'ordinateur sur la base du modèle questions-réponses.

4.2. Le graphe des séquences

Dans cette étude, nous avons restrient l'observation à l'interrogation des ouvrages. Deux raisons sont à l'origine de cette décision : comme le souligne le tableau de la figure 39, la recherche d'ouvrage est la plus fréquente ; pour toute classe d'ouvrage, le modèle d'interrogation est identique.

Type de document	Nbre d'interrogation
Ouvrages	4857
Rapports	2800
Périodiques	1612
Films vidéo	375

Figure 39 : Nombre d'interrogations par type de documents (de Février à Juin 1993).

4.2.1. Les procédures élémentaires

L'interaction avec le logiciel Médiathèque se construit autour des choix de l'utilisateur en réponse aux questions qui lui sont posées. Aussi, nous considérons ici chaque saisie de réponse comme une procédure élémentaire.

Les procédures élémentaires que nous avons considérées dans Médiathèque sont de deux ordres :

- les entrées de choix de menu qui répondent à l'une des trois questionssuivantes :

  "- Votre choix : AUT, MOT, MCL, LOC, NOC, DAC, TYP, THV, CLA ou STOP ?"

  "- Question suivante: ET, OU , SAUF, INIT, HIST, STOP, LIST, IMP, HELP, REM :"

  "- Votre choix : AUT MOT MCL LOC NOC DAC LAN DAT THV TYP CLA HELP STOP :"

Un choix correspond à l'entrée au clavier d'une de ces combinaisons de trois/quatre lettres suivie de la validation (c'est-à-dire du retour chariot). Aussi les procédures élémentaires sont définies dans le graphe des séquences et représentées dans le fichier de capture, par : Choix_AUT, Choix_MOT, Choix_MCL, etc.

- les saisies de paramètres en réponse aux questions du système, par exemple :
  question posée suite au choix de menu MCL :
  "- Donnez au singulier et sans articles le mot-cle ou le debut du mot-cle recherche :"
  question posée suite au choix de menu AUT :
  "- Donnez le nom ou le debut du nom de l'auteur :"
  question posée suite au choix de menu IMP :
  "- Impression minimum ou complete (M ou C) ?"
  questions posées suite au choix de menu DAC :
  "- Donnez sur 4 caracteres la date de la conference (ex 1980) :
  - Donnez sur 2 caracteres le mois de la conference
  (ex : 01 pour janvier... ou retour chariot) :
  - Donnez sur 2 caracteres le jour de la conference (ou retour chariot) :"

De même que pour les entrées de menu, nous considérons qu'une procédure élémentaire est une entrée validée. Nous retrouvons ainsi, dans le graphe des séquences, les procédures élémentaires suivantes : Entrer_mot_clef (mot_clef), Entrer_nom_auteur (nom), Choix_imprime (choix), Entrer_annee_conf (annee), Entrer_mois_conf (mois), Entrer_jour_conf (jour_facultatif), etc. Ces procédures sont enregistrées dans le fichier de capture sous la forme : Entrer_mot_clef (interface), Entrer_nom_auteur (cout), Choix_imprime (C), Entrer_annee_conf (1980), Entrer_mois_conf (01), Entrer_jour_conf (), etc.

4.2.2. Le graphe des séquences

Le graphe des séquences présenté par la figure 40 est compacté. En effet, afin d'en simplifier la lecture nous y avons regroupé les procédures élémentaires de même nature.

- ***Choix_Votre_choix*** regroupe les neuf choix possibles en réponse à la question(cf. figure 37) :
  "Votre choix : AUT, MOT, MCL, LOC, NOC, DAC, TYP, THV, CLA ou STOP ?"
  Une fois le choix défini, l'utilisateur devra, excepté pour le choix HELP, entrer les paramètres complétant son choix. Ceci est représenté par Entrer_parametre(s)_du_choix, qui peut-être la procédure élémentaire

Entrer_nom_auteur(nom) ou Entrer_mot_cle(mot). Le choix STOP entraîne l'arrêt de l'interrogation des ouvrages ; il n'est pas inclus dans Choix_Votre_choix.

- ***Choix_Question_suivante*** regroupe les neuf choix en réponse à la question suivante (cf. figure 38) :

  "Question suivante: ET, OU , SAUF, INIT, HIST, STOP, LIST, IMP, HELP, REM :"

  Ici aussi le choix STOP n'est pas inclus dans Choix_Question_suivante.

- Enfin, ***Choix_Sup*** regroupe les trois réponses que sont Choix_LAN, Choix_DAT et Choix_HELP. Ces choix viennent s'ajouter aux options de Choix_Votre_choix à partir de l'entrée du deuxième terme d'une question. L'utilisateur répond ici à la question :

  "Votre choix : AUT MOT MCL LOC NOC DAC LAN DAT THV TYP CLA HELP STOP :"

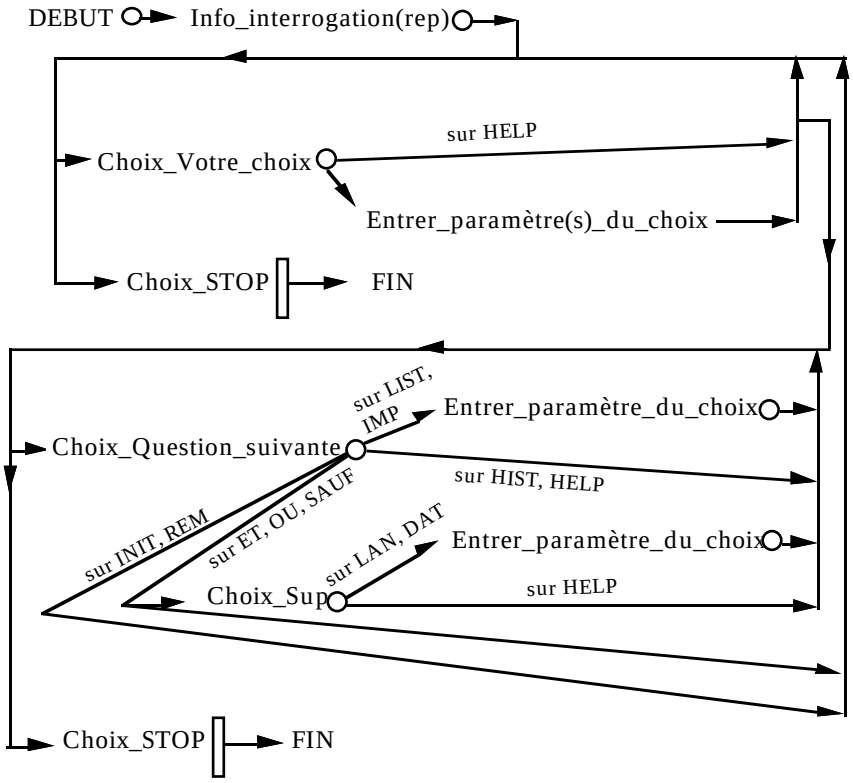


Figure 40 : Résumé du graphe des séquences pour l'interface de Médiathèque.

La version complète du graphe des séquences pour l'interface de Médiathèque est donnée en annexe 4.

4.3. Mise en œuvre de la capture

Cette interface utilisateur a été réalisée sous Unix, dans le langage Logotel [Logotel 87]. Logotel est un langage de programmation très simple qui permet

d'écrire des programmes d'interfaces entre une ou plusieurs applications Texto et un utilisateur final. Texto est, quant à lui, un langage de gestion de base de données. Le but des programmes Logotel est en général de permettre à des utilisateurs d'accéder à des applications Texto, sans connaître ni les commandes de Texto, ni les applications elles-mêmes.

4.3.1. Technique utilisée

Un programme Logotel est constitué d'une suite de modules où chaque module contient une suite d'instructions Logotel. La figure 41 donne un exemple d'un module du programme Logotel de Médiathèque. L'appel à ce module permet l'impression du détail des choix posés à l'utilisateur. La dernière instruction Logotel de ce module (“aller”) est un branchement vers un autre module. Si Logotel a fini d'exécuter un module sans avoir rencontré l'instruction “aller”, le programme est alors terminé.

```
module
info_sv          / / nom du module
action
imprimer Voulez-vous

 - Affiner votre question (par un ET) :        ET
 - Etendre votre question (par un OU) :        OU
 - Remonter a la question precedente :         REM
 - Poser une nouvelle question :               INIT
 - L'historique de la question :               HIST
 - Stopper l'interrogation :                   STOP
 - Lister les reponses a l'ecran :             LIST
 - Imprimer un listing des reponses :          IMP

aller ques_sv
```

Figure 41 : Médiathèque : le module “info_sv”. En gras : les instructions Logotel.

Dans le langage Logotel, il n'existe qu'une seule technique pour communiquer avec l'utilisateur : l'instruction Logotel “question” qui permet de poser une question à l'utilisateur après impression ou non d'un texte au terminal. La réponse de l'utilisateur est alors mémorisée dans une variable qui aura pour nom celui du module en cours d'exécution.

Nous notons, en effet, que dans les deux cas de procédure élémentaire que nous considérons (que ce soient les choix de menu ou les entrées de paramètre), il s'agit de chaînes de caractères saisies au clavier en réponse à une question posée par le logiciel Médiathèque. Pour la capture, notre tâche a ainsi consisté à rechercher chacune des instructions “question”, et à ajouter trois lignes de code qui nous permettent de sauvegarder, dans le fichier de capture, la procédure

engagée. La figure 42 reprend le module appelé pour la saisie d'un mot clef, paramètre du choix de menu MCL.

```
module
MCL
action
question Donnez  au singulier et sans articles le mot-cle ou le debut
 du mot-cle recherche  :
imprimer-fichier * /Mediatheque/Base/ouvrages/log/balbo.interro
imprimer-fichier +[date JJ:MM:AA+HH:NN:SS]@ Entrer_mot_clef([MCL])
imprimer-fichier *
si *
modifier texte : MCL=[MCL]*
aller poser_q
si niveau=1
aller choix_1
aller choix_ch
```

Figure 42 : Exemple de mise en œuvre de la capture pour Médiathèque . En gras : les instructions Logotel. L'encadré en pointillé repère l'enregistrement vers le fichier de capture.

4.3.2. Le recueil des données

Les utilisateurs du logiciel Médiathèque sont des étudiants de troisième cycle et des chercheurs désireux de savoir si un ouvrage est disponible localement à la médiathèque. L'interrogation s'effectue depuis un terminal en accès libre à la médiathèque. Les résultats donnés ci-dessous sont extraits du traitement d'un fichier regroupant 210 interrogations des ouvrages. Nous donnons un extrait de ce fichier en annexe 4.

4.4. Analyse fournie par ÉMA et interprétation des résultats

Nous détaillons ici l'analyse automatique fournie par ÉMA ainsi que les interprétations complémentaires.

Puisque le dialogue est directif et que l'utilisateur n’a pas la possibilité de naviguer, il n'y a pas de rupture possible dans l'utilisation de Médiathèque ; mais ces contraintes ont un lien avec les occurrences d’annulations immédiates.

4.4.1. Annulations immédiates

De même que pour les détections de rupture, nous ne pouvons pas dire qu'il existe de réelles possibilités d'annulation immédiate. Nous mentionnons toutefois deux des choix qui s'offrent à l'utilisateur et qui vont dans ce sens : ce sont les actions Choix_REM et Choix_INIT, deux choix de menu.

- Choix_REM permet de remonter au niveau de la question précédente en faisant abstraction du dernier terme saisi ; ce choix intervient essentiellement

lorsque l'utilisateur s'aperçoit qu'il n'y a aucune réponse à la question construite, alors qu'à l'étape précédente il y en avait au moins une.
- Choix_INIT permet de réinitialiser la question afin d'en poser une nouvelle.

L'observation de ces deux choix dans le fichier de capture est néanmoins intéressante. Ils sont très utilisées par un petit nombre d'utilisateurs et précèdent souvent Choix_STOP, ce en quoi ils peuvent être considérés comme des annulations. Ils dénotent dans ce dernier cas un problème de guidage : l'utilisateur n'est pas suffisamment informé sur les possibilités qui lui sont offertes, et notamment que Choix_STOP peut-être utilisé à tout moment pour sortir de l'interrogation des ouvrages.

4.4.2. Répétitions

Les répétitions observées se situent notamment à deux niveaux : sur les choix Choix_REM, et Choix_liste (rep).
- Le choix Choix_REM est celui cité au paragraphe précédent.
- Le choix Choix_liste (rep) permet l'affichage à l'écran de la liste des ouvrages trouvés à la Médiathèque pour la question posée. Le paramètre *rep* sera soit “C”, soit “M”, selon que l'utilisateur désire visualiser la liste avec les informations complètes (option “C”) ou minimales (option “M”) sur les ouvrages recherchés.

La répétition du choix Choix_REM est une répétition normale.
Quant à la répétition de Choix_liste, elle est quasi-systématique et est due au fait que le paramètre demandé (“C” ou “M”) doit être tapé en majuscule, alors que pour tous les autres choix, les minuscules sont acceptées (cf. extrait du fichier de capture annoté par ÉMA, donné figure 43). Nous avons ici la mise en avant d'un problème d'homogénéité.

```
+18:06:93+ 15:46:05@ media ------DEBUT-OUVRAGES------
+18:06:93+ 15:46:10@ Info_interrogation(o)
+18:06:93+ 15:46:25@ Choix_MOT
+18:06:93+ 15:46:33@ Entrer_mot_titre(AGENT)
+18:06:93+ 15:46:40@ Choix_LIST
+18:06:93+ 15:46:44@ Choix_liste(m)
+18:06:93+ 15:46:47@ Choix_liste(m)
                                                (*Répétition*)
+18:06:93+ 15:46:50@ Choix_liste(M)
                                                (*Répétition*)
+18:06:93+ 15:48:20@ Choix_STOP
+18:06:93+ 15:48:20@ ----FIN interrogation OUVRAGES----
```

Figure 43 : Extrait du fichier de capture, annoté par ÉMA, pour le logiciel Médiathèque.

4.4.3. Invalidités

Une première capture avait mis en évidence une gêne importante de la part des utilisateurs parce que toutes les commandes devaient être saisies en majuscule. Suite à ces premières observations, la saisie en minuscule a été autorisée. Seul l'oubli sur Choix_liste(rep) a entraîné la répétition observée au paragraphe précédent. Le fichier que nous avons traité ici fait suite à cette première modification.

Les invalidités observées lors de l'utilisation du logiciel Médiathèque restent nombreuses. Elles peuvent être séparées en deux sous-ensembles. Le premier sous-ensemble regroupe les retours chariots inopinés, le second, les entrées de caractères ne correspondant à aucun des choix attendus.

Lorsqu'une entrée de choix de menu est attendue, la procédure élémentaire enregistrée est la chaîne de caractères commençant par “Choix_” à laquelle sont ajoutés les caractères saisis au clavier par l'utilisateur. Ainsi les retours chariot inopinés sont repérés dans le fichier de capture par un enregistrement de la forme “+16:06:93+15:02:28@ Choix_”. Quant aux entrées de caractères qui ne correspondent à aucune des actions attendues, la chaîne entrée au clavier est ajoutée à “Choix_”. Ainsi l'on trouve dans le fichier de capture, des enregistrements de la forme: “+16:06:93+15:03:59@ Choix_ROUTING” ou “+28:06:93+ 15:12:00@ Choix_MOT”, où Choix_ROUTING et Choix_MOT ne correspondent à aucune procédure élémentaire valide du graphe des séquences.

- Les retours chariot inopinés (dont nous donnons un exemple figure 44, seconde invalidité annotée) se produisent en particulier après l'impression de la liste des ouvrages trouvés (après l'action Choix_liste (rep)). Une interprétation possible de cette observation est la suivante : lors de l'utilisation répétée du retour chariot pour passer d'une page à l'autre de l'impression des résultats, l'utilisateur a souvent tendance à entrer un retour chariot de trop. Nous ne mettrons pas en cause ici de problème particulier d'ergonomie.

```
+24:06:93 +16:54:32@ Choix_AUT
+24:06:93 +16:54:35@ Entrer_nom_auteur(hubert)
+24:06:93 +16:54:43@ Choix_LIST
+24:06:93 +16:54:45@ Choix_liste(C)
+24:06:93 +17:02:09@ Choix_LI
                                                    (*Invalide*)
+24:06:93 +17:02:12@ Choix_LIST
+24:06:93 +17:02:15@ Choix_liste(C)
+24:06:93 +17:49:28@ Choix_
                                                    (*Invalide*)
```

Figure 44 : Extrait du fichier de capture, annoté par ÉMA, pour le logiciel Médiathèque.

- Les entrées de caractères invalides sont, pour la plupart, des erreurs de frappe (cf. figure 44, première invalidité annotée). Ces erreurs obligent l'utilisateur à retaper sa dernière commande en soignant sa frappe clavier. Ici l'interface textuelle est mise en défaut, l'utilisabilité de telles interfaces n'est pas triviale, et les erreurs commises sont nombreuses. Nous soulevons à nouveau le manque de souplesse de cette interface.

4.5. Conclusion

Nous détaillons ici les résultats observés pour 8 interrogations, pour lesquelles nous fournissons, en annexe 4, les détails des fichiers annotés.

	Invalidités	Répétitions
Utilisateur no 1	2	0
Utilisateur no 2	3	0
Utilisateur no 3	3	1
Utilisateur no 4	2	1
Utilisateur no 5	1	2
Utilisateur no 6	1	0
Utilisateur no 7	1	0
Utilisateur no 8	2	0
TOTAL	**15**	**4**

Les 4 répétitions observées concernent toutes le problème d'homogénéité présenté au paragraphe 4.4.2. Quant aux invalidités, 10 concernent des retours chariots inopinés faisant suite à l'exécution de la procédure élémentaire Choix_Liste, les autres faisant référence à de mauvaises entrées.

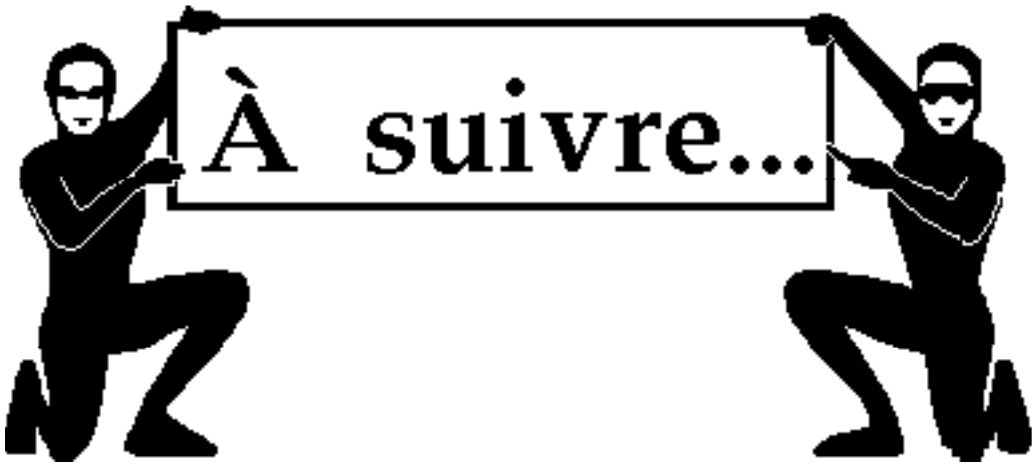
5. Conclusion

Les trois systèmes que nous avons choisis pour valider ÉMA sont très complémentaires par l'espace des tâches permises, le type d'IHM offert et les

environnements d'accueil pour la réalisation de la capture : tâches de conception et souplesse d'une interface à manipulation directe sous X Window, tâches simples grand public et un dialogue dirigé par une interface graphique simulée sur HyperCard, tâches de recherche d'information par question-réponse sur terminal alphanumérique Unix Logotel. A cette diversité, ÉMA répond par une solution commune vers l'automatisation de l'évaluation ergonomique des interfaces utilisateur. ÉMA ne permet pas de détecter tous les problèmes d'utilisabilité mais notre mécanisme d'analyse avec ses notions de rupture, d'annulation, de répétitions et d'invalidité est un progrès sur les techniques automatiques actuelles.

En tout état de cause, les détections d'ÉMA sont des indicateurs d'anomalie qu'un évaluateur expert peut utilement mettre à profit. L'étude des DAB est à ce titre assez probante : ÉMA a été capable de relever les problèmes "graves" qu'un expert humain avait par ailleurs identifiés expérimentalement sur le terrain. Pour Compo, le nombre réduit de sujets ne nous a pas permis de confirmer certaines de nos remarques, mais il a été suffisant pour soulever certains problèmes au moins une fois. Tous ces problèmes ont convergé vers un même constat : la lourdeur d'utilisation de Compo. Pour Médiathèque, ÉMA a bien mis en évidence les invalidités et les répétitions pour faute de saisie couplée à l'absence de souplesse dans la structure du dialogue.

ÉMA est inopérant pour les évaluations subjectives ou les cas simples de problèmes "mineurs" pour lesquels les utilisateurs trouvent immédiatement une parade : dans le DAB modifié, l'expression "Cancel to continue", d'abord perçue comme contradictoire, puis rapidement assimilée, est un exemple de problème non détectable par ÉMA puisqu'il ne donne pas lieu à une erreur de comportement. Cette dernière remarque ouvre la discussion sur les limites actuelles d'ÉMA et les perspectives qu'il faudrait envisager. Ces deux points font l'objet de notre chapitre de conclusion.



CONCLUSION

Cette conclusion se scinde en trois parties. Nous résumons dans un premier temps la contribution de notre travail. La deuxième partie présente les limites d'ÉMA. Quant à la troisième et dernière partie, elle expose les perspectives envisagées pour ce travail.

1. Contribution de la thèse

La contribution de ce travail s'inscrit au sein de la discipline de l'étude de l'Interaction Homme-Machine, et plus particulièrement à la mise à disposition d'outils pour l'évaluation ergonomique des IHM.

