Plan du cours

Introduction Générale

L'informatique est la discipline permettant de résoudre automatiquement des problèmes au moyen des programmes. Cette discipline permet alors le remplacement de l'être humain par la machine afin d'accomplir des tâches journalières fastidieuses.

Pour atteindre la solution automatique, on doit passer tout d'abord par la solution manuelle en décrivant le problème entre main en suivant le processus de développement d'un logiciel de l'analyse de besoin comprenant la description informelle et formelle, en passant par la conception architecturale et détaillée, pour atteindre finalement un code source compilable voire un exécutable.

Cependant, loin des mathématiques, la plupart des problèmes rencontrés n'ont pas vraiment une description formelle bien précise. En effet, c'est facile de résoudre ces problèmes manuellement sans savoir vraiment comment on a fait ça. Ce sont des problèmes qui nécessitent plusieurs informations qui font partie de la vie de l'être humain, un niveau d'intelligence et de raisonnement d'importance et même d'intervenir d'autres informations pouvant être rencontrées dans d'autres contextes. On est en face alors des problèmes mal définis où il n'y a pas vraiment une description formelle du problème et par conséquent pas des solutions automatiques exactes et pas des résultats extrêmement satisfaisants.

Les systèmes, ciblant des problèmes mal définis, et même leurs résultats nécessitent toujours une amélioration. Cette amélioration peut provenir de l'utilisateur du système via l'interface au moyen de son feedback. En effet, l'utilisateur, via l'interface, après la visualisation des résultats initiaux, peut injecter à la machine une information supplémentaire, en plus de sa requête remise, connue sous le nom de feedback. Ce feedback va, plus ou moins, contribuer à ajuster l'état interne du système et son comportement interne ce qui influence positivement la performance du système.

Ces systèmes informatiques qui tiennent en compte l'utilisateur que ce soit au moment du développement du système ou à travers la forte interaction avec l'utilisateur lors de l'exécution sont qualifiés comme des systèmes interactifs.

Tout au long de ce cours, nous introduisons alors ce genre de systèmes (les systèmes interactifs) ainsi que leurs différentes notions de base, les architectures proposées dans ce contexte ainsi que leurs différentes technologies.

Chapitre 1 : Introduction aux Systèmes Interactifs

1. Introduction

Ce chapitre présente les principes, les méthodes, et les outils pour le développement des systèmes interactifs. Comme le domaine de génie des systèmes interactifs (GSI) est une continuité du domaine de l'interface Homme-Machine (IHM), au-delà de la conception centrée sur l'utilisateur, l'accent est alors mis sur les notions de l'IHM.

2. Définitions

• ***Un système informatique*** *:* est un ensemble des moyens informatiques et de télécommunication ayant pour finalité d'élaborer, traiter, acheminer, présenter ou déduire des données. En grosso modo, un système informatique se constitue d'un noyau fonctionnel et une interface assurant la communication avec l'environnement externe y compris l'utilisateur.

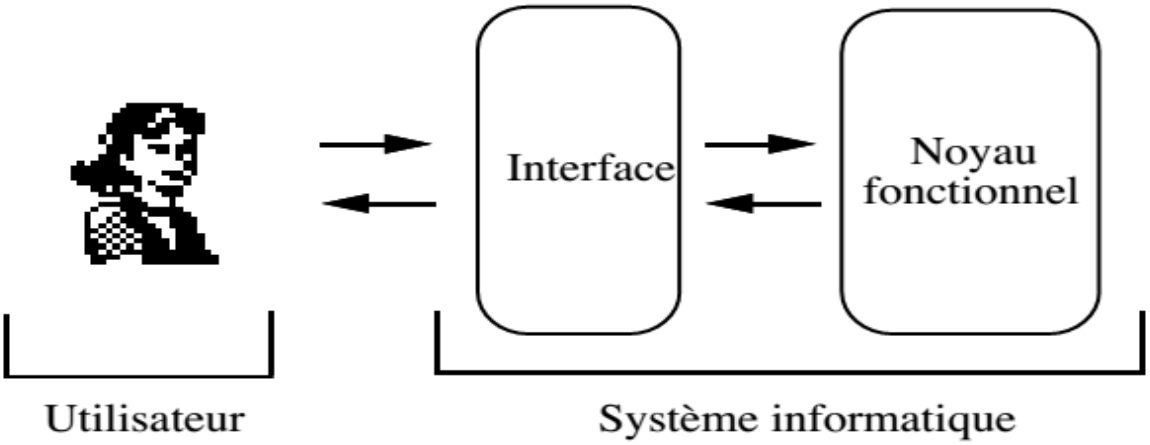

Fig.1.1 Un système informatique et son utilisateur [1].

• ***Un système interactif :*** est un système informatique dont le fonctionnement dépend d'informations fournies par un environnement externe qu'il ne contrôle pas [2]. En effet, le système interactif prend en compte, au cours de son exécution, des informations communiquées par le ou les utilisateurs du système, et qui produit, au cours de son exécution, une représentation perceptible de son état interne. Les systèmes interactifs sont également appelés les systèmes ouverts par opposition aux systèmes fermés ou autonomes dont le fonctionnement peut être entièrement décrit par des algorithmes [2