A ce titre, nous avons , dans un premier temps, clarifié l'espace dans lequel nous nous situons. Nous avons tout d'abord posé les bases de notre discussion en précisant les apports des disciplines de l'ergonomie et du génie logiciel à nos travaux, disciplines à la croisée desquelles se situe l'étude de l'Interaction Homme-Machine (chapitre I). Puis, après avoir cerné la définition de la notion de tâche, nous avons précisé le rôle des modèles et outils d'analyse supports (chapitre II), afin de définir l'apport de ces méthodes et outils dans la conception logicielle, et plus précisément pour l'évaluation des IHM. De même, nous avons produit une taxonomie des systèmes d'évaluation ergonomique des IHM (chapitre III), afin de se donner les moyens de comparer les méthodes et outils existants, et ainsi d'être en mesure de situer notre travail.

Le fruit de notre réflexion sur l'automatisation de l'évaluation a donné naissance à ÉMA, un mécanisme d'analyse d'automatique pour l'évaluation ergonomique des interfaces utilisateur (chapitre V et VI). Afin de mener à bien l'analyse, ÉMA se scinde en trois parties : deux composantes parallèles (la composante de la tâche et de l'utilisateur), et la troisième composante qui autorise, sur la base des deux premières, la génération automatique de l'analyse.

- ***La composante de l'utilisateur***
  Cette composante reflète la tâche effective de l'utilisateur où les actions de l'utilisateur sont capturées grâce à l'ajout de code dans l'interface en cours d'évaluation.

- ***La composante de la tâche***
  Nous modélisons la tâche au moyen du graphe des séquences. Ce graphe représente les cheminements possibles dans l’espace des commandes de

l'interface utilisateur. Notre modèle définit ainsi la logique de fonctionnement du système.

- ***Le mécanisme d'analyse***
  C'est à partir de l'analyse manuelle de fichiers de capture, ainsi que nos propres utilisations quotidiennes, que nous avons pu identifier des profils de comportement. A ces profils nous avons fait correspondre des règles heuristiques utilisables pour le dépouillement automatique. Ces règles de comportement sont au nombre de quatre :
  - les détections de rupture,
  - les annulations immédiates,
  - les répétitions,
  - les invalidités.

L'analyse est ainsi construite sur la base d'une recherche de ces règles de comportement au sein du fichier de capture et à l'aide du graphe des séquences.

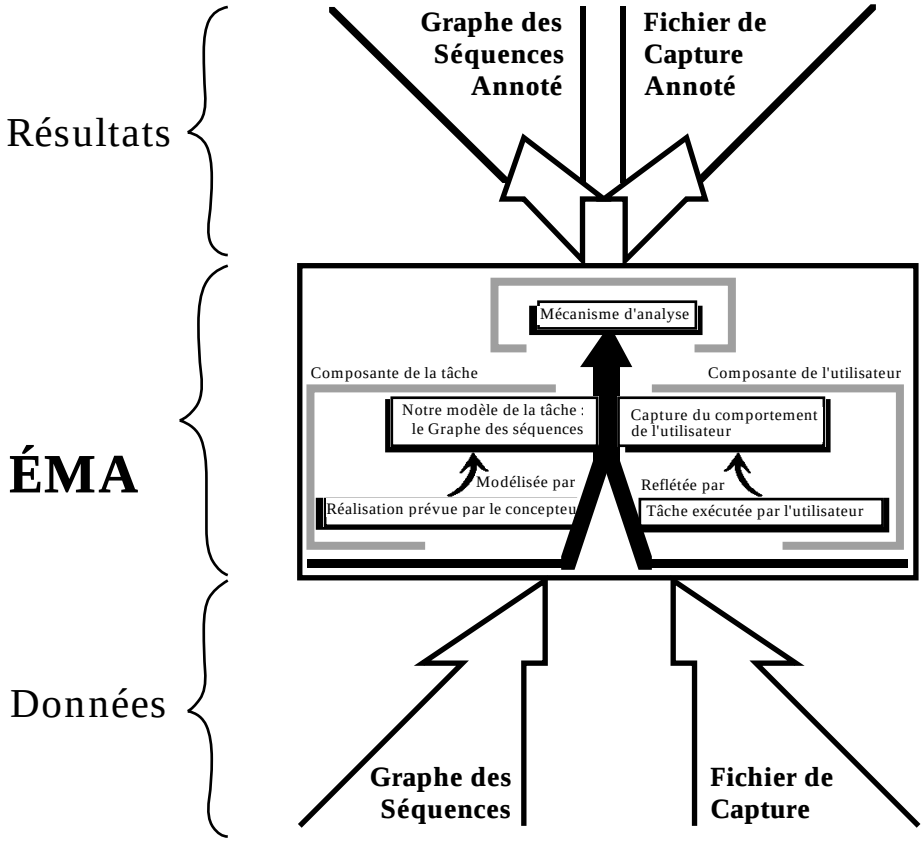


Figure 1: Principe de fonctionnement d'ÉMA

Comme le schématise la figure 1, ÉMA accepte en entrée deux données principales : le graphe des séquences et le fichier de capture. Notre mécanisme

produit, en sortie, un graphe des séquences annoté ainsi qu'un fichier de capture annoté. Les annotations localisent les points de rencontre des ruptures, des annulations immédiates, des répétitions ou des invalidités.

L'expert ergonome construit son évaluation au vu non seulement du fichier de capture annoté, mais aussi du graphe des séquences annotés. Nous lui fournissons ainsi un double pointage des anomalies détectées. En effet, au sein du fichier de capture les informations sont dispersées, mais aussi souvent trop nombreuses pour apporter une aide efficace pour l'évaluation. Alors que les annotations au sein du graphe des séquences regroupent ces anomalies autour de la représentation de la tâche adoptée.

ÉMA est un support à l'évaluation, un outil de travail pour l'étude de l'ergonomie des logiciels.

2. Les limites d'ÉMA

A l'heure actuelle nous observons une limite importante liée à la forme du graphe des séquences. En effet, pour certaines tâches de conception pure (par exemple pour le logiciel de dessin MiniDraw, sur NeXT), nous obtenons un graphe des séquences à deux niveaux (c'est-à-dire où chacun des fils de la racine de l'arbre est une feuille, comme le schématise la figure 2). Pour de telles applications, une fois le logiciel lancé, l'utilisateur peut à tout moment exécuter n'importe laquelle des procédures élémentaires qui lui est offerte.

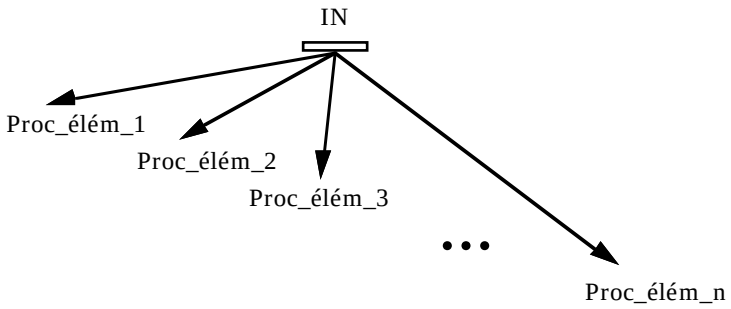


Figure 2 : Forme d'un graphe des séquences à deux niveaux

Sur un tel graphe, il n'existe pas de rupture de séquence. Ainsi ÉMA est limité dans les résultats qu'il construit aux détections sur le fichier de capture.

En fait, nous constatons que notre mécanisme s'adapte bien à des interfaces pour lesquelles l'utilisation est structurée ou mixte, c'est-à-dire si le graphe des séquences associé inclut plusieurs niveaux.

3. Les perspectives de travail

Nous décomposons les perspectives en deux sous-ensembles : celles à court et moyen terme, et celles à long terme. Les perspectives à court et moyen terme énoncent des dispositions qui peuvent être envisagées dans un avenir proche. Pour la mise en place des perspectives à long termes, nous ne disposons pas à l'heure actuelle de toutes les réponses. Ces perspectives posent des questions qui nécessitent une avancée importante au niveau recherche, tant sur le plan informatique qu'ergonomique.

3.1. Les perspectives à court et moyen terme

Les perspectives à court et moyen terme sont au nombre de trois : l'ajout de fonctionnalités à ÉMA, l'évaluation d'interfaces multimodales ainsi que l'intégration d'ÉMA à un générateur d'interfaces.

Ajout de fonctionnalités à ÉMA

L'ajout de fonctionnalité s'envisage en collaboration avec des ergonomes. Une telle collaboration n'a pas pu être réalisée dans le cadre de cette thèse, mais elle devient indispensable avant de poursuivre plus avant. Cette collaboration permettra, d'une part, de confronter ÉMA à la réalité des évaluations ergonomiques d'interfaces utilisateur et ainsi de déterminer les fonctions supplémentaires que doit posséder un bon outil d'évaluation afin d'en agrémenter ÉMA.

D'ores et déjà nous avons quelques idées à ce propos :

- mettre en évidence dans le fichier de capture des actions particulières ainsi que dans le fichier du graphe des séquences. Cela peut être utile, comme nous l'avons souligné lors de la conclusion de l'expérimentation conduite sur les ATM (chapitre VI, paragraphe 3.5),
- se donner les moyens, à partir de la trace produite, de rejouer l'interaction,
- mettre à disposition des concepteurs un outil statistique qui autoriserait :
    - le calcul du nombre d'exécutions de chaque procédure élémentaire,
    - l'observation des délais entre procédures élémentaires,
    - la mise en évidence des procédures élémentaires jamais utilisées,
    - la prise en compte de plusieurs fichiers de capture pour une même interface,
    - etc.

Evaluation d'interfaces multimodales

Nous n'avons pas eu l'occasion de tester l'apport d'ÉMA pour des logiciels interactifs utilisant d'autres médias que le clavier ou la souris. Pour ce faire, nous comptons intégrer ÉMA à NEIMO [Salber 93].

La technologie actuelle de la communication Homme-Machine, et notamment la reconnaissance de la parole et du geste, impose le plus souvent des limitations contraignantes. L'objectif de NEIMO est de contourner ces difficultés avec un Magicien d'Oz. Cette technique consiste à faire exécuter par un compère humain, dissimulé, les fonctions manquantes d'un système interactif utilisé par un utilisateur représentatif accomplissant une tâche déterminée, au moyen d'une interface multimodale. Ainsi nous pourrons aborder, progressivement, les formes langagières de la communication selon des modalités moins usuelles : parole, expressions faciale et gestuelle.

Intégration d'ÉMA au sein d'un générateur d'interface

Il est essentiel de penser, dès les premières étapes du cycle de développement des logiciels, aux moyens que l'on se donnera pour l'évaluation. Aussi l'intégration d'ÉMA au sein d'un générateur d'interface peut-elle ouvrir la voie dans cette direction en sensibilisant les différents concepteurs (ergonomes et informaticiens) aux possibilités offertes pour répondre positivement à cette constatation.

De plus, dans l'état actuel, ÉMA est un outil lourd d'utilisation. En effet, il faut d'une part générer le graphe des séquences, et d'autre part implémenter la capture au sein de l'interface cible. Il est donc nécessaire de fournir au concepteurs des moyens afin de l'aider dans ces deux tâches. Pour ce faire, encore une fois, l'intégration d'ÉMA dans un générateur d'interfaces utilisateur est intéressante pour les deux points précisés ci-dessus.

- L'insertion du code nécessaire à la capture s'envisage bien plus aisément lors de la génération du code de l'interface utilisateur, qu'a posteriori, comme c'est le cas actuellement.
- D'autre part, pourquoi ne pas concevoir automatiquement le graphe des séquences à partir de la formalisation crée pour le générateur ? Cette seconde hypothèse, bien que plus complexe que la précédente, nous semble intéressante à envisager.

Toutes ces perspectives s'envisagent afin de consolider le rôle d'ÉMA dans la tâche de l'évaluation ergonomique des interfaces utilisateur.

3.2. Les perspectives à plus long terme

Les perspectives à long terme envisagent non plus ÉMA comme un mécanisme d'analyse automatique, mais comme un outil d'évaluation à part entière.

L'évaluation d'une interface ne peut se satisfaire de l'observation des seules actions sur les dispositifs en entrées. Le méta-discours a aussi sa part dans la communication : commentaires oraux (cf. techniques de "thinking aloud"), mouvements du regard [Jacob 90], localisation de la bouche [Yuille 91] ou expression faciale [Terzopoulos 90]. Ces actes constituent une source importante d'information dont la fusion doit servir à lever des incertitudes sur l'interprétation à apporter au comportement de l'utilisateur. On constate une situation analogue en robotique mobile où les données en provenance de capteurs hétérogènes sont fusionnées en vue d'augmenter la robustesse du système. Aussi, afin de compléter le rôle d'ÉMA dans l'évaluation, serait-il intéressant de prendre en compte ce méta-discours pour la construction du critique. Ce critique peut s'envisager selon la méthode QOC [McLean 91] (cf. chapitre I, figure 6), c'est-à-dire qu'il propose plusieurs hypothèses et le concepteur ergonome ou informaticien choisit celle qui lui convient le mieux.

Les perspectives à long terme envisagent ainsi des évolutions d'ÉMA qui le font sortir du rôle qu'il a à l'heure actuelle, à savoir un mécanisme d'analyse automatique pour l'évaluation ergonomique des interfaces utilisateur, pour le faire entrer dans la catégorie des outils offrant un critique automatique.

ANNEXES

Annexe 1

Principes d'utilisation des diagrammes syntaxiques

Les diagrammes syntaxiques permettent de décrire la syntaxe d'un langage.

La syntaxe d'un langage est l'étude de la forme que doivent avoir les énoncés du langage, indépendamment de leur sens et de leur utilisation.

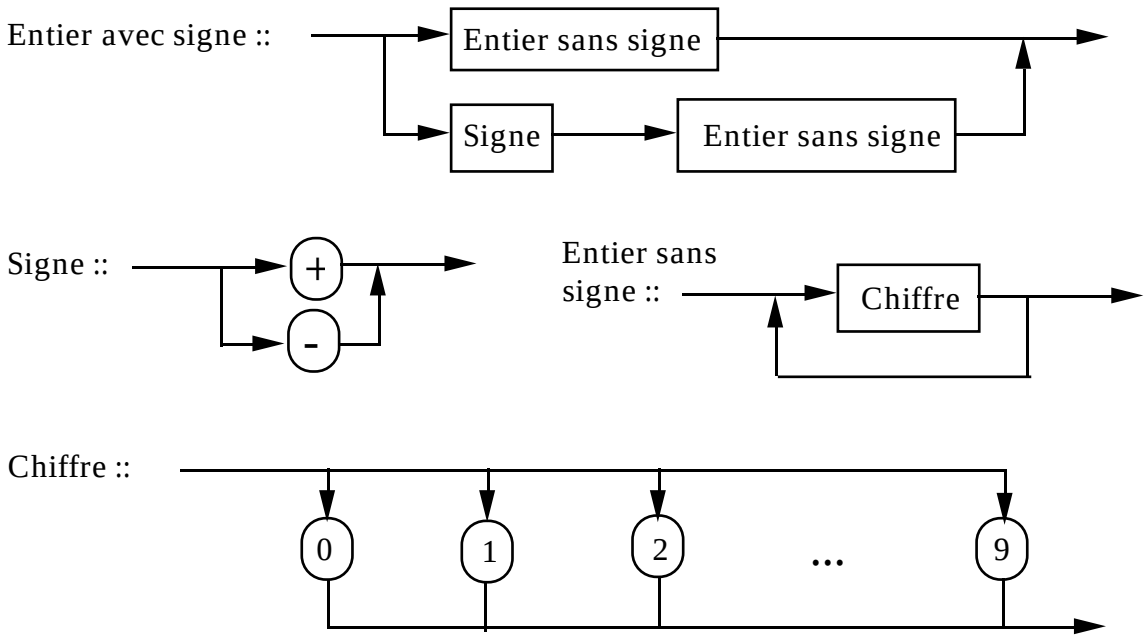
La syntaxe est exprimée par une grammaire, ensemble de règles qui permettent de construire et de reconnaître les énoncés permis du langage.

Une règle décrit un énoncé permis à partir d'énoncés plus simple, ceci jusqu'aux énoncés réduits aux mots ou signes du langage qu'on appelle les symboles terminaux.

A chaque règle est associée une variable syntaxique qui dénote l'ensemble des énoncés décrits par la règle ou qui désigne un élément quelconque de cet ensemble (c'est-à-dire un énoncé quelconque décrit par la règle).

Les symboles terminaux sont dans des encadrés à bords ronds. Les variables syntaxiques sont dans des encadrés à bords carrés et sont définis en fonction d'autres variables syntaxiques ou de symboles terminaux.

Par exemple, les diagrammes syntaxiques associés à la définition d'un entier avec signe sont les suivants :



Annexe 2

Plate-forme d'expérimentation : Compo

1. Rapport technique sur la construction du fichier de capture

Le fichier de nom scrute.c regroupe les fonctions qui vont permettre de scruter des procédures élémentaires. Ces fonctions viennent se greffer dans le code du logiciel en des points clés.

La liste des fonctions disponibles est la suivante :

ScruteInit ()
ScruteEnd ()

ScruteCb(descriptif, evt, widgetID)
ScruteCbForIconPath(widgetID, IPstruct, PtrStructDonnee)
ScruteScrollCb(w,client,cb)

ScruteSxt(evt)
ScruteRealizeToplevel (descriptif, ID)

ScruteIsToplevel (Widg)

Pour chacune de ces fonctions nous donnerons :

- ses modalités d'utilisation
- un exemple d'utilisation
- le code C associé

Pour toute utilisation d'une des fonctions du fichier scrute.c, il faut inclure les fichiers suivants :

```
#include "scrute.h"
#include "scrute.itf"
```

ScruteInit

Initialisation de l'horloge pour la scrutation des actions de l'utilisateur

Exemple d'utilisation

ScruteInit ();

Code C

```
/*   ***********************************************
  NAME : ScruteInit
  PARAMETERS :
      INPUT : none
      OUTPUT : none
  RESULT : none
```

```
   AIM : Initialization for the scrutinisation
  MODIFICATION:
*************************************************    */
void ScruteInit ()
{
  time_t nb;
  struct tm *ht; /* hour of the treatment : courent hour */
 char cmde[300];
 FILE *fd;
 int i;

  /* pour recup des secondes et microsecondes : 6/12 */
  struct timeval tp;
  struct timezone tzp;
  long m,s;

  /* initialization of the courent hour */
  time(&nb);
  ht = localtime(&nb);

  gettimeofday(&tp,&tzp);
  seconds = tp.tv_sec;
  micro = tp.tv_usec;

   sprintf(fichier,"%s/Scrute/scrute.%d_%d.%d_%d_%d",
          DIREXEC,
           ht->tm_mday,ht->tm_mon+1,
           ht->tm_hour,ht->tm_min,ht->tm_sec);

  fd = fopen(fichier,"a");

 if (fd != NULL){
    fprintf(fd,"START THE APPLICATION AT :: %d:%d:%d  ::  %ld+%ld\n\n",
              ht->tm_hour,ht->tm_min,ht->tm_sec,
             seconds,micro);
   fclose(fd);
  }
  else {
    printf("scrute.c : ScruteInit can't open file %s\n", fichier);
  }

  /* Init of the ToplevelList */
  for(i=0;i<=MaxToplevel;i++)  ToplevelList[i] = (elt_ToplevelList *) NULL;
}
```

ScruteEnd

Libération de la place mémoire utilisée pour la gestion de la scrutation

Exemple d'utilisation

```
ScruteEnd();
```

Code C

```
/*    ***********************************************
   NAME : ScruteEnd
   PARAMETERS : none
   RESULT : none
   AIM : free the lists in ToplevelList
   MODIFICATION:
************************************************    */
void ScruteEnd()
{
  int i;
   elt_ToplevelList *elt, *next;

   for(i=0;i<=MaxToplevel;i++) {
    elt = ToplevelList[i];
    while ( elt != (elt_ToplevelList *) NULL ) {
     next = elt->next;
     Free (elt);
     elt = next;
    }
  }
}
```

ScruteCb

Récupération d'une action de l'utilisateur à l'entrée d'une procédure Callback

Exemple d'utilisation

```
static void MenuFileOpen( Widg, tag, PtrStructDonnee)
Widget  Widg;
StrPac *tag;
XmAnyCallbackStruct *PtrStructDonnee;
{
...
#ifdef SCRUTE  /* code added by SBalbo, 22.1.92 */
  if ((Widg!=NULL) && (PtrStructDonnee!=NULL))
   {
     n=0;
      XtSetArg(args[n],XmNlabelString,&oklab);n++;
      XtGetValues(Widg,args,n);
       XmStringGetLtoR (oklab, XmSTRING_DEFAULT_CHARSET, &label);
        sprintf(descript,"**MenuFileOpen**--**derouler menu 'Fichier/%s'", label);

    ScruteCb (descript,
                PtrStructDonnee->event,
                Widg);
   }
```

```
#endif  /* end code added by SBalbo, 22.1.92 */
...
}
```

Code C

```
/*    ***********************************************
   NAME : ScruteCb
   PARAMETERS :
         INPUT : descriptif, evt, widgetID
         OUTPUT :
   RESULT : none
   AIM : scrutinization of the events inside the callbacks,
        fills the file : Scrute/scrute.date.time
   MODIFICATION:
************************************************    */
void ScruteCb(descriptif, evt, widgetID)

     char *descriptif; /* description of the event */
      XEvent *evt; /* minimal structure of the third
                                  parameter in the callback procedure */
     Widget widgetID; /* widgetID of the widget concerned
                         on va chercher le toplevel correspondant a ce widget */
{
  char descript[300];
  FILE *fd;
   struct tm *ht; /* hour of the treatment : current hour */
   Time millisec, he, dividende, quotient;
  int me, se, s100e;
   time_t nb;

   /* pour recup des secondes et microsecondes : 6/12 */
   struct timeval tp;
   struct timezone tzp;
   long m,s;

    /* initialisation of the current hour */
    time(&nb);
    ht = localtime(&nb);

   /* init ds s et m du nombre des secondes et microsecondes ecoulees
      depuis le demarrage de l'application                          */
   gettimeofday(&tp,&tzp);
   s = tp.tv_sec;
   m = tp.tv_usec;
   if (seconds == s) {
     m = m - micro;
    s = 0;
   }
   else {
```

```
  s = s - seconds;
}

debug ("scrute.c : begin of ScruteCb\n        desc = %s\n",descriptif);

if (evt == (XEvent *)NULL){
  debug ("scrute.c : ScruteCb : cas evt a NULL\n");
  he = 0; me = -1; se = -1; s100e = -1;
}
else {
   /* initialisation of the time of the event */
   millisec = evt->xbutton.time; /* millisecondes */

   /* convertion of the millisec in he, me, se & s100e */
   quotient = (Time) (millisec/1000);
   s100e = millisec - quotient*1000;

   dividende = quotient;
   quotient = (Time) (dividende/60);
   se = dividende - quotient*60;

   he = (Time) (quotient/60);
   me = quotient - he*60;
}

 /* recherche du toplevelID associe au widgetID */
 debug("\n\navant boucle\n");
 while (ScruteIsToplevel((long) XtWindow (widgetID)) == SCRUTE_NOT_TL) {

   debug("inside boucle : widgetID = %ld\n",(long) XtWindow (widgetID));

  widgetID = XtParent(widgetID);
}
 debug("apres boucle\n\n\n");
 /* fin de la recherche */

if ( (he == 0) && (me == -1) && (se == -1) && (s100e == -1))
    sprintf (descript,"%ld+%ld\#\#**(%ld)%s\n",
             s,m,
             (long) XtWindow (widgetID),descriptif);
else
    sprintf (descript,"%ld+%ld\#%d:%d:%d:%d\#**(%ld)%s\n",
             s,m,
             he,me,se,s100e,
             (long) XtWindow (widgetID),descriptif);

 fd = fopen(fichier,"a");
if (fd != NULL){
  fprintf(fd,descript);
```

```
   fclose(fd);
  }
  else {
    printf("scrute.c : ScruteCb can't open file %s\n", fichier);
  }

  debug("scrute.c : end of ScruteCb\n");
}
```

ScruteCbForIconPath

Procédure Callback que l'on associe au chemin iconique pour les cas particuliers dans lesquels aucune callback n'est associee par l'application

Exemple d'utilisation

```
/*------ add callback */
if (IconPathSt->idRoot!=NULL)
   XtAddCallback(IconPathTableOfWidget[NumIconPathButton],
                        XmNactivateCallback, IconPaletteActivate, IconPathSt);

#ifdef SCRUTE /* SBalbo, 11.2.92, add a cb pour scruter dans tous les cas */
else {
        XtAddCallback(IconPathTableOfWidget[NumIconPathButton],
                              XmNactivateCallback,
         ScruteCbForIconPath, IconPathSt);
}
#endif /* end of SBalbo, 11.2.92 */
```

Code C

```
/*    ***********************************************
   NAME : ScruteCbForIconPath
   PARAMETERS :
        INPUT : widgetID, IPstruct, PtrStructDonnee
        OUTPUT :
   RESULT : none
   AIM : Cb associee au chemin iconique pour les cas particuliers
        dans lesquels aucune cb n'est associee par l'application
  MODIFICATION:
************************************************    */
void ScruteCbForIconPath(widgetID, IPstruct, PtrStructDonnee)
    Widget widgetID;
    IconPathStruct *IPstruct;
     XmAnyCallbackStruct *PtrStructDonnee;
{
 Arg args[3];
 char *label;
 XmString oklab;
 char descript[200];
 int n;
```

```
  debug("Entree dans ScruteCbForIconPath, IPstruct->text = %s\n",IPstruct->text);

  n=0;
   XtSetArg(args[n],XmNlabelString,&oklab);n++;
   XtGetValues(IPstruct->label,args,n);

    XmStringGetLtoR (oklab, XmSTRING_DEFAULT_CHARSET, &label);

    sprintf(descript,"**Chemin Iconique**selectionner '%s'", label);
   ScruteCb (descript,PtrStructDonnee->event,widgetID);

   debug ("sortie de ScruteCbForIconPath\n");
}
```

ScruteSxt

Scrutinisation des événements à l'interieur de la mainloop.
Seuls les événements en direction d'une toplevel nous interessent.

Exemple d'utilisation

```
/*    ***********************************************
    NAME : testevent
    PARAMETERS :
        INPUT :
        OUTPUT :
    RESULT : int
    AIM : Check event for the part in MOTIF.
          The res is 0 if there is no event or 1
          if there is an X event.
          Don't wait for an X event just check.
    MODIFICATION:
************************************************     */
int testevent()
{
XEvent e;

while(XtPending())
 {
   if (XtPeekEvent(&e))
   {
     XtNextEvent(&e);

#ifdef SCRUTE     /* SBalbo, 25.11.91 : add the scrutinisation */
     ScruteSxt (&e);
#endif

     XtDispatchEvent(&e);
     return(1);
   }
 }
```

```
return (0);
}
```

Code C

```
/*    ***********************************************
   NAME : ScruteSxt
   PARAMETERS :
        INPUT : evt
        OUTPUT :
   RESULT : none
   AIM : scrutinization of the events inside the mainloop,
        we are interested only in the events for the toplevel
        fills the file : Scrute/scrute.date.time
  MODIFICATION:
************************************************    */
void ScruteSxt( evt)

     XEvent *evt; /* minimal structure of the third
                                parameter in the callback procedure */
{
  FILE *fd;
  char desc[100];
   struct tm *ht; /* hour of the treatment : current hour */
   time_t nb;
  long windowID;
  int inToplevelList = 0; /* =1 si windowID est une toplevel */
  int i;
   elt_ToplevelList *elt;

  /* pour recup des secondes et microsecondes : 6/12 */
  struct timeval tp;
  struct timezone tzp;
  long m,s;

  if (evt == (XEvent *)NULL){
    debug ("scrute.c : ScruteSxt : cas evt a NULL\n");
   strcpy(desc," ");
  }
  else {
     /* initialization of the descrition of the event */
    switch (evt->type) {

/*  **************début  commentaire  ************
        case ButtonPress:
          strcpy(desc,"ButtonPress");
         break;
        case ButtonRelease:
          strcpy(desc,"ButtonRelease");
         break;
```

```
        case CirculateNotify :
          strcpy(desc,"CirculateNotify");
          break;
        case CirculateRequest:
          strcpy(desc,"CirculateRequest");
          break;
  ******************** fin commentaire *************/

      case ClientMessage:
          strcpy(desc,"ClientMessage");
          break;

/** **************début commentaire ************
        case ColormapNotify:
          strcpy(desc,"ColormapNotify");
          break;
******************** fin commentaire *************/

        case ConfigureNotify:
          strcpy(desc,"ConfigureNotify");
          break;
        case ConfigureRequest:
          strcpy(desc,"ConfigureRequest");
          break;
        case CreateNotify:
          strcpy(desc,"CreateNotify");
          break;
        case DestroyNotify:
          strcpy(desc,"DestroyNotify");
          break;

/* **************début commentaire ************
        case EnterNotify:
          strcpy(desc,"EnterNotify");
          break;
        case Expose:
          strcpy(desc,"Expose");
          break;
******************** fin commentaire ***************/

        case FocusIn:
          strcpy(desc,"FocusIn");
          break;

/* **************début commentaire ************
        case FocusOut:
          strcpy(desc,"FocusOut");
          break;
        case GraphicsExpose:
```

```
          strcpy(desc,"GraphicsExpose");
          break;
        case GravityNotify:
          strcpy(desc,"GravityNotify");
          break;
        case KeymapNotify:
          strcpy(desc,"KeymapNotify");
          break;
        case KeyPress:
          strcpy(desc,"KeyPress");
          break;
        case KeyRelease:
          strcpy(desc,"KeyRelease");
          break;
        case LeaveNotify:
          strcpy(desc,"LeaveNotify");
          break;
******************** fin commentaire **************/

        case MapNotify:
          strcpy(desc,"MapNotify");
          break;

/* **************début commentaire ************
        case MappingNotify:
          strcpy(desc,"MappingNotify");
          break;
        case MapRequest:
          strcpy(desc,"MapRequest");
          break;
        case MotionNotify:
          strcpy(desc,"MotionNotify");
          break;
        case NoExpose:
          strcpy(desc,"NoExpose");
          break;
******************** fin commentaire **************/

        case PropertyNotify:
          strcpy(desc,"PropertyNotify");
          break;

/* **************début commentaire ************
        case ReparentNotify:
          strcpy(desc,"ReparentNotify");
          break;
        case ResizeRequest:
          strcpy(desc,"ResizeRequest");
          break;
```

```
        case SelectionClear:
          strcpy(desc,"SelectionClear");
          break;
        case SelectionNotify:
          strcpy(desc,"SelectionNotify");
          break;
        case SelectionRequest:
          strcpy(desc,"SelectionRequest");
          break;
******************** fin commentaire **************/

        case UnmapNotify:
          strcpy(desc,"UnmapNotify");
          break;

/* **************début commentaire ************
        case VisibilityNotify:
          strcpy(desc,"VisibilityNotify");
          break;
******************** fin commentaire **************/

        default:
          strcpy(desc," ");

          break;
   } /* end switch (evt->type) */

   if (strcmp(desc," ")==0) {
     /* do nothing */
   }
   else {

     windowID = (long)(evt->xany).window;
    i = 0;
     inToplevelList = 0; /* Faux */

      while ((i<=MaxToplevel) && (!inToplevelList)) {
        elt = ToplevelList[i];
        while ((elt != (elt_ToplevelList *) NULL) &&
             (elt->ID != windowID)             ) {

          elt = elt->next;
        }
        i++;
        inToplevelList = ((elt != (elt_ToplevelList *) NULL) &&
                          (elt->ID == windowID)            );
     } /* end while (inToplevelList) */

     if (inToplevelList == 1) {
```

```
      debug ("inside scrute.c : ScruteSxt, intoplevel = 1 : windowId = %ld\n",
                        windowID);

       /* initialisation of the current hour */
       time(&nb);
       ht = localtime(&nb);

       /* init ds s et m du nombre des secondes et microsecondes ecoulees
          depuis le demarrage de l'application                          */
       gettimeofday(&tp,&tzp);
       s = tp.tv_sec;
       m = tp.tv_usec;
       if (seconds == s) {
         m = m - micro;
        s = 0;
       }
       else {
        s = s - seconds;
       }
       fd = fopen(fichier,"a");
       if (fd != NULL){
          fprintf(fd,/*%d:%d:%d-*/"%ld+%ld\##**(%ld)**%s\n",
                  s,m,
                  windowID,desc);
        fclose(fd);
       }
       else {
         printf("scrute.c : ScruteSxt can't open file %s\n", fichier);
       }
       debug("  FFFFFF IIIIIIII NNNNNNNNN     de ScruteSxt\n");
    }/* end if (inToplevelList == 1) */
   } /* end if (strcmp(desc," ")==0) */
  } /* end if (evt == (XEvent *)NULL) */

}
```

ScruteScrollCb

Callback qu'il faut associer aux barres de défilement pour que l'on puisse repérer les mouvements dans ces dernières. A la callback, sera passé un identificateur qui permettra de reconnaitre la barre de défilement concernée. La déclaration du tableau client sera alors nécessaire pour identifier le type de barre de défilement.

Exemple d'utilisation

```
#ifdef SCRUTE  /* SBalbo : 19.2.92 */
static int client[15] = {
         SCRUTE_SB_WS_H;     /* Working Space, barre Horizontale */
        SCRUTE_SB_WS_V;      /* Working Space, barre Verticale */
         SCRUTE_SB_IP;       /* Icon Path */
```

```
        SCRUTE_SB_LE_H;      /* Liste d'Erreur, barre Horizontale */
        SCRUTE_SB_LE_V;      /* Liste d'Erreur, barre Verticale */
        SCRUTE_SB_GI_H ;     /* Get Informations, barre Horizontale */
        SCRUTE_SB_GI_V;      /* Get Informations, barre Verticale */
        SCRUTE_SB_PD_H ;     /* Pull Down, barre Horizontale */
        SCRUTE_SB_PD_V;      /* Pull Down, barre Verticale */
        SCRUTE_SB_O_H ;      /* Ouvrir..., barre Horizontale */
        SCRUTE_SB_O_V;       /* Ouvrir..., barre Verticale */
 };
#endif /* end of SBalbo : 19.2.92 */
...

#ifdef SCRUTE  /* SBalbo : 19.2.92 */
  Widget w_horiz, w_vertic;
#endif /* end of SBalbo : 19.2.92 */
...
#ifdef SCRUTE      /* SBalbo, 19.2. 92 , scrutation des mvt ds ScrollBar*/
/* récupération des numéros de widget des barres de défilement
    de la zone de travail                                          */
 n=0;
  XtSetArg(arg[n], XmNhorizontalScrollBar, &w_horiz); n++;
  XtSetArg(arg[n], XmNverticalScrollBar, &w_vertic); n++;
  XtGetValues ( idP->TabWidget[NumScrolWorkingSpace], arg,n);

  XtAddCallback(w_horiz,XmNvalueChangedCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_H]);
  XtAddCallback(w_horiz,XmNdecrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_H]);
  XtAddCallback(w_horiz,XmNincrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_H]);
  XtAddCallback(w_horiz,XmNpageDecrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_H]);
  XtAddCallback(w_horiz,XmNpageIncrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_H]);

  XtAddCallback(w_vertic,XmNvalueChangedCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_V]);
  XtAddCallback(w_vertic,XmNdecrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_V]);
  XtAddCallback(w_vertic,XmNincrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_V]);
  XtAddCallback(w_vertic,XmNpageDecrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_V]);
  XtAddCallback(w_vertic,XmNpageIncrementCallback,
              ScruteScrollCb, &client[SCRUTE_SB_WS_V]);


/* récupération des numéros de widget des barres de défilement
    du chemin iconique                                             */
```

```
  /*------ Berangere, IconPath -------*/
   AddScrollForIconPath(&w_horiz,&w_vertic);
  /*------ Berangere, end ------------*/

  XtAddCallback(w_horiz,XmNvalueChangedCallback,
                ScruteScrollCb, &client[SCRUTE_SB_IP]);
  XtAddCallback(w_horiz,XmNdecrementCallback,
                ScruteScrollCb, &client[SCRUTE_SB_IP]);
  XtAddCallback(w_horiz,XmNincrementCallback,
                ScruteScrollCb, &client[SCRUTE_SB_IP]);
  XtAddCallback(w_horiz,XmNpageDecrementCallback,
                ScruteScrollCb, &client[SCRUTE_SB_IP]);
  XtAddCallback(w_horiz,XmNpageIncrementCallback,
                ScruteScrollCb, &client[SCRUTE_SB_IP]);

#endif  /* End of SBalbo, 19.2. 92 */
```

Code C

```
/*    ***********************************************
   NAME : ScruteScrollCb
   RESULT : none
   AIM : Cb used to scrutinize the events associated to the scrollbars
   MODIFICATION:
************************************************    */
void ScruteScrollCb  (w,tag,cb)
    Widget w;
    int *tag;  /* cste definie ds scrute.itf */
     XmScrollBarCallbackStruct *cb;

{
 char descript[200];

  debug("entree ds ScruteScrollCb\n");

 switch (cb->reason) {
  case XmCR_INCREMENT:
    debug ("       XmCR_INCREMENT.....\n");
   break;
  case XmCR_DECREMENT:
    debug ("       XmCR_DECREMENT.....\n");
   break;
  case XmCR_PAGE_INCREMENT:
    debug ("       XmCR_PAGE_INCREMENT.....\n");
   break;
  case XmCR_PAGE_DECREMENT:
    debug ("       XmCR_PAGE_DECREMENT.....\n");
   break;
  case XmCR_VALUE_CHANGED:
    debug (" XmCR_VALUE_CHANGED.....\n"); break;
```

```
 default: printf("AUTRE VALEUR : I M P O S S I B L E \n"); break;
 }

  strcpy(descript,"**ScruteScrollCb");

 switch (*tag) {
  case SCRUTE_SB_WS_H :
    strcat(descript,"**Working Space**scroll horizontal");
  break;
  case SCRUTE_SB_WS_V :
    strcat(descript,"**Working Space**scroll vertical");
  break;
 case SCRUTE_SB_IP :
    strcat(descript,"**Icon Path**scroll horizontal");
  break;
 case SCRUTE_SB_LE_H :
    strcat(descript,"**Liste d'Erreur**scroll horizontal");
  break;
 case SCRUTE_SB_LE_V :
    strcat(descript,"**Liste d'Erreur**scroll vertical");
  break;
 case SCRUTE_SB_GI_H :
     strcat(descript,"**Fenetre d'Information**scroll horizontal");
  break;
 case SCRUTE_SB_GI_V :
     strcat(descript,"**Fenetre d'Information**scroll vertical");
  break;
 case SCRUTE_SB_PD_H :
    strcat(descript,"**Pull Down Palette**scroll horizontal");
  break;
 case SCRUTE_SB_PD_V :
   strcat(descript,"**Pull Down Palette**scroll vertical");
  break;
 case SCRUTE_SB_O_H :
    strcat(descript,"**Fenetre Ouvrir...**scroll horizontal");
  break;
 case SCRUTE_SB_O_V :
    strcat(descript,"**Fenetre Ouvrir...**scroll vertical");
  break;
 default :
     printf("scrute.c : ScruteScrollCb, P R O B L E M E ...\n");
 }

  debug("inside ScruteScrollCb, descript = %s\n", descript);

 ScruteCb (descript, cb->event, w);
  debug("sortie de ScruteScrollCb \n");

}
```

ScruteRealizeToplevel

Fonction appelée à chaque création d'une toplevel, elle va enregistrer le numéro de cette toplevel et dans le fichier d'observation cette création sera notée

Exemple d'utilisation

```
XtRealizeWidget(toplevel);

#ifdef SCRUTE    /* SBalbo, 29.11.91 */
  ScruteRealizeToplevel(
        "**DisplayAppliLogic**Vue Logique de l'application",
        TLVueLogA,title, (long) XtWindow (toplevel));
#endif    /******************/
```

Code C

```
/*    ***********************************************
   NAME : ScruteRealizeToplevel
   PARAMETERS :
        INPUT : descriptif,TLtype,TLtitle,TLid
        OUTPUT :
   RESULT : none
   AIM : insert in the ToplevelList the given ID, and
        fills the file Scrute/scrute.date.time with the
         given descriptif and ID
  MODIFICATION:
************************************************    */
void ScruteRealizeToplevel(descriptif,TLtype,TLtitle,TLid)
    char *descriptif;
    int TLtype; /* type du Toplevel, cf. ~/C/Incl/scrute.itf */
    char *TLtitle;
    long TLid;

{
 FILE *fd;
  elt_ToplevelList *elt;

  /* pour recup des secondes et microsecondes : 6/12 */
  struct timeval tp;
  struct timezone tzp;
  long m,s;

  /* code added by SBalbo, 22.1.92 */
  Arg args[3];
  int n=0;
  char *label;
  XmString oklab;
  /* end code added by SBalbo, 22.1.92 */

  /* init ds s et m du nombre des secondes et microsecondes ecoulees
     depuis le demarrage de l'application                    */
```

```
  gettimeofday(&tp,&tzp);
  s = tp.tv_sec;
  m = tp.tv_usec;
  if (seconds == s) {
    m = m - micro;
   s = 0;
  }
  else {
   s = s - seconds;
  }

  debug ("inside ScruteRealizeToplevel :::: TLid = %d, descriptif = %s\n",
         TLid,descriptif);

  elt = (elt_ToplevelList *) Malloc (sizeof(elt_ToplevelList));

 elt->ID = TLid;
 elt->next = ToplevelList[TLtype];

 ToplevelList[TLtype] = elt;

  /* remplissage ds Scrute/scrute.date.time */
  fd = fopen(fichier,"a");
  if (fd != NULL){
    fprintf(fd,"OUVERTURE TOPLEVEL : %s (%ld) [%s] ... DATE : %ld+%ld\n",
            descriptif, TLid, TLtitle, s, m);
   fclose(fd);
  }
  else {
     printf("scrute.c : ScruteRealizeToplevel can't open file %s\n", fichier);
  }
}
```

ScruteIsToplevel

Fonction bouléenne qui renvoit SCRUTE_TL or SCRUTE_NOT_TL selon que le Widget passé en parametre appartient a la liste des toplevels ou non.

Exemple d'utilisation

if (**ScruteIsToplevel**((long) XtWindow (XtParent(Widg))) == SCRUTE_NOT_TL)

Code C

```
/*    ********************************************
   NAME : ScruteIsToplevel
   PARAMETERS : long
   RESULT : SCRUTE_TL or SCRUTE_NOT_TL selon que le Widget
         passe en parametre appartient a la liste des
           toplevels ou pas
   AIM :
   MODIFICATION:
```

```
*************************************************    */
int ScruteIsToplevel ( windowID )
    long windowID;
{
  int inToplevelList; /* =1 si windowID est une toplevel, 0 sinon */
  int i;
   elt_ToplevelList *elt;

  i = 0;
  inToplevelList = 0; /* Faux */

   while ((i<=MaxToplevel) && (!inToplevelList)) {
    elt = ToplevelList[i];
     while ((elt != (elt_ToplevelList *) NULL) &&
            (elt->ID != windowID)            ) {
      elt = elt->next;
    }
    i++;
     inToplevelList = ((elt != (elt_ToplevelList *) NULL) &&
                       (elt->ID == windowID)            );
  } /* end while (inToplevelList) */

  if (inToplevelList == 1)   return  (SCRUTE_TL);
  else                       return  (SCRUTE_NOT_TL);
}
```

2. Fichiers de capture

Le schéma du fichier de capture ne suit pas rigoureusement la définition qui en a été faite dans le corps du mémoire (chapitre IV). A cela, deux raisons :

1. la capture des actions au sein du logiciel Compo a été implémentée bien avant que le format du fichier de capture ressemble à ce que nous vous avons présenté au chapitre IV.
2. le fonctionnement même du traitement des actions élémentaires dans l'environnement X n'est pas aussi trivial que pour des interfaces utilisateur telles les ATM ou Médiathèque. En effet, il n'y a pas de bijection entre les actions à détecter et les événements informatiques générés par l'utilisateur et traités par la machine. Nous avons ainsi été amenés à construire cette traduction. Aussi, même à l'heure actuelle, la construction du fichier de capture aurait dû s'effectuer en deux temps :

   première étape : capture sous X des événements

   deuxième étape : retranscription, dans notre format, de cette capture.

2.1. Format adopté initialement

ligne 1 : **START THE APPLICATION AT :: *hh*:*mm*:*ss* :: *sec*+*s100***

Les autres lignes du fichier respectent l'un des formats suivants :

Action gérée par Compo :

ss+s100#HH:MM:SS:S100#**(no)**nom_cb**nom_de_l'action

Création de topLevel :

OPEN TOPLEVEL : ***nom_fen*** **(*no*) ... DATE :** ***ss+s100***

Action gérée par mwm :

ss+s100##**(no)**type_de_l'événement_X

Action gérée par motif :

ss+s100##**(no)**type_de_l'action

où :

sec est le nombre de secondes écoulées depuis le 1er janvier 1970,

hh:mm:ss donne l'heure système,

ss et ***s100*** donnent le nombre de secondes et millisecondes écoulées entre le lancement du logiciel et l'enregistrement de l'action,

HH, ***MM***, ***SS*** et ***S100*** donnent la date, lorsqu'elle est connue, de l'événement associé à l'action (date donnée en millisecondes),

nom_fen donne le titre associé à la fenêtre créée,

no reprend le numéro associé au topLevel correspondant à la fenêtre,

type_de_l'événement_X est le type X de l'événement

nom_de_l'action reprend le nom d'une tâche élémentaire de l'arbre des séquences,

type_de_l'action_syntaxique donne le type de l'action syntaxique gérée par motif,

nom_cb est le nom de la callback appelée pour le traitement

2.2. Fichier de capture

START THE APPLICATION AT :: 19:3:45 :: 759607425+327137

OUVERTURE TOPLEVEL : **palette**Fenetre MultiLangues (7340049) [???] ... DATE : 22+514768
24+906796##**(7340049)**ConfigureNotify
24+931957##**(7340049)**ConfigureNotify
24+950451##**(7340049)**MapNotify
25+177970##**(7340049)**FocusIn
26+231389#1025:54:8:941#**(7340049)**MultiLangActivate**fenetre multilangue**selectionner 'OK'

OUVERTURE TOPLEVEL : **DisplayPalette**Palette Principale (7340072) [PALETTE PRINCIPALE] ... DATE : 26+986185
27+153081##**(7340049)**UnmapNotify
27+252423##**(7340049)**DestroyNotify
28+793278##**(7340072)**ConfigureNotify
28+831361##**(7340072)**ConfigureNotify
28+897170##**(7340072)**MapNotify
28+981763##**(7340072)**FocusIn
29+950281##**(7340072)**FocusIn
30+30820#1025:54:12:665#**(7340072)**activateAppli**Palette principale**selectionner 'Application'
OUVERTURE TOPLEVEL : **DisplayAppliPalette**Palette des Applications (7340207) [PALETTE DES APPLICATIONS] ... DATE : 33+662504
34+466574##**(7340072)**FocusIn
34+513271##**(7340207)**ConfigureNotify
34+548013##**(7340207)**ConfigureNotify
34+565134##**(7340207)**MapNotify
34+826280##**(7340207)**FocusIn
36+234348#1025:54:18:812#**(7340207)**ButtonActivateAppli**Palette**selectionner 'DemoPape'
38+929351#1025:54:21:627#**(7340207)**MenuFileOpen**--**derouler menu 'Fichier/Ouvrir selection'
OUVERTURE TOPLEVEL : **DisplayAppliStruct**Vue Structurelle de l'application (7340383) [APPLICATION :: DemoPape] ... DATE : 42+122967
42+246270##**(7340207)**FocusIn
43+156911##**(7340207)**FocusIn
43+183175##**(7340383)**ConfigureNotify
43+204751##**(7340383)**ConfigureNotify
43+239491##**(7340383)**MapNotify
43+447827##**(7340383)**FocusIn
44+653894#1025:54:27:342#**(7340383)**ButtonActivateInStructViewAppli**Vue Structurelle de l'application**selectionner 'pap_c1'
47+465719#1025:54:30:171#**(7340383)**MenuFileOpen**--**derouler menu 'Fichier/Ouvrir selection'
OUVERTURE TOPLEVEL : **DisplayScenarioStruct**Vue Structurelle du scenario (7340542) [SCENARIO :: pap_c1 === Type :: PREMIER] ... DATE : 49+695213
49+835925##**(7340383)**FocusIn
51+530388##**(7340383)**FocusIn
51+564745##**(7340542)**ConfigureNotify
51+584076##**(7340542)**ConfigureNotify
51+602534##**(7340542)**MapNotify
51+783303##**(7340542)**FocusIn
53+543323#1025:54:36:200#**(7340542)**ButtonActivateInStructViewScenario**Vue Structurelle du scenario**selectionner 'curseur1'
56+58162#1025:54:38:762#**(7340542)**MenuFileOpen**--**derouler menu 'Fichier/Ouvrir selection'
OUVERTURE TOPLEVEL : **DisplayGroupStruct**Vue Structurelle du groupement (7340727) [GROUPEMENT :: curseur1 === Type :: REMPLACEMENT] ... DATE : 57+997654
58+89755##**(7340542)**FocusIn
59+869648##**(7340542)**FocusIn
59+891021##**(7340727)**ConfigureNotify
59+912612##**(7340727)**ConfigureNotify
59+981979##**(7340727)**MapNotify
60+141739##**(7340727)**FocusIn
63+336821#1025:54:45:531#**(7340727)**ButtonActivateInStructViewGroup**Vue Structurelle du groupement**selectionner 'QuelleJauge'
66+753221#1025:54:49:391#**(7340727)**MenuOptionIdraw**--**derouler menu 'Options/Vue Graphique'

OUVERTURE TOPLEVEL : **DisplayInfoVG**Info graphique (7340773) [INFO GRAPHIQUE : curseur1] ... DATE : 67+368805
79+689071##**(7340727)**FocusIn
80+124172##**(7340727)**FocusIn
80+144124##**(7340773)**ConfigureNotify
80+209101##**(7340773)**ConfigureNotify
80+225414##**(7340773)**MapNotify
80+353777##**(7340773)**FocusIn
80+388698##**(7340773)**FocusIn
80+433430##**(7340773)**ConfigureNotify
87+972890##**(7340727)**FocusIn
89+879093#1025:55:12:436#**(7340727)**MenuFileOpen**--**derouler menu 'Fichier/Ouvrir selection'
OUVERTURE TOPLEVEL : **DisplayObjectStruct**Vue Structurelle de l'objet (7340915) [OBJET :: QuelleJauge === Type :: JAUGE] ... DATE : 96+609821
96+772562##**(7340727)**FocusIn
131+923590##**(7340727)**FocusIn
131+963712##**(7340915)**ConfigureNotify
131+992825##**(7340915)**ConfigureNotify
132+9655##**(7340915)**MapNotify
132+224622##**(7340915)**FocusIn
134+820285#1025:55:57:528#**(7340915)**MenuOptionIdraw**--**derouler menu 'Options/Vue Graphique'
OUVERTURE TOPLEVEL : **DisplayInfoVG**Info graphique (7340972) [INFO GRAPHIQUE : QuelleJauge] ... DATE : 135+562680
153+593066##**(7340915)**FocusIn
154+39299##**(7340915)**FocusIn
154+72971##**(7340972)**ConfigureNotify
154+93535##**(7340972)**ConfigureNotify
154+109430##**(7340972)**MapNotify
154+211056##**(7340972)**FocusIn
154+245734##**(7340972)**FocusIn
154+295518##**(7340972)**ConfigureNotify
191+612226##**(7340915)**FocusIn
199+156556#1025:57:1:645#**(7340915)**MenuType**--**derouler menu 'Type/Objet Baladeur'
200+666904##**(7340915)**FocusIn
209+153998#1025:57:11:818#**(7340915)**MenuFileQuit**--**derouler menu 'Fichier/Quitter'
OUVERTURE TOPLEVEL : **MenuFileQuit**Message de confirmation sur Quit (7340996) [MESSAGE DE CONFIRMATION : Voulez vous reellement quitter l'application] ... DATE : 210+506849
210+559177##**(7340915)**FocusIn
210+695201##**(7340996)**ConfigureNotify
210+763478##**(7340996)**MapNotify
210+948101##**(7340996)**FocusIn
212+637403#1025:57:15:346#**(7340996)**MsgConfirmationCbQuit**fenetre MESSAGE DE CONFIRMATION : Voulez vous reellement quitter l'application **selectionner 'Confirmer'
OUVERTURE TOPLEVEL : **MsgSave1**Message de confirmation (7340999) [MESSAGE DE CONFIRMATION : Modification sur le fichier : /Share/Network/gadget/balbo/Compo/Application/DemoPape/pap_c1/curseur1.GPT] ... DATE : 213+153617
213+181811##**(7340996)**UnmapNotify
213+205418##**(7340996)**DestroyNotify
213+237881##**(7340915)**FocusIn
213+476799##**(7340999)**ConfigureNotify
213+579767##**(7340999)**MapNotify
213+608860##**(7340999)**FocusIn
216+740727#1025:57:19:450#**(7340999)**MsgCbSave1**selectionner 'Sauver'

OUVERTURE TOPLEVEL : **MsgSave1**Message de confirmation (7341002) [MESSAGE DE CONFIRMATION : Modification sur le fichier :
/Share/Network/gadget/balbo/Compo/Application/DemoPape/pap_c1/curseur1/QuelleJauge.OBJ] ... DATE : 218+855458
218+883909##**(7340999)**UnmapNotify
218+908405##**(7340999)**DestroyNotify
218+925370##**(7340915)**FocusIn
219+23687##**(7341002)**ConfigureNotify
219+100413##**(7341002)**MapNotify
219+130722##**(7341002)**FocusIn
220+844500#1025:57:23:554#**(7341002)**MsgCbSave1**selectionner 'Sauver'
235+547176##**(7341002)**UnmapNotify

Annexe 3

Plate-forme d'expérimentation : Distributeurs automatiques de billets

1. Extrait du script du DAB Initial, sans retour sur les choix effectués

Stack: ATM Initial

16 cards; 4 backgrounds.
Stack script:

```
-- This script contains action definitions associated with the
-- following parts of the prototype:

--      Interface Implementation
--      Domain Simulation
--      Machine Cash Simulation

-- The OpenStack definition is associated with the Interface
-- Implementation. The definitions associated with the Domain
-- Simulation are:

--      AccountBalance
--     DebitBalance

-- The definitions associated with the Machine Cash Simulation are:

--      MachineCash
--      MachineCashWithdrawal

on OpenStack
  -- Supresses menuBar and TitleBar

  -- hide menuBar
  hide titleBar

  -- Sandrine, 7 August 92 ---------
  -- the file ObsFile will contain the observation of the user's action

  global ObsFile

  put "Obs at " & the seconds into ObsFile
  open file ObsFile

  write "ATM Prototype ----------" & return to file ObsFile
  write "Beginning of the session at : " & the long time & return to file ObsFile

  -- Sandrine, end -----------------
end OpenStack

-- Sandrine, 7 August 92 --------
on CloseStack
```

```
  global ObsFile

  write "End of the session at : " & the long time & return to file ObsFile

  close file ObsFile
end CloseStack

on Record Chaine
  global ObsFile

  write the long time & " @ " & Chaine & return to file ObsFile
end Record
-- Sandrine, end --------------------
```

...

Background 3: bkgnd id 6173

9 cards; 17 background buttons; 0 background fields.

Background 3 script:

```
-- Sandrine, 8/8/92 ---------
on mouseUp
  Record ("Bkgnd_mouseUp")
end mouseUp

on keyDown theKey
  Record ("Bkgnd_keyDown_on_(" & theKey & ")")
  send "keyDown theKey" to HyperCard
end keyDown
-- End Sandrine ----------
```

Background button 1: 1

Visible.

Background button 1 script:

```
on mouseUp
  -- Sandrine, 8/8/92 -------
  Record ("Click_number(1)")
  -- end Sandrine -----------

  -- instantiates buttonPressed procedure with the name of the button

  put the short name of me into theName
  buttonPressed theName
end mouseUp
```

Background button 11: Bottom Button

Visible.

Background button 11 script:

```
on mouseUp
  -- Sandrine, 8/8/92 -------
    Record ("Click_Bottom_Button")
  -- end Sandrine -----------

    -- instantiates buttonPressed procedure with the name of the button

  put the short name of me into theName
  buttonPressed theName
end mouseUp
```

Card field 2: Displayed PIN

Visible.

Card field 2 script:

```
-- Sandrine, 9/8/92
on mouseUp
    Record ("Click_in_Displayed_PIN_field_from_Get_PIN_card")
end mouseUp
-- end Sandrine
```

Card field 1: Cash

Visible.

Card field 1 script:

```
on mouseUp
  -- Sandrine, 9/8/92
    Record ("Click_Handover_Cash")
  -- end Sandrine

  -- goes to card "Return Card"

  go to card "Return Card"
end mouseUp
```

2. DAB initial

2.1. Graphe des séquences initial

```
(* Arbre des séquences pour l'ATM initial
        Création : Sandrine BALBO
        Date : 11.08.93
*****************************************)

DEBUT
        In                              -I->    Enter_Card (Noun) .
        Enter_Card                      -I->    Enter_PIN_1 (Digit) .
```

```
      Enter_PIN_1                -I->   Enter_PIN_2 (Digit) .
      Enter_PIN_2                -I->   Enter_PIN_3 (Digit) .
      Enter_PIN_3                -I->   Enter_PIN_4 (Digit) .
      Enter_PIN_4                -I->   Select_Withdraw_Yes ;
                                        Select_Withdraw_No ;
                                        Remove_Card (Noun) .
      Select_Withdraw_Yes        -I->   Enter_Amount (Digit) ;
                                        Select_Cancel .
      Select_Withdraw_No         -I->   Remove_Card (Noun) .
      Enter_Amount               --->   Select_Valid ;
                                        Select_Cancel .
      Remove_Card                -I->   Enter_Card (Noun) ; Out .
      Select_Cancel              -I->   Remove_Card (Noun) .
      Select_Valid               -I->   Remove_Cash ;
                                        Remove_Card (Noun) .
      Remove_Cash                -I->   Remove_Card (Noun) .
FIN
```

2.2. Graphe des séquences annoté

```
IN      -I->
                          ENTER_CARD ( NOUN )
                        (* Invalide :
                           10:19:53 @ BKGND_KEYDOWN_ON( 1)
                        *) .

ENTER_CARD      -I->
                          ENTER_PIN_1 ( DIGIT ) .

ENTER_PIN_1     -I->
                          ENTER_PIN_2 ( DIGIT ) .

ENTER_PIN_2     -I->
                          ENTER_PIN_3 ( DIGIT )
                        (* Invalide :
                           10:20:17 @ BKGND_BUTTON_NAME ( TOPBOTTOM,MOUSEUP)
                           10:20:26 @ DISPLAYED_PIN_FIELD_FROM ( GETPINCARD,MOUSEUP)
                           10:20:35 @ BKGND_KEYDOWN_ON( 0)
                        *) .

ENTER_PIN_3     -I->
                          ENTER_PIN_4 ( DIGIT )
                        (* Invalide :
                           10:48:39 @ BKGND_BUTTON_NAME ( 0,MOUSEUP)
                        *) .

ENTER_PIN_4     -I->
                           SELECT_WITHDRAW_YES ;
                           SELECT_WITHDRAW_NO ;
                          REMOVE_CARD .

SELECT_WITHDRAW_YES       -I->
                           ENTER_AMOUNT ( DIGIT )
                         (* Repetition :
                          10:20:59 @ ENTER_AMOUNT ( 0)
```

```
                        10:28:26 @ ENTER_AMOUNT ( 0)
                        10:38:45 @ ENTER_AMOUNT ( 0)
                        10:39:23 @ ENTER_AMOUNT ( 5)
                        10:46:17 @ ENTER_AMOUNT ( 0)
                        10:46:58 @ ENTER_AMOUNT ( 5)
                      *) ;
                          SELECT_CANCEL ( DIGIT ) .

SELECT_WITHDRAW_NO      -I->
                          REMOVE_CARD ( NOUN ) .

ENTER_AMOUNT      --->
                        SELECT_VALID ;
                        SELECT_CANCEL .

REMOVE_CARD     -I->
                          ENTER_CARD ( NOUN )
                     (* Invalide :
                        10:28:06 @ BKGND_KEYDOWN_ON( 4)
                        10:28:07 @ BKGND_KEYDOWN_ON( 0)
                        10:28:08 @ BKGND_KEYDOWN_ON( 0)
                        10:28:08 @ BKGND_KEYDOWN_ON( 0)
                        10:45:45 @ BKGND_KEYDOWN_ON( 5)
                        10:45:46 @ BKGND_KEYDOWN_ON( 0)
                        10:45:47 @ BKGND_KEYDOWN_ON( 0)
                         10:48:29 @ BKGND_BUTTON_NAME ( STEPHEN,MOUSEUP)
                    *);
                          OUT ( NOUN ) .

SELECT_CANCEL     -I->
                          REMOVE_CARD ( NOUN ) .

SELECT_VALID      -I->
                        REMOVE_CASH ;
                        REMOVE_CARD .

REMOVE_CASH     -I->
                          REMOVE_CARD ( NOUN )
                     (* Invalide :
                        10:45:33 @ BKGND_KEYDOWN_ON(
                        10:45:34 @ BKGND_KEYDOWN_ON(
                    *) .
```

2.3. Fichier de capture annoté

```
(*################################*)
(*# TANIA AS EILEEN ON ATM 1, FIRST SIMULATION*)

(*ATM PROTOTYPE ----------*)
10:10:34 @ IN
10:19:41 @ ENTER_CARD (                    EILEEN)
10:19:53 @ BKGND_KEYDOWN_ON( 1)
                        (* Invalide *)
10:19:58 @ ENTER_PIN_1 ( 1)
10:20:02 @ ENTER_PIN_2 ( 0)
10:20:02 @ ENTER_PIN_3 ( 0)
```

```
10:20:17 @ BKGND_BUTTON_NAME ( TOPBOTTOM,MOUSEUP)
                        (* Invalide *)
10:20:26 @ DISPLAYED_PIN_FIELD_FROM ( GETPINCARD,MOUSEUP)
                        (* Invalide *)
10:20:35 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:20:38 @ ENTER_PIN_4 ( 0)
10:20:48 @ SELECT_WITHDRAW_YES
10:20:55 @ ENTER_AMOUNT ( 3)
10:20:59 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:21:10 @ SELECT_VALID
10:21:17 @ REMOVE_CASH
10:21:35 @ REMOVE_CARD ( EILEEN)


(*################################*)
(*# ANDY AS ADAM ON ATM 1, FIRST SIMULATION*)

(*ATM PROTOTYPE ----------*)
10:28:03 @ ENTER_CARD ( ADAM)
10:28:06 @ BKGND_KEYDOWN_ON( 4)
                        (* Invalide *)
10:28:07 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:28:08 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:28:08 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:28:11 @ ENTER_PIN_1 ( 4)
10:28:13 @ ENTER_PIN_2 ( 0)
10:28:13 @ ENTER_PIN_3 ( 0)
10:28:13 @ ENTER_PIN_4 ( 0)
10:28:19 @ SELECT_WITHDRAW_YES
10:28:24 @ ENTER_AMOUNT ( 3)
10:28:26 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:28:30 @ SELECT_VALID
10:28:36 @ REMOVE_CASH
10:28:45 @ REMOVE_CARD ( ADAM)

10:28:48 @ ENTER_CARD ( ADAM)
10:28:51 @ ENTER_PIN_1 ( 4)
10:28:51 @ ENTER_PIN_2 ( 0)
10:28:52 @ ENTER_PIN_3 ( 0)
10:28:52 @ ENTER_PIN_4 ( 0)
10:28:58 @ SELECT_WITHDRAW_NO
10:29:07 @ REMOVE_CARD ( ADAM)

(*###################################*)
(*# BECKY AS STEPHEN ON ATM 1, SECOND SIMULATION*)

(*ATM PROTOTYPE ----------*)
10:38:11 @ ENTER_CARD ( STEPHEN)
10:38:14 @ ENTER_PIN_1 ( 5)
10:38:15 @ ENTER_PIN_2 ( 0)
10:38:16 @ ENTER_PIN_3 ( 0)
```

```
10:38:16 @ ENTER_PIN_4 ( 0)
10:38:22 @ SELECT_WITHDRAW_YES
10:38:44 @ ENTER_AMOUNT ( 3)
10:38:45 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:38:47 @ SELECT_VALID
10:39:00 @ REMOVE_CARD ( STEPHEN)

10:39:10 @ ENTER_CARD ( STEPHEN)
10:39:12 @ ENTER_PIN_1 ( 5)
10:39:13 @ ENTER_PIN_2 ( 0)
10:39:13 @ ENTER_PIN_3 ( 0)
10:39:17 @ ENTER_PIN_4 ( 0)
10:39:20 @ SELECT_WITHDRAW_YES
10:39:22 @ ENTER_AMOUNT ( 2)
10:39:23 @ ENTER_AMOUNT ( 5)
                        (* Repetition *)
10:39:25 @ SELECT_VALID
10:39:29 @ REMOVE_CASH
10:39:35 @ REMOVE_CARD ( STEPHEN)


(*##################################*)
(*# PETER AS STEPHEN ON ATM 1, FIRST SIMULATION*)

(*ATM PROTOTYPE ----------*)
10:45:33 @ BKGND_KEYDOWN_ON(
                        (* Invalide *)
10:45:34 @ BKGND_KEYDOWN_ON(
                        (* Invalide *)
10:45:39 @ ENTER_CARD ( STEPHEN)
10:45:45 @ BKGND_KEYDOWN_ON( 5)
                        (* Invalide *)
10:45:46 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:45:47 @ BKGND_KEYDOWN_ON( 0)
                        (* Invalide *)
10:45:54 @ ENTER_PIN_1 ( 5)
10:45:55 @ ENTER_PIN_2 ( 0)
10:45:56 @ ENTER_PIN_3 ( 0)
10:45:57 @ ENTER_PIN_4 ( 0)
10:46:08 @ SELECT_WITHDRAW_YES
10:46:16 @ ENTER_AMOUNT ( 3)
10:46:17 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:46:20 @ SELECT_VALID
10:46:33 @ REMOVE_CARD ( STEPHEN)

10:46:35 @ ENTER_CARD ( STEPHEN)
10:46:39 @ ENTER_PIN_1 ( 5)
10:46:40 @ ENTER_PIN_2 ( 0)
10:46:41 @ ENTER_PIN_3 ( 0)
10:46:42 @ ENTER_PIN_4 ( 0)
10:46:48 @ SELECT_WITHDRAW_YES
10:46:56 @ ENTER_AMOUNT ( 2)
10:46:58 @ ENTER_AMOUNT ( 5)
                        (* Repetition *)
```

```
10:47:01 @ SELECT_VALID
10:47:13 @ REMOVE_CASH
10:47:27 @ REMOVE_CARD ( STEPHEN)

10:47:31 @ ENTER_CARD ( STEPHEN)
10:47:35 @ ENTER_PIN_1 ( 5)
10:47:36 @ ENTER_PIN_2 ( 0)
10:47:36 @ ENTER_PIN_3 ( 0)
10:47:37 @ ENTER_PIN_4 ( 0)
10:47:41 @ SELECT_WITHDRAW_NO
10:47:54 @ REMOVE_CARD ( STEPHEN)

10:47:55 @ ENTER_CARD ( STEPHEN)
10:47:58 @ ENTER_PIN_1 ( 5)
10:47:59 @ ENTER_PIN_2 ( 0)
10:48:00 @ ENTER_PIN_3 ( 0)
10:48:01 @ ENTER_PIN_4 ( 0)
10:48:14 @ SELECT_WITHDRAW_NO
10:48:27 @ REMOVE_CARD ( STEPHEN)

10:48:28 @ ENTER_CARD ( STEPHEN)
10:48:29 @ BKGND_BUTTON_NAME ( STEPHEN,MOUSEUP)
                        (* Invalide *)
10:48:37 @ ENTER_PIN_1 ( 0)
10:48:37 @ ENTER_PIN_2 ( 0)
10:48:38 @ ENTER_PIN_3 ( 0)
10:48:38 @ ENTER_PIN_4 ( 0)
10:48:39 @ BKGND_BUTTON_NAME ( 0,MOUSEUP)
                        (* Invalide *)
10:48:41 @ REMOVE_CARD ( STEPHEN)

10:48:42 @ ENTER_CARD ( STEPHEN)
10:48:50 @ ENTER_PIN_1 ( 5)
10:48:51 @ ENTER_PIN_2 ( 5)
10:48:51 @ ENTER_PIN_3 ( 5)
10:48:51 @ ENTER_PIN_4 ( 5)
10:48:53 @ REMOVE_CARD ( STEPHEN)
10:48:59 @ OUT
```

3. DAB modifié

3.1. Graphe des séquences initial

```
(* Arbre des séquences pour l'ATM modifié
        Création : Sandrine BALBO
        Date : 11.08.93
****************************************)

DEBUT
        In                      -I->    Enter_Card (Noun) .
        Enter_Card              -I->    Enter_PIN_1 (Digit) ;
                                        Select_Cancel .
        Enter_PIN_1             -I->    Enter_PIN_2 (Digit) ;
                                        Select_Cancel  .
```

```
      Enter_PIN_2                -I->  Enter_PIN_3 (Digit) ;
                                       Select_Cancel  .
      Enter_PIN_3                -I->  Enter_PIN_4 (Digit) ;
                                       Select_Cancel  .
      Enter_PIN_4                -I->  Select_Withdraw ;
                                       Select_Balance ; Select_End ;
                                       Remove_Card (Noun) .
      Select_Cancel_Continue     -I->  Select_Withdraw ;
                                       Select_Balance ; Select_End .
      Select_Cancel              -I->  Remove_Card (Noun) .
      Select_Withdraw            -O->  Enter_Amount (Digit) ;
                                       Select_Cancel_Continue .
      Select_Balance             -I->  Select_Cancel_Continue .
      Enter_Amount               --->  Select_Valid ;
                                       Select_Cancel_Continue.
      Select_End                 -I->  Remove_Card(Noun) .
      Remove_Card                -I->  Enter_Card (Noun) ; Out .
      Select_Valid               -I->  Remove_Cash ;
                                       Select_Cancel_Continue .
      Remove_Cash                -I->  Select_Withdraw ;
                                       Select_Balance ; Select_End .
FIN
```

3.2. Graphe des séquences annoté

```
IN      -I->
                        ENTER_CARD ( NOUN )

ENTER_CARD      -I->
                        ENTER_PIN_1 ( DIGIT ) ;
                        SELECT_CANCEL ( DIGIT ) .

ENTER_PIN_1     -I->
                        ENTER_PIN_2 ( DIGIT ) ;
                        SELECT_CANCEL ( DIGIT ) .

ENTER_PIN_2     -I->
                        ENTER_PIN_3 ( DIGIT ) ;
                        SELECT_CANCEL ( DIGIT ) .

ENTER_PIN_3     -I->
                        ENTER_PIN_4 ( DIGIT ) ;
                        SELECT_CANCEL ( DIGIT ) .

ENTER_PIN_4     -I->
                        SELECT_WITHDRAW ;
                        SELECT_BALANCE
                    (* Invalide :
                                                    10:35:53 @ BKGND_BUTTON_NAME (
BOTTOM_BUTTON,MOUSEDOWN)
                        10:35:53 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                    *);
                        SELECT_END ;
                        REMOVE_CARD .
```

```
SELECT_CANCEL_CONTINUE      -I->
                              SELECT_WITHDRAW
                            (* Invalide :
                               10:36:15 @ BKGND_KEYDOWN_ON( 2)
                               10:36:15 @ BKGND_KEYDOWN_ON( 5)
                            *);
                              SELECT_BALANCE
                            (* Invalide :
                                                              10:24:00 @ BKGND_BUTTON_NAME (
MIDDLE_BOTTOM,MOUSEDOWN)
                               10:24:00 @ BKGND_BUTTON_NAME ( MIDDLE_BOTTOM,MOUSEUP)
                            *);
                              SELECT_END .

SELECT_CANCEL      -I->
                              REMOVE_CARD ( NOUN ) .

SELECT_WITHDRAW       -O->
                              ENTER_AMOUNT ( DIGIT )
                            (* Invalide :
                               10:23:10 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:23:16 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                                                              10:23:26 @ AMOUNT_FIELD_FROM (
GET_AMOUNT_CARD,MOUSEUP)
                               10:23:33 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:23:35 @ BKGND_BUTTON_NAME ( TOP_BOTTOM,MOUSEUP)
                               10:23:38 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:24:32 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:24:34 @ BKGND_BUTTON_NAME ( TOP_BOTTOM,MOUSEUP)
                               10:24:40 @ BKGND_BUTTON_NAME ( MIDDLE_BOTTOM,MOUSEUP)
                               10:50:15 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:50:20 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:50:25 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                               10:50:25 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                            *)
                              (* Repetition :
                               10:23:05 @ ENTER_AMOUNT ( 0)
                               10:24:24 @ ENTER_AMOUNT ( 0)
                               10:30:14 @ ENTER_AMOUNT ( 0)
                               10:36:24 @ ENTER_AMOUNT ( 5)
                               10:50:14 @ ENTER_AMOUNT ( 0)
                               10:50:47 @ ENTER_AMOUNT ( 0)
                               10:51:15 @ ENTER_AMOUNT ( 5)
                            *) ;
                                SELECT_CANCEL_CONTINUE ( DIGIT ) .

SELECT_BALANCE      -I->
                                SELECT_CANCEL_CONTINUE
                              (* Annulation :
                               10:24:11 @ SELECT_CANCEL_CONTINUE
                               10:30:51 @ SELECT_CANCEL_CONTINUE
                               10:36:00 @ SELECT_CANCEL_CONTINUE
                               10:51:28 @ SELECT_CANCEL_CONTINUE
                            *) .

ENTER_AMOUNT      --->
```

```
                        SELECT_VALID
                      (* Invalide :
                         10:50:34 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                         10:50:53 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                      *);
                        SELECT_CANCEL_CONTINUE
                      (* Annulation :
                        10:23:49 @ SELECT_CANCEL_CONTINUE
                      *).

SELECT_END      -I->
                        REMOVE_CARD ( NOUN ).

REMOVE_CARD     -I->
                        ENTER_CARD ( NOUN );
                       OUT ( NOUN ).

SELECT_VALID    -I->
                        REMOVE_CASH ;
                        SELECT_CANCEL_CONTINUE
                     (* Annulation :
                       10:50:36 @ SELECT_CANCEL_CONTINUE
                       10:50:58 @ SELECT_CANCEL_CONTINUE
                     *).

REMOVE_CASH     -I->
                        SELECT_WITHDRAW ;
                       SELECT_BALANCE ;
                       SELECT_END .
```

3.3. Fichier de capture annoté

```
(*2. ATM MODIFIé @ FICHIERS DE CAPTURE*)

(*# ################################*)
(*# TANIA AS EILEEN - ATM 2, SECOND SIMULATION*)

(*ATM  NO  2----------*)
10:22:31 @ IN
10:22:42 @ ENTER_CARD ( EILEEN)
10:22:45 @ ENTER_PIN_1 ( 1)
10:22:47 @ ENTER_PIN_2 ( 0)
10:22:47 @ ENTER_PIN_3 ( 0)
10:22:48 @ ENTER_PIN_4 ( 0)
10:22:55 @  SELECT_WITHDRAW
10:23:03 @ ENTER_AMOUNT ( 3)
10:23:05 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:23:10 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:23:16 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:23:26 @ AMOUNT_FIELD_FROM ( GET_AMOUNT_CARD,MOUSEUP)
                        (* Invalide *)
10:23:33 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
```

```
10:23:35 @ BKGND_BUTTON_NAME ( TOP_BOTTOM,MOUSEUP)
                        (* Invalide *)
10:23:38 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:23:49 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:23:59 @ SELECT_BALANCE
10:24:00 @ BKGND_BUTTON_NAME (  MIDDLE_BOTTOM,MOUSEDOWN)
                        (* Invalide *)
10:24:00 @ BKGND_BUTTON_NAME ( MIDDLE_BOTTOM,MOUSEUP)
                        (* Invalide *)
10:24:11 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:24:16 @ SELECT_WITHDRAW
10:24:22 @ ENTER_AMOUNT ( 3)
10:24:24 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:24:32 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:24:34 @ BKGND_BUTTON_NAME ( TOP_BOTTOM,MOUSEUP)
                        (* Invalide *)
10:24:40 @ BKGND_BUTTON_NAME ( MIDDLE_BOTTOM,MOUSEUP)
                        (* Invalide *)
10:25:21 @ SELECT_VALID
10:25:32 @ REMOVE_CASH
10:25:37 @ SELECT_END
10:25:42 @ REMOVE_CARD ( EILEEN)

(*###################################*)
(*# ANDY AS ADAM ON ATM 2, SECOND SIMULATION*)

(*ATM  NO  2----------*)
10:29:54 @ ENTER_CARD ( ADAM)
10:29:58 @ ENTER_PIN_1 ( 4)
10:30:01 @ ENTER_PIN_2 ( 0)
10:30:01 @ ENTER_PIN_3 ( 0)
10:30:02 @ ENTER_PIN_4 ( 0)
10:30:09 @ SELECT_WITHDRAW
10:30:13 @ ENTER_AMOUNT ( 3)
10:30:14 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:30:17 @ SELECT_VALID
10:30:26 @ REMOVE_CASH
10:30:38 @ SELECT_BALANCE
10:30:51 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:31:00 @ SELECT_END
10:31:03 @ REMOVE_CARD ( ADAM)

(*###################################*)
(*# BACKY AS STEPHEN ON ATM 2, FIRST SIMULATION*)

10:35:27 @ ENTER_CARD ( STEPHEN)
10:35:30 @ ENTER_PIN_1 ( 5)
10:35:32 @ ENTER_PIN_2 ( 0)
10:35:32 @ ENTER_PIN_3 ( 0)
10:35:32 @ ENTER_PIN_4 ( 0)
```

```
10:35:37 @ SELECT_BALANCE
10:35:53 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEDOWN)
                        (* Invalide *)
10:35:53 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:36:00 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:36:09 @ SELECT_WITHDRAW
10:36:15 @ BKGND_KEYDOWN_ON( 2)
                        (* Invalide *)
10:36:15 @ BKGND_KEYDOWN_ON( 5)
                        (* Invalide *)
10:36:23 @ ENTER_AMOUNT ( 2)
10:36:24 @ ENTER_AMOUNT ( 5)
                        (* Repetition *)
10:36:27 @ SELECT_VALID
10:36:32 @ REMOVE_CASH
10:37:21 @ SELECT_END
10:37:26 @ REMOVE_CARD ( STEPHEN)


(*####################################*)
(*# PETER AS STEPHEN ON ATM 2, SECOND SIMULATION*)

(*ATM NO 2----------*)
10:49:50 @ ENTER_CARD ( STEPHEN)
10:49:53 @ ENTER_PIN_1 ( 5)
10:49:56 @ ENTER_PIN_2 ( 0)
10:49:56 @ ENTER_PIN_3 ( 0)
10:49:57 @ ENTER_PIN_4 ( 0)
10:50:07 @ SELECT_WITHDRAW
10:50:12 @ ENTER_AMOUNT ( 3)
10:50:14 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:50:15 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:20 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:25 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:25 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:27 @ SELECT_VALID
10:50:34 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:36 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:50:39 @ SELECT_WITHDRAW
10:50:46 @ ENTER_AMOUNT ( 3)
10:50:47 @ ENTER_AMOUNT ( 0)
                        (* Repetition *)
10:50:50 @ SELECT_VALID
10:50:53 @ BKGND_BUTTON_NAME ( BOTTOM_BUTTON,MOUSEUP)
                        (* Invalide *)
10:50:58 @ SELECT_CANCEL_CONTINUE
                        (*Annulation *)
10:51:02 @ SELECT_WITHDRAW
```

10:51:04 @ ENTER_AMOUNT (2)
10:51:15 @ ENTER_AMOUNT (5)
(* Repetition *)
10:51:16 @ SELECT_VALID
10:51:21 @ REMOVE_CASH
10:51:24 @ SELECT_BALANCE
10:51:28 @ SELECT_CANCEL_CONTINUE
(*Annulation *)
10:51:33 @ SELECT_END
10:51:39 @ REMOVE_CARD (STEPHEN)
10:52:35 @ OUT

4. Texte de l'évaluation retranscrite par Ian Denley

Retranscription intégrale du texte pour l'évaluation du DAB initial et du DAB modifié. Ces évaluations ont été effectuées à l'Unité d'Ergonomie de l'Université College, à Londres, en août 1992.

4.1. Evaluation du DAB modifié

1) Semantic problem with "enter" + "remove". At least one user looked for an "enter" key i.e. Compatibility problem with view of pressing "enter" key on current ATM's.

2) i) Keypad layout - not compatible with existing conventions (however, no errors in keying by users - though some 'virtual' errors where they checked themselves before making the error
ii)a) button shape - inconsistency between input functions of "0", "valid" + "concel" (see also 3 below), b) layout of keys confusing

3) Transparency + salience problem with "valid" key not being next to dialogue. Common error amongst users.
User often tried all 3 other selection keys before seeing the "valid" key next to the number pad.
[Some errors with "cancel to continue" (also see 7 over)]

4) Semantic problem with "cancel to continue", dialogue should be re-phrased

5) No error message if you should click on another card (i.e. no feedback)

6) When remain cards there should be no buffering on mouse click.
At least one user double clicked + hence removed + then re-entered his card

7) At least one user on failing to see "valid" key - cancelled and started again

8) If making an operation selection, there should also be a dialogue that informs poeple how to get back to last menu if they change their mind.
eg. if select withdrawals + then want to return to account balance.

9) Generally no feedback on errors

10) On having an insufficient amount users are forced back to the beginning to input a new amount. i.e. No feedback or support few orientation (i.e. "Would you like to check your balance?"), and no reversibility (i.e. Input new amount)

11) Larger delay on voice would be better. It still sneaks after the card is removed.

12) Most users tried to use the keybord to input numbers (for their PIN (Personnal Identification Number)).

13) Users would expect "end of operations" to lead to automatic ejection of card.

4.2. Evaluation du DAB initial

14) Keybord layout closer to conventions - but still not entirely compatible.

15) Some users missed the dialogue informing them of their account balance. i.e. not salient - re-design dialogue.

16) No cancel button - should you start to put in a wrong PIN number (you have to remove card and start again)

17) Again no reversibility if you have insufficient funds

18) One user tried to position cursor before entering PIN - (unecessary action - no error message - no feedback)

19) If the PIN number is unfinished other actions should give feedback to use. One user tried other "unamed" buttons when nothing happened

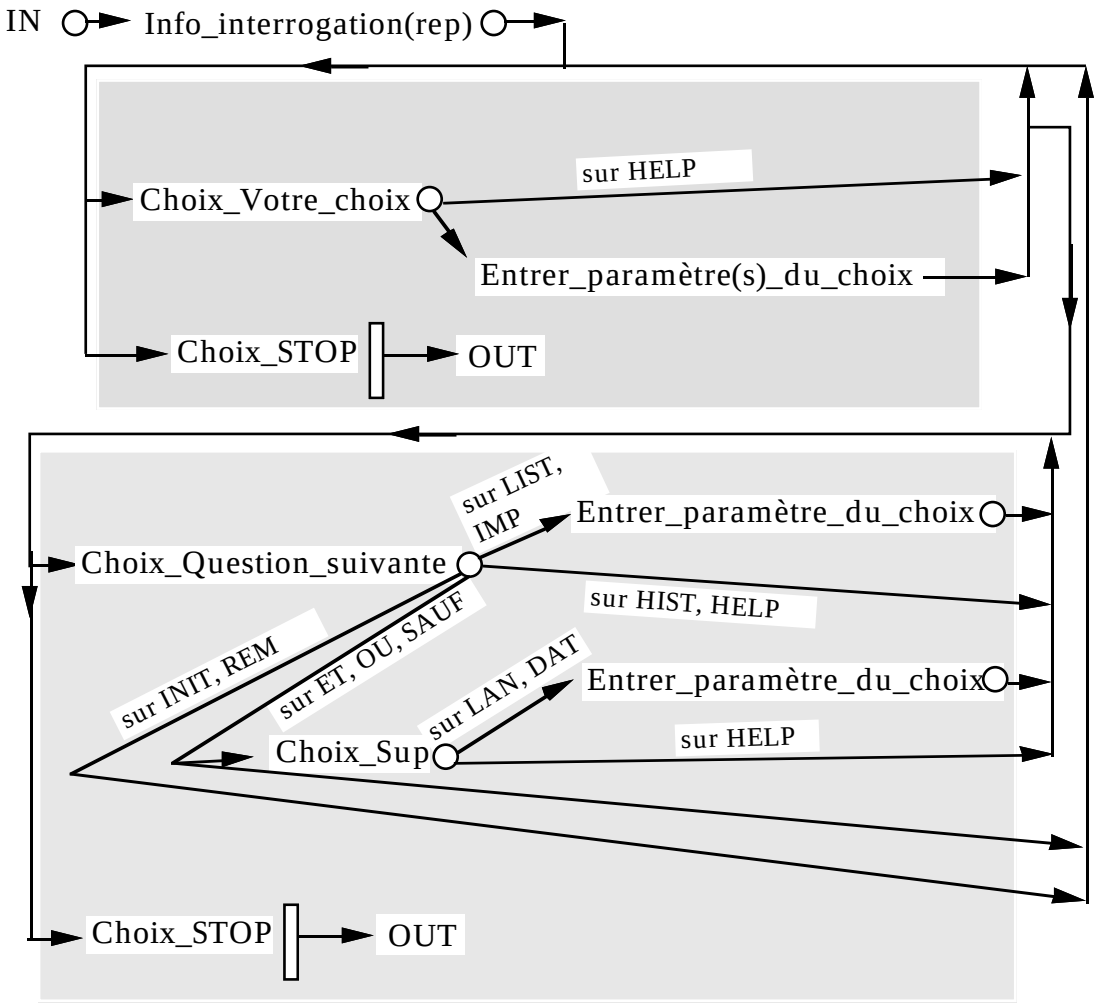
20) Again incorrect PIN needs card removed

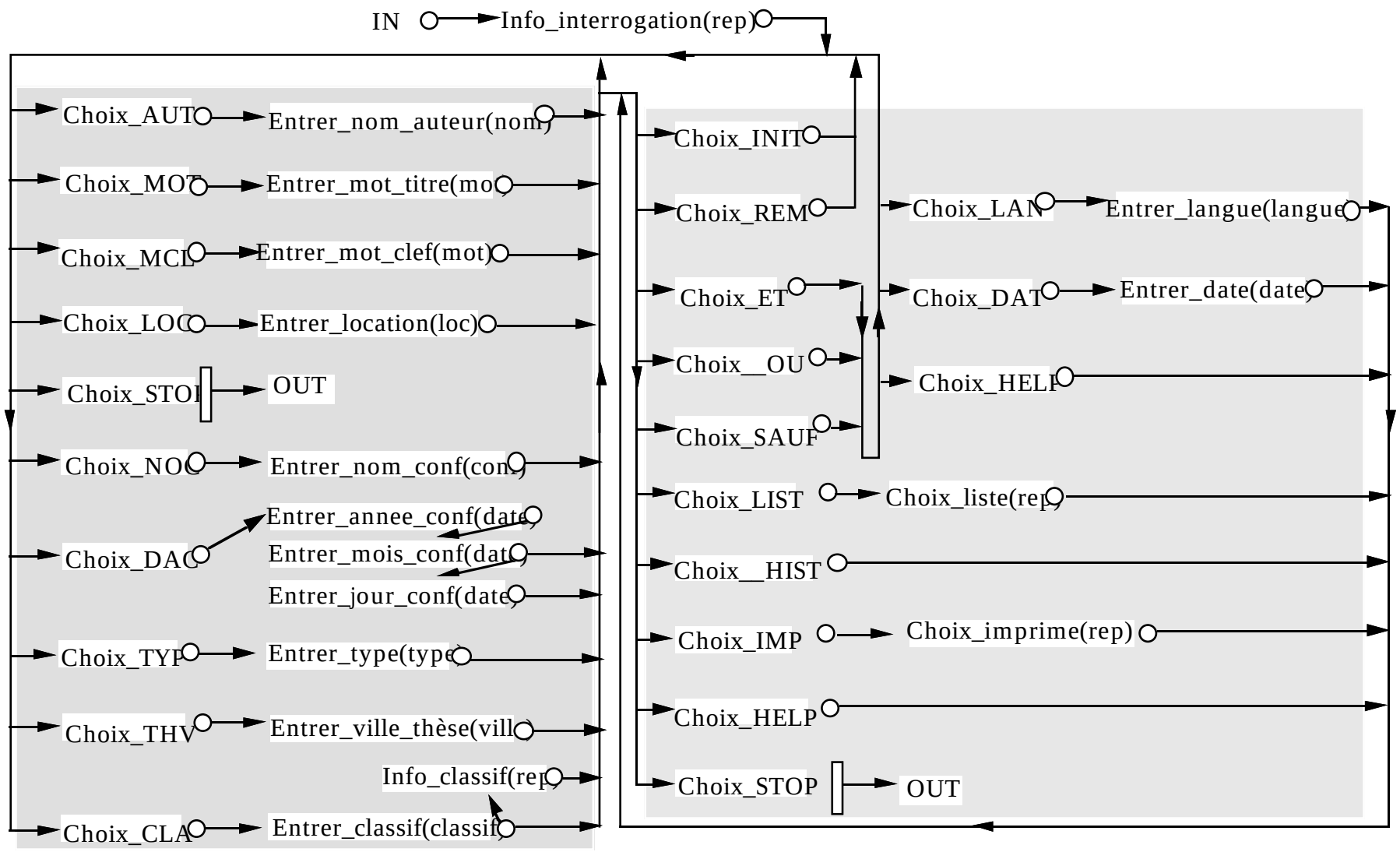
Annexe 4

Plate-forme d'expérimentation : Médiathèque

1. Graphe des séquences : représentation graphique

Ci-dessous, nous reprenons le résumé de l'arbre des séquences tel que nous l'avons présenté au chapitre V, figure 40. Les zones grisée et hachurée identifient, sur l'arbre des séquences détaillé à la page suivante, la correspondance établie avec le résumé représenté ci-dessous.




IN
Info_interrogation(rep)
Choix_AUT
Entrer_nom_auteur(nom)
Choix_MOT
Entrer_mot_titre(mot)
Choix_MCL
Entrer_mot_clef(mot)
Choix_LOC
Entrer_location(loc)
Choix_STOP
OUT
Choix_NOC
Entrer_nom_conf(conf)
Entrer_annee_conf(date)
Choix_DAC
Entrer_mois_conf(date)
Entrer_jour_conf(date)
Choix_TYP
Entrer_type(type)
Choix_THV
Entrer_ville_thèse(ville)
Info_classif(rep)
Choix_CLA
Entrer_classif(classif)
Choix_INIT
Choix_REM
Choix_LAN
Entrer_langue(langue)
Choix_ET
Choix_DAT
Entrer_date(date)
Choix__OU
Choix_HELP
Choix_SAUF
Choix_LIST
Choix_liste(rep)
Choix__HIST
Choix_IMP
Choix_imprime(rep)
Choix_HELP
Choix_STOP
OUT

2. Graphe des séquences annoté

```
IN      -O->
                              INFO_INTERROGATION ( REP ) .

INFO_INTERROGATION        -O->
                             CHOIX_AUT ;
                             CHOIX_MOT ;
                             CHOIX_MCL ;
                             CHOIX_LOC ;
                             CHOIX_STOP ;
                             CHOIX_NOC ;
                             CHOIX_DAC ;
                             CHOIX_TYP ;
                             CHOIX_THV ;
                             CHOIX_CLA .

CHOIX_AUT       -O->
                              ENTRER_NOM_AUTEUR ( NOM ) .

CHOIX_MOT       -O->
                              ENTRER_MOT_TITRE ( MOT )
                          (* Invalide :
                            +17:06:93+11:02:32@ CHOIX_
                            +17:06:93+11:02:44@ CHOIX_
                         *) .

CHOIX_MCL       -O->
                              ENTRER_MOT_CLEF ( MOT ) .

CHOIX_LOC       -O->
                              ENTRER_LOCATION ( LOC ) .

CHOIX_STOP       -O->
                            OUT .

CHOIX_NOC       -O->
                              ENTRER_NOM_CONF ( NOM ) .

CHOIX_DAC       -O->
                              ENTRER_ANNEE_CONF ( DATE ) .

CHOIX_TYP       -O->
                             ENTRER_TYPE ( TYPE ) .

CHOIX_THV       -O->
                              ENTRER_VILLE_THESE ( VILLE ) .

CHOIX_CLA       -O->
                              ENTRER_CLASSIF ( CLASSIF ) .

OUT      -O->
                            IN .

ENTRER_ANNEE_CONF        -O->
                              ENTRER_MOIS_CONF ( DATE ) .
```

```
ENTRER_MOIS_CONF      -O->
                        ENTRER_JOUR_CONF ( DATE ) ;
                       CHOIX_AUT ( DATE ) ;
                       CHOIX_MOT ( DATE ) ;
                       CHOIX_MCL ( DATE ) ;
                       CHOIX_LOC ( DATE ) ;
                       CHOIX_STOP ( DATE ) ;
                       CHOIX_NOC ( DATE ) ;
                       CHOIX_DAC ( DATE ) ;
                       CHOIX_TYP ( DATE ) ;
                       CHOIX_THV ( DATE ) ;
                       CHOIX_CLA ( DATE ) ;
                        CHOIX_INIT ( DATE ) ;
                       CHOIX_ET ( DATE ) ;
                       CHOIX_OU ( DATE ) ;
                        CHOIX_SAUF ( DATE ) ;
                        CHOIX_LIST ( DATE ) ;
                        CHOIX_HIST ( DATE ) ;
                        CHOIX_IMP ( DATE ) ;
                        CHOIX_HELP ( DATE ) ;
                        CHOIX_REM ( DATE ) .

ENTRER_JOUR_CONF      -O->
                        CHOIX_INIT ;
                       CHOIX_ET ;
                       CHOIX_OU ;
                        CHOIX_SAUF ;
                        CHOIX_LIST ;
                        CHOIX_HIST ;
                       CHOIX_IMP ;
                        CHOIX_HELP ;
                       CHOIX_REM ;
                       CHOIX_AUT ;
                       CHOIX_MOT ;
                       CHOIX_MCL ;
                       CHOIX_LOC ;
                        CHOIX_STOP ;
                        CHOIX_NOC ;
                        CHOIX_DAC ;
                        CHOIX_TYP ;
                        CHOIX_THV ;
                        CHOIX_CLA .

ENTRER_NOM_AUTEUR      -O->
                        CHOIX_INIT ;
                       CHOIX_ET ;
                       CHOIX_OU ;
                        CHOIX_SAUF ;
                        CHOIX_LIST ;
                        CHOIX_HIST ;
                       CHOIX_IMP ;
                        CHOIX_HELP ;
                       CHOIX_REM ;
                       CHOIX_AUT ;
                       CHOIX_MOT ;
                       CHOIX_MCL ;
                       CHOIX_LOC ;
```

```
                        CHOIX_STOP;
                        CHOIX_NOC;
                        CHOIX_DAC;
                        CHOIX_TYP;
                        CHOIX_THV;
                        CHOIX_CLA.

ENTRER_MOT_TITRE    -O->
                        CHOIX_INIT;
                       CHOIX_ET
                     (* Invalide :
                       +16:06:93+15:03:59@ CHOIX_ROUTING
                    *);
                        CHOIX_OU;
                        CHOIX_SAUF;
                        CHOIX_LIST;
                        CHOIX_HIST;
                        CHOIX_IMP;
                        CHOIX_HELP;
                       CHOIX_REM;
                       CHOIX_AUT;
                       CHOIX_MOT;
                       CHOIX_MCL;
                       CHOIX_LOC;
                        CHOIX_STOP;
                        CHOIX_NOC;
                        CHOIX_DAC;
                        CHOIX_TYP;
                        CHOIX_THV;
                        CHOIX_CLA.

ENTRER_MOT_CLEF     -O->
                        CHOIX_INIT;
                       CHOIX_ET;
                       CHOIX_OU;
                        CHOIX_SAUF;
                        CHOIX_LIST;
                        CHOIX_HIST;
                        CHOIX_IMP;
                        CHOIX_HELP;
                       CHOIX_REM;
                       CHOIX_AUT;
                       CHOIX_MOT;
                       CHOIX_MCL;
                       CHOIX_LOC;
                        CHOIX_STOP;
                        CHOIX_NOC;
                        CHOIX_DAC;
                        CHOIX_TYP;
                        CHOIX_THV;
                        CHOIX_CLA.

ENTRER_LOCATION     -O->
                        CHOIX_INIT;
                       CHOIX_ET;
                       CHOIX_OU;
                        CHOIX_SAUF;
```

CHOIX_LIST ;
CHOIX_HIST ;
CHOIX_IMP ;
CHOIX_HELP ;
CHOIX_REM ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

ENTRER_NOM_CONF -O->
CHOIX_INIT ;
CHOIX_ET ;
CHOIX_OU ;
CHOIX_SAUF ;
CHOIX_LIST ;
CHOIX_HIST ;
CHOIX_IMP ;
CHOIX_HELP ;
CHOIX_REM ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

ENTRER_TYPE -O->
CHOIX_INIT ;
CHOIX_ET ;
CHOIX_OU ;
CHOIX_SAUF ;
CHOIX_LIST ;
CHOIX_HIST ;
CHOIX_IMP ;
CHOIX_HELP ;
CHOIX_REM ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

```
ENTRER_VILLE_THESE      -O->
                         CHOIX_INIT;
                        CHOIX_ET;
                        CHOIX_OU;
                         CHOIX_SAUF;
                         CHOIX_LIST;
                         CHOIX_HIST;
                        CHOIX_IMP;
                         CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_AUT;
                        CHOIX_MOT;
                        CHOIX_MCL;
                        CHOIX_LOC;
                         CHOIX_STOP;
                         CHOIX_NOC;
                         CHOIX_DAC;
                         CHOIX_TYP;
                         CHOIX_THV;
                         CHOIX_CLA.

ENTRER_CLASSIF      -O->
                          INFO_CLASSIF(REP);
                          CHOIX_INIT(REP);
                         CHOIX_ET(REP);
                         CHOIX_OU(REP);
                          CHOIX_SAUF(REP);
                          CHOIX_LIST(REP);
                          CHOIX_HIST(REP);
                         CHOIX_IMP(REP);
                         CHOIX_HELP(REP);
                         CHOIX_REM(REP);
                         CHOIX_AUT(REP);
                         CHOIX_MOT(REP);
                         CHOIX_MCL(REP);
                         CHOIX_LOC(REP);
                         CHOIX_STOP(REP);
                         CHOIX_NOC(REP);
                         CHOIX_DAC(REP);
                         CHOIX_TYP(REP);
                         CHOIX_THV(REP);
                         CHOIX_CLA(REP).

INFO_CLASSIF        -O->
                         CHOIX_INIT;
                        CHOIX_ET;
                        CHOIX_OU;
                         CHOIX_SAUF;
                         CHOIX_LIST;
                         CHOIX_HIST;
                        CHOIX_IMP;
                         CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_AUT;
                        CHOIX_MOT;
                        CHOIX_MCL;
```

CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

CHOIX_INIT -O->
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

CHOIX_ET -O->
CHOIX_LAN ;
CHOIX_DAT ;
CHOIX_HELP ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

CHOIX_OU -O->
CHOIX_LAN ;
CHOIX_DAT ;
CHOIX_HELP ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;
CHOIX_LOC ;
CHOIX_STOP ;
CHOIX_NOC ;
CHOIX_DAC ;
CHOIX_TYP ;
CHOIX_THV ;
CHOIX_CLA .

CHOIX_SAUF -O->
CHOIX_LAN ;
CHOIX_DAT ;
CHOIX_HELP ;
CHOIX_AUT ;
CHOIX_MOT ;
CHOIX_MCL ;

```
                        CHOIX_LOC ;
                        CHOIX_STOP ;
                        CHOIX_NOC ;
                        CHOIX_DAC ;
                        CHOIX_TYP ;
                        CHOIX_THV ;
                        CHOIX_CLA .

CHOIX_LIST      -O->
                          CHOIX_LISTE ( REP )
                       (* Invalide :
                        +16:06:93+14:53:29@ CHOIX_
                        +16:06:93+14:53:48@ CHOIX_
                        +16:06:93+15:06:03@ CHOIX_
                        +16:06:93+15:06:06@ CHOIX_
                         +17:06:93+10:06:58@ CHOIX_LISTE( M)
                        +17:06:93+10:18:54@ CHOIX_
                         +18:06:93+10:50:38@ CHOIX_LISTE( C)
                        +18:06:93+10:53:08@ CHOIX_
                        +18:06:93+10:53:10@ CHOIX_
                         +18:06:93+15:46:47@ CHOIX_LISTE( M)
                         +18:06:93+15:46:50@ CHOIX_LISTE( M)
                        +18:06:93+15:48:15@ CHOIX_
                        +22:06:93+11:25:18@ CHOIX_
                        +24:06:93+17:02:09@ CHOIX_LI
                        +24:06:93+17:49:28@ CHOIX_
                       *) .

CHOIX_HIST      -O->
                         CHOIX_INIT ;
                        CHOIX_ET ;
                        CHOIX_OU ;
                         CHOIX_SAUF ;
                         CHOIX_LIST ;
                         CHOIX_HIST ;
                        CHOIX_IMP ;
                         CHOIX_HELP ;
                        CHOIX_REM ;
                         CHOIX_STOP .

CHOIX_IMP       -O->
                          CHOIX_IMPRIME ( REP ) .

CHOIX_HELP      -O->
                         CHOIX_INIT ;
                        CHOIX_ET ;
                        CHOIX_OU ;
                         CHOIX_SAUF ;
                         CHOIX_LIST ;
                         CHOIX_HIST ;
                        CHOIX_IMP ;
                         CHOIX_HELP ;
                        CHOIX_REM ;
                         CHOIX_STOP .

CHOIX_REM       -O->
                         CHOIX_INIT ;
```

```
                                CHOIX_ET ;
                                CHOIX_OU ;
                                 CHOIX_SAUF ;
                                 CHOIX_LIST ;
                                 CHOIX_HIST ;
                                CHOIX_IMP ;
                                 CHOIX_HELP ;
                                CHOIX_REM ;
                                CHOIX_AUT ;
                                CHOIX_MOT ;
                                CHOIX_MCL ;
                                CHOIX_LOC ;
                                 CHOIX_STOP ;
                                 CHOIX_NOC ;
                                 CHOIX_DAC ;
                                 CHOIX_TYP ;
                                 CHOIX_THV ;
                                 CHOIX_CLA .

CHOIX_LAN       -O->
                                  ENTRER_LANGUE ( LANGUE ) .

CHOIX_DAT       -O->
                                  ENTRER_DATE ( DATE ) .

CHOIX_HELP      -O->
                                  CHOIX_INIT ;
                                 CHOIX_ET ;
                                 CHOIX_OU ;
                                  CHOIX_SAUF ;
                                  CHOIX_LIST ;
                                  CHOIX_HIST ;
                                 CHOIX_IMP ;
                                  CHOIX_HELP ;
                                 CHOIX_REM ;
                                 CHOIX_AUT ;
                                 CHOIX_MOT ;
                                 CHOIX_MCL ;
                                 CHOIX_LOC ;
                                  CHOIX_STOP ;
                                  CHOIX_NOC ;
                                  CHOIX_DAC ;
                                  CHOIX_TYP ;
                                  CHOIX_THV ;
                                  CHOIX_CLA .

CHOIX_LISTE     -O->
                                   CHOIX_INIT ;
                                  CHOIX_ET
                             (* Invalide :
                               +23:06:93+15:55:46@ CHOIX_CONTROL
                            *) ;
                                  CHOIX_OU ;
                                   CHOIX_SAUF ;
                                   CHOIX_LIST ;
                                   CHOIX_HIST ;
                                  CHOIX_IMP ;
```

```
                        CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_STOP.

CHOIX_IMPRIME     -O->
                        CHOIX_INIT;
                        CHOIX_ET;
                        CHOIX_OU;
                        CHOIX_SAUF;
                        CHOIX_LIST;
                        CHOIX_HIST;
                        CHOIX_IMP;
                        CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_STOP.

ENTRER_LANGUE     -O->
                        CHOIX_INIT;
                        CHOIX_ET;
                        CHOIX_OU;
                        CHOIX_SAUF;
                        CHOIX_LIST;
                        CHOIX_HIST;
                        CHOIX_IMP;
                        CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_STOP.

ENTRER_DATE       -O->
                        CHOIX_INIT;
                        CHOIX_ET;
                        CHOIX_OU;
                        CHOIX_SAUF;
                        CHOIX_LIST;
                        CHOIX_HIST;
                        CHOIX_IMP;
                        CHOIX_HELP;
                        CHOIX_REM;
                        CHOIX_STOP.
```

3. Fichier de capture annoté

```
+16:06:93+14:52:53@ IN
+16:06:93+14:52:55@ INFO_INTERROGATION( N)
+16:06:93+14:52:58@ CHOIX_AUT
+16:06:93+14:53:03@ ENTRER_NOM_AUTEUR( LEGRAND)
+16:06:93+14:53:07@ CHOIX_LIST
+16:06:93+14:53:09@ CHOIX_LISTE( C)
+16:06:93+14:53:29@ CHOIX_
                        (* Invalide *)
+16:06:93+14:53:48@ CHOIX_
                        (* Invalide *)
+16:06:93+14:53:53@ CHOIX_STOP
+16:06:93+14:53:53@ OUT
```

```
+16:06:93+15:00:32@ IN
+16:06:93+15:00:47@ INFO_INTERROGATION( O)
+16:06:93+15:01:56@ CHOIX_MOT
+16:06:93+15:02:28@ ENTRER_MOT_TITRE( SYSTEME_FLEXIBLE_DE_PRODUCTION)
+16:06:93+15:02:38@ CHOIX_MOT
+16:06:93+15:02:49@ ENTRER_MOT_TITRE( PRODUCTION)
+16:06:93+15:03:02@ CHOIX_ET
+16:06:93+15:03:11@ CHOIX_MOT
+16:06:93+15:03:19@ ENTRER_MOT_TITRE( ROUTES)
+16:06:93+15:03:34@ CHOIX_ET
+16:06:93+15:03:37@ CHOIX_MOT
+16:06:93+15:03:43@ ENTRER_MOT_TITRE( ROUTE)
+16:06:93+15:03:52@ CHOIX_ET
+16:06:93+15:03:59@ CHOIX_ROUTING
                        (* Invalide *)
+16:06:93+15:04:05@ CHOIX_MOT
+16:06:93+15:04:11@ ENTRER_MOT_TITRE( ROUTING)
+16:06:93+15:04:21@ CHOIX_ET
+16:06:93+15:04:33@ CHOIX_MOT
+16:06:93+15:04:43@ ENTRER_MOT_TITRE( BATCHE)
+16:06:93+15:04:51@ CHOIX_ET
+16:06:93+15:04:56@ CHOIX_MOT
+16:06:93+15:05:07@ ENTRER_MOT_TITRE( BATCH)
+16:06:93+15:05:13@ CHOIX_LIST
+16:06:93+15:05:16@ CHOIX_LISTE( M)
+16:06:93+15:05:49@ CHOIX_LIST
+16:06:93+15:05:59@ CHOIX_LISTE( M)
+16:06:93+15:06:03@ CHOIX_
                        (* Invalide *)
+16:06:93+15:06:06@ CHOIX_
                        (* Invalide *)
+16:06:93+15:06:13@ CHOIX_LIST
+16:06:93+15:06:23@ CHOIX_LISTE( C)
+16:06:93+15:06:37@ CHOIX_STOP
+16:06:93+15:06:37@ OUT

+17:06:93+10:06:33@ IN
+17:06:93+10:06:36@ INFO_INTERROGATION( N)
+17:06:93+10:06:40@ CHOIX_AUT
+17:06:93+10:06:44@ ENTRER_NOM_AUTEUR( DIDAY)
+17:06:93+10:06:50@ CHOIX_LIST
+17:06:93+10:06:53@ CHOIX_LISTE( M)
+17:06:93+10:06:58@ CHOIX_LISTE( M)
                        (* Invalide *)
+17:06:93+10:18:54@ CHOIX_
                        (* Invalide *)
+17:06:93+10:19:01@ CHOIX_REM
+17:06:93+10:19:12@ CHOIX_MOT
+17:06:93+10:19:23@ ENTRER_MOT_TITRE( ALGORITHM)
+17:06:93+11:02:32@ CHOIX_
                        (* Invalide *)
+17:06:93+11:02:44@ CHOIX_
                        (* Invalide *)
+17:06:93+11:02:57@ CHOIX_STOP
+17:06:93+11:02:57@ OUT

+18:06:93+10:49:58@ IN
```

```
+18:06:93+10:50:03@ INFO_INTERROGATION( N)
+18:06:93+10:50:08@ CHOIX_MOT
+18:06:93+10:50:12@ ENTRER_MOT_TITRE( WAVELET)
+18:06:93+10:50:26@ CHOIX_LIST
+18:06:93+10:50:30@ CHOIX_LISTE( C)
+18:06:93+10:50:38@ CHOIX_LISTE( C)
                        (* Invalide *)
+18:06:93+10:53:08@ CHOIX_
                        (* Invalide *)
+18:06:93+10:53:10@ CHOIX_
                        (* Invalide *)
+18:06:93+10:53:18@ CHOIX_STOP
+18:06:93+10:53:18@ OUT

+18:06:93+15:46:05@ IN
+18:06:93+15:46:10@ INFO_INTERROGATION( O)
+18:06:93+15:46:25@ CHOIX_MOT
+18:06:93+15:46:33@ ENTRER_MOT_TITRE( AGENT)
+18:06:93+15:46:40@ CHOIX_LIST
+18:06:93+15:46:44@ CHOIX_LISTE( M)
+18:06:93+15:46:47@ CHOIX_LISTE( M)
                        (* Invalide *)
+18:06:93+15:46:50@ CHOIX_LISTE( M)
                        (* Invalide *)
+18:06:93+15:48:15@ CHOIX_
                        (* Invalide *)
+18:06:93+15:48:20@ CHOIX_STOP
+18:06:93+15:48:20@ OUT

+22:06:93+11:24:24@ IN
+22:06:93+11:24:26@ INFO_INTERROGATION( N)
+22:06:93+11:24:29@ CHOIX_AUT
+22:06:93+11:24:31@ ENTRER_NOM_AUTEUR( GERO)
+22:06:93+11:24:36@ CHOIX_LIST
+22:06:93+11:24:39@ CHOIX_LISTE( C)
+22:06:93+11:25:18@ CHOIX_
                        (* Invalide *)
+22:06:93+11:25:23@ CHOIX_INIT
+22:06:93+11:25:27@ CHOIX_STOP
+22:06:93+11:25:27@ OUT

+23:06:93+15:51:18@ IN
+23:06:93+15:51:22@ INFO_INTERROGATION( O)
+23:06:93+15:51:42@ CHOIX_AUT
+23:06:93+15:51:59@ ENTRER_NOM_AUTEUR( BLAZEWICZ)
+23:06:93+15:52:13@ CHOIX_MOT
+23:06:93+15:52:46@ ENTRER_MOT_TITRE (
SCHEDULING_IN_COMPUTER_AND_MANUFACTURING_SYSTEMS)
+23:06:93+15:52:57@ CHOIX_MOT
+23:06:93+15:53:03@ ENTRER_MOT_TITRE( SCHEDULING)
+23:06:93+15:53:08@ CHOIX_ET
+23:06:93+15:53:13@ CHOIX_MOT
+23:06:93+15:53:22@ ENTRER_MOT_TITRE( CUMPUTER)
+23:06:93+15:53:40@ CHOIX_INIT
+23:06:93+15:53:50@ CHOIX_AUT
+23:06:93+15:54:01@ ENTRER_NOM_AUTEUR( BENSOUSSAN)
+23:06:93+15:54:09@ CHOIX_ET
```

```
+23:06:93+15:54:14@ CHOIX_MOT
+23:06:93+15:54:21@ ENTRER_MOT_TITRE( MATHEMATICAL)
+23:06:93+15:54:30@ CHOIX_LIST
+23:06:93+15:54:32@ CHOIX_LISTE( C)
+23:06:93+15:55:38@ CHOIX_ET
+23:06:93+15:55:46@ CHOIX_CONTROL
                        (* Invalide *)
+23:06:93+15:55:54@ CHOIX_MOT
+23:06:93+15:56:00@ ENTRER_MOT_TITRE( CONTROL)
+23:06:93+15:56:18@ CHOIX_LIST
+23:06:93+15:56:22@ CHOIX_LISTE( C)
+23:06:93+15:57:12@ CHOIX_IMP
+23:06:93+15:57:16@ CHOIX_IMPRIME( C)
+23:06:93+15:57:26@ CHOIX_STOP
+23:06:93+15:57:26@ OUT

+24:06:93+16:53:36@ IN
+24:06:93+16:53:39@ INFO_INTERROGATION( N)
+24:06:93+16:53:45@ CHOIX_MOT
+24:06:93+16:53:50@ ENTRER_MOT_TITRE( ROBUST)
+24:06:93+16:53:57@ CHOIX_LIST
+24:06:93+16:53:58@ CHOIX_LISTE( M)
+24:06:93+16:54:30@ CHOIX_ET
+24:06:93+16:54:32@ CHOIX_AUT
+24:06:93+16:54:35@ ENTRER_NOM_AUTEUR( HUBER)
+24:06:93+16:54:43@ CHOIX_LIST
+24:06:93+16:54:45@ CHOIX_LISTE( C)
+24:06:93+17:02:09@ CHOIX_LI
                        (* Invalide *)
+24:06:93+17:02:12@ CHOIX_LIST
+24:06:93+17:02:15@ CHOIX_LISTE( C)
+24:06:93+17:49:28@ CHOIX_
                        (* Invalide *)
+24:06:93+17:49:31@ CHOIX_STOP
+24:06:93+17:49:31@ OUT
```

Références bibliographiques

[Abowd 92]
G.D. Abowd, J. Coutaz & L. Nigay, “Structuring the space of interactive system properties”, in the proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, Ellivuori, Finland, Elsevier Publisher, August 1992

[Arch 92]
“The UIMS Workshop Tool Developers : A Metamodel for the Runtime Architecture of an Interactive System”, SIGCHI Bulletin, 24, 1, pp. 32-37, January 1992

[Barnard 87]
P.J. Barnard, “Cognitive Resources and the Learning of Human-Computer Dialogs”, in Interfacing Thought, edited by J.M. Carroll, The MIT Press, pp.112-158, 1987

[Barnard 92]
P. Barnard, J. May, L. Tweedie & A. Blanford, “ICS Presentation”, Document MacroMind Player (Macintosh), 1992

[Barthet 88]
M.-F. Barthet, “Logiciels interactifs et ergonomie, modèles et méthodes de conception”, Editions Dunod Informatique, 1988

[Bass 92]
L. Bass & J. Coutaz, “Developing software for the User Interface”, Addison-Wesley publishing company, 1992

[Bastien 91]
C. Bastien, “Validation de critères ergonomiques pour l'évaluation d'interfaces utilisateurs”, INRIA, Unité de recherche INRIA_Rocquencourt, rapport de recherche n° 1427, Programme 3, Mai 1991

[Bastien 93]
J.M. Bastien & D.L. Scapin, “Preliminart findings on the effectiveness of ergonomic criteria for the evaluation of human-computer interfaces”, in Proceedings of the InterCHI'93 conference, Amsterdam, April 1993

[Bernsen 93]
N.O. Bernsen, “Structuring the design space”, in the InterCHI'93 Adjunct Proceedings, pp. 211-212, Amsterdam, 1993

[Boehm 81]
B.W. Boehm, “Sotfware Engineering Economics”, Prentice-Hall, 1981

[Boehm 88]
B.W. Boehm, “A spiral Model of Software Development and Enhancement”, IEEE Computer, May 1988

[Boy 92]
G. Boy, “Méthodologies et Outils pour l'Interaction Cognitive Homme-Machine”, Habilitation à diriger des recherches, Université Pierre et Marie Curie, Paris, 29 Juin 1992

[Caelen 92]
J. Caelen & A. L. Fréchet, "Attitudes cognitives et actes de langage en situation de communication homme/machine", article soumis au congrès de l'ARC, Nancy, 1992

[Card 83]
S.K. Card, T.P. Moran & A. Newell, “The psychology of Human Computer Interaction”, Lawrence Erlbaum Associates, 1983

[Carroll 84]
J.M. Carroll & C. Carrithers, “Training Wheels in the User Interface”, Communication of the ACM, 27(8), pp. 800-807, August, 1984

[Chabert 91]
A. Chabert, “La programmation visuelle”, rapport de DEA d'informatique, Institut National Polytechnique de Grenoble, 25 Juin 1991

[Coad 91]
P. Coad & E. Yourdon, “Object-oriented analysis, 2d edition”, Prentice Hall, Yourdon Press computing series, 1991

[Colbert 92]
M. Colbert, J. Dowell & J. Long, “The application of Dowell & Long's engineering conception of Human-Computer Interaction to Military Command & Control : a view of the domain of C2 and its fitness for purpose”, Ergonomics Unit, University College London, UK, 13th March, 1992

[Collet 93]
B. Collet, “Manuel utilisateur Compo, version 1.3”, LGI-IMAG, BP 53 X, 38041 Grenoble Cedex, Juillet 1993

[Collins 69]
A.M. Collins & M.R. Quillian, “Retrieval Time from Semantic Memory”, Journal of Verbal Learning and Verbal Behavior, 8, pp. 240-247, 1969

[Coutaz 90]
Joëlle Coutaz, “Interfaces homme-ordinateur : Conception et réalisation”, Dunod Informatique, 1990

[Coutaz 93a]
J. Coutaz, G. Faconti, F. Paterno, L. Nigay & D. Salber, “MATIS: a UAN Description and Lesson Learned”, Amodeus Report, SM/WP14, 1993

[Coutaz 93b]
J. Coutaz, L. Nigay & D. Salber, “The MSM framework: A Design Space for Multi-Sensori-Motor Systems”, In Human Computer Interaction, 3rd International Conference EWHCI’93, East/West Human Computer Interaction, Moscow. L. Bass, J. Gornostaaev, C. Unger Eds. Springer Verlag Publ., Lecture notes in Computer Science, Vol. 753, pp.231-241, 1993

[Crellin 90]
J. Crellin, “PROTEUS : an approach to interface evaluation”, in proceedings of INTERACT'90, D. Diaper & al. (Ed.), 1990

[Crowley 94]
J. Crowley & J. Bedrune, “Integration and Control of Reactive Visual Processes”, 1994 European Conference on Computer Vision, (ECCV-'94), Stockholm, mai 1994

[Cuzin 92]
R. Cuzin, “Visualisation d'objets multimédia. Application aux dossiers médicaux”, mémoire présenté en vue d'obtenir le diplôme d'ingénieur CNAM, Note technique NOT 014, LGI-IMAG, BP 53, 38041 Grenbole Cedex 9, Octobre 92

[Deux 91]
O. Deux et al., “The O_2 System”, CACM, Vol. 34, n°10, Octobre 1991

[Deux 92]
“The O_2 user manual, Version 4.1, Released October 1992”, O_2 Technology, 7 rue du Parc de Clagny, 78035 Versailles Cedex, France, 1992

[Dix 91]
A.J. Dix, “Formal Methods for Interactive Systems”, Computers and Poeple Series, Academic Press, 1991

[Dowell 89]
J. Dowell & J. Long, “Toward a conception for an engineering discipline of human factors”, Ergonomics, vol. 32, No 11, pp. 1513-1535, November 1989

[Gould 87a]
J.D. Gould, S.J. Boies, S. Levy, J.T. Richards & J. Schoonard, “The 1984 Olympic message system : a test of behavioral principles of system design”, Communications of the ACM, Vol. 30, Number 9, September 1987

[Gould 87b]
J.D. Gould, "How to design usable systems", In Proceedings of INTERACT'87, H.J.Bullinger & B. Shackel (Eds), Elsevier Sciences Publishers, 1987

[Gray 93]
W.D. Gray, B.E. John & M. E. Atwood, “Project Ernestine: Validating a GOMS Analysis for Predicting and Explaining Real-World Task Performance”, Human Computer Interaction, Vol. 8, No 3, pp. 237-309, 1993

[Gray 94]
P. Gray, D. England & S. McGowan, “XUAN: Enhancing the UAN to capture temporal relations among actions”, Computing Science Research Report, IS-94-02, University of Glasgow, E-mail: reports@dcs.glasgow.ac.uk, Feb., 1994

[Hammontree 92]
M.L. Hammontree, J.J. Hendrickson & B.W. Hensley, “Integrated data capture and analysis tools for research and testing on graphical user interfaces”, in Proceedings of the CHI'92 Conference, pp.431-432, ACM Press, Monterey, 3-7 May 1992

[Hammouche 93]
H. Hammouche, De la modé;lisation des tâches à la spécification d’interfaces utilisateur, Rapport INRIA no 1959, juillet 1993.

[Harrison 87]
M.D. Harrison & A.J. Dix, “Formalising Models of Interaction in the Design of a Display Editor”, in Proceedings of the Second IFIP Conference on Human-Computer Interaction-INTERACT’87, pp.409-414, 1987

[Hartson 92]
H.R. Hartson & P.D. Gray, “Temporal aspects of tasks in the User Action Notation”, Human-computer interaction, Vol. 7, pp. 1-45, 1992

[Harvey 88]
G. Harvey, “Understanding HyperCard, for version 1.1”, Sybex Inc., 1988

[Hayes-Roth 79]
B. Hayes-Roth & F. Hayes-Roth, “A Cognitive Model of Planning”, Cognitive Science 3, pp. 275-310, 1979

[Hix 93]
D. Hix & R. Hartson, “Developing User Interfaces, Ensuring Usability through Product & Process”, Wiley, 381 pages, 1993

[Hoare 85]
C.A.R. Hoare, “Communicating Sequential Processes”, Prentice-Hall, 1985

[Hoshstrasser 89]
B.H. Hoshstrasser & N.D. Geddes, “OPAL, Operator Intent Inferencing for Intelligent Operator Support Systems”, Technical Report, Search Technology, Inc., 4725 Peachtree Corners Circle, Norcross, GA 30092, 1989

[Howes 90]
A. Howes & S.J. Payne, “Display based competence: Towards user models for menu driven interfaces”, International Journal of Man-Machine Studies, 33 (6), pp. 637-655, 1990

[Howes 91]
A. Howes & R.M. Young, “Predicting the learnability of task-Action Mappings”, CHI’91 proceedings, pp. 113-118, 1991

[Jacob 90]
R.J.K. Jacob, “What you look is what you get : eye mouvement-based interaction techniques”, in Proceedings of CHI’90 conference, ACM Press, April 1990, pp.11-18

[Jeffries 91]
R. Jeffries, J.R. Miller, C. Wharton & K.M. Uyeda, “User interface evaluation in the real word : a comparison of four techniques”, Proceedings of the CHI’91 Conference on Computer Human Interaction, New Orleans, ACM New York, pp. 119-124, 1991 May

[Johnson 91]
H. Johnson & P. Johnson, “Task Knowledge Structures : Psychological basis and integration into system design”, Acta Psychologica, pp. 3-26, 1991

[Johnson 92]
P. Johnson, H. Johnson & P. Markopoulos, “Task based design: mapping between user task models and user interaction dialogues”, in Proceeding’s of the 11th Interdisciplinary Workshop on Informatics and Psychology: Task Analysis in Human-Computer Interaction, Schärding, June 9-11, 1992

[Johnson 93]
P. Johnson, S. Wilson, P. Markopoulos & J. Pycock, “ADEPT-Advanced Design Environment for Prototyping with Task Models”, InterCHI’93 proceedings, pp. 56, 1993

[Kieras 85]
D. Kieras & P.G. Polson, “An Approach to the Formal Analysis of User Complexity”, International Journal of Man-Machine Studies, 22, pp. 365-394, 1985

[Kolski 89]
C. Kolski, “Contribution a l'ergonomie de conception des interfaces graphiques homme-machine dans les procédés industriels : application au système expert SYNOP”, Thèse présentée à l'Université de Valenciennes et du Hainaut-Cambrésis, Janvier 1989

[Krakoviak 90]
S. Krakoviak, M. Meysembourg, H. Nguyen Van, M. Riveill & C. Roisin, “Design and implementation of an object-oriented, strongly typed language for distributed applications”, Journal of Object-Oriented Programming, Sept. 1990

[Le Petit Robert]
P. Robert, “Le Petit Robert 1, dictionnaire alphabétique et analogique de la langue française”, Edition Les dictionnaires Robert-Canada S.C.C., 1992

[Lerch 89]
F. Lerch, M. Mantei & J.R. Olson, “Skilled financial planning : The Cost of Translating Ideas into Actions”, in Proceedings of CHI’89, ACM Press, pp 121-126, 1989

[Lewis 90]
C. Lewis, P. Polson, C. Wharton & J. Rieman, “Testing a Walkthrough Methodology for Theory-Based Design of Walk-Up-and-Use Interfaces”, Proceedings of the CHI’91 Conference on Computer Human Interaction, Seattle, ACM New York, pp.235-242, April 1990

[Lim 92]
K.Y. Lim, J.B. Long & N. Silcock, “Instanciation of task analysis in a structured method for user interface design”, in Proceeding’s of the 11th Interdisciplinary Workshop on Informatics and Psychology: Task Analysis in Human-Computer Interaction, Schärding, June 9-11, 1992

[Lim 93]
K.Y. Lim & J.B. Long, “Structured notations for human factors specification”, in Proceeding’s of the Ergonomics Society's 1993 Annual Conference, Edinburgh, Scotland, Edited by E.J. Lovesey, 13-16 April 1993

[Logotel 87]
“Logotel : Manuel d'utilisation”, CHEMDATA SA, Tour Saône-Croix-Rousse, 17 quai Joseph Gillet, 69316 LYON Cedex 04, 1er décembre 1987

[Long 89]
J. Long & J. Dowell, “Conceptions of the Discipline of HCI : Craft, Applied Science, and Engineering”, in Proceedings of the Fifth Conference of the BCS HCI SIG, A. Sutcliffe and L. Macaulay (Eds), Cambridge University Press, 1989

[Long 90]
J. Long & I. Denley, “Evaluation for practice”, tutorial, Ergonomics Society 1990 Annual Conference, 1990

[Löwgren 90]
J. Löwgren & T. Nordqvist, “A Knowledge-Based Tool for User Interface Evaluation and its Integration in a UIMS”, Human-Computer Interaction-INTERACT'90, pp. 395-400

[McCall 77]
J. McCall, “Factors in Software Quality”, General electric Eds, 1977

[McDermid 84]
J. McDermid & K. Ripkin, “Life Cycle Support in the ADA environment”, Cambridge University Press, 1984

[McLean 91]
A. McLean, R. Young, V. Bellotti & T.P. Moran, “Questions, Options, and Criteria : Elements of Design Space Analysis”, Human-Computer Interaction, Vol. 6, pp. 201-250, 1991

[MacLeod 93]
M. Macleod, “DRUM, Diagnostic Recorder for Usability Measurement”, NPL, DITC HCI group, Teddington, Middlesex, TW11 OLW, UK, 1993

[Miller 91]
J.R. Miller, R. Jeffries, C. Wharton & K.M. Uyeda, “User interface evaluation in the real word : a comparison of four techniques”, Proceedings of the CHI’91 Conference on Computer Human Interaction. New Orleans. ACM New York, pp. 119-124, 1991

[Monk 87]
A.F. Monk & A. Dix, “Refining early design decisions with a black box model”, in Poeple and Computer III, D. Diaper et R. Winder (Eds), Cambridge University Press, pp.147-158, 1987

[Moran 81]
T.P. Moran, “The Command Language Grammar: a representation for the user interface of interactive computer systems”, in Internal Journal of Man-Machine studies, no 15, pp. 3-50, 1981

[Moran 83]
T.P. Moran, “Getting Into a System : External-Internal Task Mapping Analysis”, CHI’83 Conference Proceedings, December 12-15, Boston, Edited by Ann Janda, 1983

[Mosier 86]
J.N. Mosier & S.L. Smith, “Application of guidelines for designing user interface software”, Behaviour and Information Technology, Vol. 5, no. 1, pp.39-46, 1986

[Motif 90]
“OSF/Motif Programmer’s Guide”, Open Software Foundation, Prentice Hall, 1990

[Nanard 90]
J. Nanard, “La manipulation directe en interface homme-machine”, thèse présentée à l'université des Sciences et Techniques du Languedoc, Décembre 1990

[Neal 83]
A.S. Neal & R.M. Simons, “Playback : a method for evaluating the usability of software and its documentation”, Proceedings of the CHI’83 Conference on Computer Human Interaction , ACM New York, pp. 78-82, December 1983

[Newell 72]
A. Newell & H.A. Simon, “Human Problem Solving”, Englewood Cliffs, New Jersey, Prentice-Hall, 1972

[Nielsen 89]
J. Nielsen, “Usability Engineering at a Discount”, in Designing and Using Human Computer Interfaces and Knowledge-Based Systems, Salvendy & Smith Eds., Elsevier North Holland, pp. 394-401, 1989

[Nielsen 90a]
J. Nielsen & R. Molich, “Heuristic evaluation of user interfaces”, Proceedings of the CHI’90 Conference on Computer Human Interaction, Seatle, ACM New York, pp.349-256, 1990

[Nielsen 90b]
J. Nielsen & R. Molich, “Improving a human-computer dialogue : What designers know about traditionnal interface design”, Communication of the ACM 33-3, March 1990

[Nielsen 94]
J. Nielsen & J. Levy, “Measuring Usability: Preference vs. Perfomance”, Communications of the ACM, 37(4), pp.66-75, April, 1994

[Nigay 94]
L. Nigay , “Conception et modélisation logicielles des systèmes interactifs : application aux interfaces multimodales”, thèse de l’Université Joseph Fourier, Grenoble, Janvier 1994

[Norman 86]
D.A. Norman, “Cognitive Engineering”, in User Centered System Design, D.A. Norman & S.W. Draper, Lawrence Erlbaum Associates, pp.31-61, 1986

[Normand 92a]
V. Normand, “Le modèle SIROCO : de la spécification conceptuelle des interfaces utilisateur à leur réalisation”, thèse de l’Université Joseph Fourier, Grenoble, 1992

[Normand 92b]
V. Normand, “Task modelling in HCI : purposes and means”, Rapport de Recherche n° PTI/92-02, Thomson CSF, Division systèmes défense et contrôle, 7 rue des Mathurins, BP 10, 92223 Bagneux Cedex, Juillet 1992

[Palanque 92]
P. Palanque, “Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur”, thèse de doctorat de l'université Toulouse I, spécialité Informatique, 25 Septembre 1992

[Payne 86]
S.J. Payne & T.R.G. Green, “Task-Action Grammars: a model of the mental representation of task languages”, in Human-Computer Interaction, Vol. 2, No. 2, pp. 93-133, Lawrence Erlbaum Associates Publishers, 1986

[Pfaff 85]
G. E. Pfaff (Eds.), “User Interface Management Systems”, Eurographics Seminars, Springer-Verlag, 1985

[Pierrel 87]
J-M. Pierrel, “Dialogue Oral Homme-Machine”, Hermes, 1987

[Pierret-Goldreich 89]
C. Pierret-Goldreich, I. Delouis & D.L. Scapin, “Un Outil d'Acquisition et de Représentation des Taches Orienté-Objet”, rapport de recherche INRIA no 1063, Programme 8 : Communication Homme-Machine, août 1989

[Pirsig 78]
R.M. Pirsig, “Traité du zen et de l'entretien des motocyclettes”, Editions du Seuil, 1978

[Pleczon 92]
P. Pleczon, “Eléments de méthodologie et outils pour l'assistance à l'opérateur: application à la conduite automobile”, thèse de doctorat de l'Ecole Nationale Supérieur de l'Aéronautique et de l'Espace, spécialité informatique, 1992

[Pollier 91]
A. Pollier, “Evaluation d’une interface par des ergonomes : diagnostics et stratégies”, Rapport de recherche INRIA no 1391, Février 1991

[Pressman 87]
R.S. Pressman, “Software engineering, a practioner's approach”, McGraw-Hill Book Company, 1987

[Rasmussen 86]
J. Rasmussen, “Information processing and human-machine interaction : an approch to cognitive engineering”, North-Holland series in system science and engineering, A.P. Sage (Ed.), 1986

[Rauterberg 92]
M. Rauterberg, “Cognitive complexity in man computer interaction : an empirical method of a quantitative measurement”, in Proceeding’s of the 11th Interdisciplinary Workshop on Informatics and Psychology: Task Analysis in Human-Computer Interaction, Schärding, June 9-11, 1992

[Ravden 89]
S. Ravden & G. Johnson, “Evaluating usability of human-computer interfaces : a practical method”, Ellis Horwood books in information technology, 1989

[Reason 90]
J. Reason, “Human Error”, Cambridge University Press, 1990

[Reisner 81]
P. Reisner, “Formal Grammar and Design of an Interactive System”, IEEE Transaction on Software Engineering, SE-5, pp. 229-240, 1981

[Rettig 93]
M. Rettig & G. Simons, “A project planning and development process for small teams”, Communication of the ACM, Vol. 36, No. 10, October 1993

[Richard 90]
J.F. Richard, “Les activités mentales. Comprendre, raisonner, trouver des solutions”, Armand Colin, 1990

[Royce 70]
W.W. Royce, “Managing the development of large software systems”, Proc. WESTCON, Calif., USA, 1970

[Salber 93]
D. Salber & J. Coutaz, “Applying the Wizard of Oz Technique to the Study of Multimodal Systems”, East-West HCI conference, Moscow, August 1993

[Scapin 90]
D.L. Scapin, “Des critères ergonomiques pour l'évaluation et la conception d'interfaces utilisateurs”, Acte du XXVI Congrès de la SELF, 3-5 Octobre, Montréal, Canada, 1990

[Schmucker 86]
K. Schmucker, “MacApp: an application framework”, Byte 11(8), pp.189-193, 1986

[Schoman 77]
K. Schoman & D.T. Ross, “Structured Analysis for requirements definition”, IEEE Trans. Software Eng., SE-3 (1), pp. 6-15, 1977

[Sébillotte 91]
S. Sébillotte, “Décrire des tâches selon les objectifs des opérateurs. De l'interview à la formalisation”, revue Travail humain, tome 54, no 3, pp.193-223, 1991

[Sellen 92]
A.J. Sellen, G.P. Kurtenbach & W.A.S. Buxton, “The Prevention of Mode Errors through Sensory Feedback”, Human Computer Interaction, Vol. 7, Number 2, pp. 141-164, 1992

[Senach 90]
B. Senach, “Evaluation ergonomique des interfaces homme-machine : une revue de la littérature”, Rapport de recherche INRIA n° 1180, Programme 8, Communication Homme-Machine, Mars 1990

[Shneiderman 82]
B. Shneiderman, “Multi-party grammars and related features for designing interactive systems”, IEEE Transactions on Systems, Man, and Cybernetics, 12, pp.148-154, 1982

[Siochi 91]
A.C. Siochi & D. Hix, “A study of computer-supported user interface evaluation using maximal repeating pattern analysis”, Proceedings of the CHI’91 Conference on Computer Human Interaction, New Orleans, ACM New York, pp. 301-305, May 1991

[Smith 86]
S.L. Smith & J.N. Mosier, “A design evaluation checklist for user-system interface software”, Report #MTR-9480 EDS_TR_84-358, The MITRE Corporation, Bedford, MA, 1982

[Sommerville 89]
I. Sommerville, “Software engineering, third edition”, Addison-Wesley Publishing Company, 1989

[Sukaviriya 93]
P.N. Sukaviriya & J.D. Foley, “Supporting adaptative interfaces in a knowledge-based user interface environment”, in Proceedings of the Intelligent User Interfaces'93 conference, 1993

[Sutcliffe 94]
A. Sutcliffe & P. Faraday, “Designing Presentation in Multimedia Interfaces”, CHI’94 proceedings, Boston, pp. 92-98, 1994

[Tarby 93]
J-C. Tarby, “Gestion automatique du dialogue homme-machine à partir de spécifications Conceptuelles”, thèse de l’Université Toulouse I, spécialité informatique, 1993

[Tardieu 83]
H. Tardieu, A Rochfeld & R. Coletti, “La méthode Merise”, Edition des Organisations, Paris 1983

[Tauber 90]
M.J. Tauber, “ETAG: Extended Task-Action Grammar - A language for the description of the user’s task language”, in Proceedings of INTERACT’90, D. Diaper et al. (Editors), Elsevier Science Publishers, North-Holland, 1990

[Terzopoulos 90]
D. Terzopoulos & K. Waters, “Analysis of facial images using physical and anatomical models”, in IEEE, 1990, pp. 727-732

[Theaker 89]
C.J. Theaker, R. Phillips, T.M.E. Frost & W.R. Love, “HIMS : a tool for HCI evaluations”, HCI'89, People and Computers V, A. Sutcliffe & L. Macauley (Eds), Cambridge University Press, 5-8 Sept. 1989

[Thovtrup 91]
H. Thovtrup & J. Nielsen, “Assassing the usability of a user interface standard”, Proceedings of the CHI’91 Conference on Computer Human Interaction, New Orleans, ACM New York, pp. 335-341, May 1991

[ToolboxMac 92]
“Inside Macintosh, Macintosh Toolbox Essentials”, Apple Technical Library, Addison-Wesley Publishing Company, October 92

[Valentin 93]
A. Valentin, G. Vallery & R. Lucongsang, “L’évaluation ergonomique des logiciels, une démarche itérative de conception”, ANACT, Colection Outils et Méthode, 151 pages, 1993

[Vanderdonckt 90]
J. Vanderdonckt, “Modèle de la présentation d’une application : les règles ergonomiques”, rapport IHM/Ergo/3, Facultés Universiatires Notre-Dame de la Paix, Institut d’Informatique, 21 rue Grandgagnage, B-5000 Namur, 27 Août 1990

[Wanner 89]
J.C. Wanner, “L'homme et la sécurité dans les systèmes pilotés”, La Recherche, Juillet-Août 1989

[Watt 87]
D. Watt, B. Wichmann & W. Finlay, “Ada : Language and Methodology”, Prentice Hall, 1987

[Webster 89]
B.F. Webster, “The NeXT Book”, Addison-Wesley, Reading, Mass., 1989

[Whitefield 91]
A. Whitefield, F. Wilson & J. Dowell, “A framework for human factors evaluation”, Behaviour and information technology, vol. 10, no. 1, pp.65-79, 1991

[Whiteside 88]
J. Whiteside, J. Bennett & K. Holtzblatt, “Usability engineering : our experience and evolution”, in Handbook of Human-Computer Intercation, M. Helander (Ed.), Elsevier Science Publishers, 1988

[Wilson 91]
M. Wilson, “The first MMI2 Demonstrator: A Multimodal Interface for Man Machine Interaction with Knowledge Based System”, Deliverable D7, ESPRIT project 2474 MMI2, Tech. report Rutherford Appleton Laboratory, Chilton Didcot Oxon OX11 0QX, RAL-91-093, 1991

[Young 90]
R.M. Young & J. Whittington, “A knowledge Analysis of interactivity”, Proceedings of INTERACT'90, edited by D. Diaper, G. Cockton & B. Shackel, Elsevier Scientific Publishers B.V, pp.207-212, Cambridge (United Kingdom), 27-31st August 1990

[Yuille 90]
A.L. Yuille, “Deformable templates for face recognition”, in Journal of Cognitive Neuroscience, 3(1), 1991, pp.59-70

Table des matières

Merci.....3

Plan de la thèse.....5

INTRODUCTION.....7

1. Le sujet.....7
2. Objectifs.....9
3. Organisation du mémoire.....9

Chapitre I : Génie logiciel et évaluations ergonomiques des interfaces utilisateur.....11
1. Evaluation et génie logiciel.....13
    1.1. Le modèle en cascade.....13
    1.2. Le modèle en V.....16
    1.3. Le modèle en spirale.....19
    1.4. Evaluation et qualité en génie logiciel.....21
2. Evaluation et ergonomie cognitive.....24
    2.1. Démarches de développement.....24
        Approche artisanale.....25
        Approche science appliquée.....26
        Approche ingénierie.....26
    2.2. Qualité en ergonomie.....28
        2.2.1. Conception centrée sur l'utilisateur.....28
        2.2.2. Mesure de la qualité ergonomique.....30
3. Synthèse.....32
    De quoi s'agit-il?.....34
    Qui effectue l'évaluation?.....34
    Où s'effectue l'évaluation?.....34
    Quand s'effectue l'évaluation?.....35
    Comment procède-t-on pour évaluer les interfaces ?.....35

Pourquoi évalue-t-on les interfaces utilisateur?..... 35

PARTIE 1 : LE DOMAINE DE L'ÉTUDE 37

Chapitre II : Etude de la tâche 39
1. Définitions 41
    1.1. Tâche 41
    1.2. Granularité d'une tâche élémentaire et action 44
    1.3. Tâche prévue et tâche effective 45
    1.4. Modèle de tâche 46
2. Axes d'observation : finalité et formalisme 47
3. Modèles de tâche et finalité 47
    3.1. Modèle de tâche comme composant du système final 47
    3.2. Modèles de tâche comme outil de conception 48
        3.2.1. Modèles de tâche et définition des besoins : DIANE, JSD*, MAD, CLG 48
        3.2.2. Modèles de tâche et spécifications externes : ETAG, UAN 56
        3.2.3. Modèles de tâche et évaluation : GOMS, CCT, ETIT, spécifications externes 61
        3.2.4. Un pont entre les différentes phases du cycle de vie d'un logiciel : SIROCO, ADEPT 64
4. Modèles de tâche et formalisme 70
    4.1. Eléments directeurs : fonctions et concepts 70
    4.2. Rigueur sémantique 70
    4.3. Puissance d'expression 71
        4.3.1. Niveaux de description 71
        4.3.2. Relations entre tâches / Attributs des tâches 71
        4.3.3. Relations entre les entrées et les sorties 72
    4.4. Lisibilité 72
    4.5. Facilité / Complexité d'utilisation 73
5. Synthèse 74

Chapitre III : Evaluation ergonomique 77
1. Classifications usuelles 80
1.1. Approches prédictives et approches expérimentales 80
1.2. Présence virtuelle ou réelle de l'utilisateur et du système 82
1.3. Lacunes des taxonomies existantes 83
2. Une taxonomie pour le choix de méthodes d'évaluation 84
2.1. Moyens humains 84
2.2. Ressources matérielles 86
2.3. Connaissances requises 87
2.4. Facteurs situationnels 88
2.5. Résultats fournis 90
2.6. Un cadre taxonomique simplifié 91
2.6.1. Utilisateur et Interface 92
2.6.2. Savoir utilisé 93
2.6.3. Automatisation 93
3. Systèmes et techniques d'évaluations ergonomiques : état de l'art 94
3.1. Techniques non automatisées 94
3.1.1. Où le savoir heuristique domine 94
3.1.2. Utilisation de métriques 97
3.1.3. Une méthode formelle : Le Cognitive Walkthrough 99
3.1.4. Discussion 100
3.2. Techniques avec capture automatique 101
3.2.1. Playback 101
3.2.2. Le système de Hammontree 102
3.2.3. Discussion 103
3.3. Techniques d'évaluation et analyse automatique 103
3.3.1. Méthodes théoriques et formalisées 103
3.3.2. Où capture et analyse automatique cohabitent : Maximal Repeating Pattern (MRP) 107
3.3.3. Discussion 107
3.4. Techniques avec critique automatique 107
3.4.1. Knowledge-based Review of user Interfaces : KRI 108
3.4.2. Le système expert SYNOP 109
3.4.3. Discussion 110
4. Conclusion 110

CONTRIBUTION 113

Chapitre IV : ÉMA, un mécanisme d'analyse automatique pour l'évaluation ergonomique des interfaces utilisateur 115
1. Positionnement d'ÉMA 117
1.1. Moyens humains 118
1.2. Ressources matérielles 119
1.3. Connaissances requises 119
1.4. Facteurs situationnels 120
1.5. Résultats fournis 121
2. ÉMA, présentation technique 121
2.1. La composante de la tâche 122
2.1.1. Les principes 122
2.1.2. Notation graphique 124
2.1.3. Notation textuelle 129
2.2. La composante de l'utilisateur 132
2.2.1. Notion d'événement 132
2.2.2. Structure du fichier de capture 134
2.3. Le mécanisme d'analyse 135
2.3.1. Règle 1 : Détection de rupture 136
Exemple de détection de rupture 137
2.3.2. Règle 2 : Annulation immédiate 138
Exemple d'annulation immédiate 140
2.3.3. Règle 3 : Répétition 141
Exemple de répétition 141
2.3..4. Règle 4 : Invalidité 142
Exemple d'invalidité 142
3. ÉMA et principes d'ergonomie 143
3.1. Rupture dans l'espace des tâches 144
3.2. Annulation immédiate 145
3.3. Répétition 145
3.4. Invalidité 146
3.5. En synthèse 146
4. Conclusion 147

Chapitre V : ÉMA, Etudes de cas 149
1. Justification du choix des plates-formes d'expérimentation 151
2. Compo 153

2.1. Présentation générale 153
2.2. Les procédures élémentaires considérées pour le graphe des séquences 156
2.3. Mise en œuvre de la capture 157
2.3.1. Technique utilisée 158
2.3.2. Le recueil des données 159
2.4. Analyse selon les schémas d'ÉMA et interprétation des résultats 159
2.4.1. Détections de rupture 160
2.4.2. Annulations immédiates 162
2.4.3. Répétitions 162
2.5. Conclusion 163
3. Distributeurs Automatiques de Billets : le DAB initial et le DAB modifié 164
3.1. Présentation générale 164
3.1.1. DAB initial, sans retour sur les choix 165
3.1.2. DAB modifié avec choix multiples 167
3.2. Les graphes des séquences 170
3.2.1. Les procédures élémentaires considérées 170
3.2.2. DAB Initial : Le graphe des séquences 171
3.2.3. DAB modifié : Le graphe des séquences 172
3.3. Mise en œuvre de la capture 172
3.3.1. Technique utilisée 173
3.3.2. Le recueil des données 174
3.4. Analyses heuristiques et apports de l'analyse fournie par ÉMA 174
3.4.1. Bases de l'évaluation heuristique de l'expert 174
3.4.2. DAB initial sans retour sur les choix 176
3.4.2.1. Résultats de l'évaluation de l'expert 176
3.4.2.2. Analyse fournie par ÉMA 177
3.4.3. DAB modifié avec choix multiple 179
3.4.3.1. Résultats de l'évaluation de l'expert 179
3.4.3.2. Analyse fournie par ÉMA 181
3.5. Conclusion 183
4. Médiathèque 183
4.1. Présentation générale 183
4.2. Le graphe des séquences 185
4.2.1. Les procédures élémentaires 185
4.2.2. Le graphe des séquences 186
4.3. Mise en œuvre de la capture 187
4.3.1. Technique utilisée 188

4.3.2. Le recueil des données.....189
4.4. Analyse fournie par ÉMA et interprétation des résultats.....189
4.4.1. Annulations immédiates.....189
4.4.2. Répétitions.....190
4.4.3. Invalidités.....191
4.5. Conclusion.....192
5. Conclusion.....192

CONCLUSION 195

1. Contribution de la thèse 197
2. Les limites d'ÉMA 199
3. Les perspectives de travail 200
    3.1. Les perspectives à court et moyen terme 200
        Ajout de fonctionnalités à ÉMA 200
        Evaluation d'interfaces multimodales 201
        Intégration d'ÉMA au sein d'un générateur d'interface 201
    3.2. Les perspectives à plus long terme 202

ANNEXES 203

Annexe 1 : Principes d'utilisation des diagrammes syntaxiques 205

Annexe 2 : Plate-forme d'expérimentation : Compo 209
1. Rapport technique sur la construction du fichier de capture 211
    ScruteInit 211
    ScruteEnd 212
    ScruteCb 213
    ScruteCbForIconPath 216
    ScruteSxt 217
    ScruteScrollCb 222
    ScruteRealizeToplevel 226
    ScruteIsToplevel 227
2. Fichiers de capture 228
    2.1. Format adopté initialement 228
    2.2. Fichier de capture 229

Annexe 3 : Plate-forme d'expérimentation : Distributeurs automatiques de billets 233
1. Extrait de script du DAB initial 235
    Stack: ATM Initial 235
    Background 3: bkgnd id 6173 236
    Background button 1: 1 236
    Background button 11: Bottom Button 236
    Card field 2: Displayed PIN 237

Card field 1: Cash 237
2. DAB initial 237
2.1. Graphe des séquences initial 237
2.2. Graphe des séquences annoté 238
2.3. Fichier de capture annoté 239
3. DAB modifié 242
3.1. Graphe des séquences initial 242
3.2. Graphe des séquences annoté 243
3.3. Fichier de capture annoté 245
4. Texte de l'évaluation retranscrite par Ian Denley 248
4.1. Evaluation du DAB modifié 248
4.2. Evaluation du DAB initial 249

Annexe 4 : Plate-forme d'expérimentation : Médiathèque 251
1. Graphe des séquences : représentation graphique 253
2. Graphe des séquences annoté 255
3. Fichier de capture annoté 263

Références bibliographiques 267

Table des matières 283

Résumé :
L'ingénierie des Interfaces Homme-Machine s'est manifestée jusqu'ici par le développement de modèles et d'outils d'aide à la construction d'interfaces utilisateur. Dans le processus itératif de développement des interfaces, l'évaluation est souvent négligée par les développeurs ou reste le produit d'une démarche artisanale. Notre but est de fournir une aide automatique en vue d'une évaluation plus performante.

Après une revue de l'état de l'art, nous proposons deux espaces taxonomiques qui permettent de situer les techniques d'évaluation actuelles et les modèles de tâches possibles. Nous proposons ensuite ÉMA, un mécanisme d'analyse automatique pour l'évaluation ergonomique des interfaces utilisateur. L'analyse utilise trois sources d'information : une base de profils de comportement modélisés sous forme de règles, une représentation formelle de l'espace des tâches réalisables avec le logiciel testé, ainsi que les données comportementales enregistrées au cours des sessions d'utilisation du logiciel testé. ÉMA est aujourd'hui un détecteur d'anomalies. Il conviendra dans nos perspectives d'extension, d'en faire un critique explicatif, voire correctif.

Mots clés :
Interaction Homme-Machine, Ergonomie des logiciels, Génie Logiciel, Evaluation de l'utilisabilité d'interfaces utilisateur, Modèle de tâche, Capture du comportement.

Abstract:
Until now, Human-Computer Interaction has been primarily concerned with the development of models and tools to assist the construction of user interfaces. In the interface development iterative process, evaluation is often neglected by the developers or is the product of craft approaches. Our goal is to provide an automatic help to increase evaluation performance.

After a review of the state-of-the-art, we submit two frameworks to classify actual evaluation techniques and task analysis mechanisms. We then present ÉMA, an automatic analysis mechanism for the ergonomic evaluation of user interfaces. The analysis uses three sources of information: a set of heuristic rules that model patterns of behaviour, a data-flow oriented dialogue model of the tested interface, and the acquisition, during the interaction, of the end user's actions. ÉMA is today a usability anomalies detector. In the long term, ÉMA extensions consider providing a software usability explanatory critic, as well as a corrective critic.

Key words:
Human-Computer Interaction, Software ergonomics, Software engineering, Usability evaluation of user interfaces, Task model, Behaviour acquisition.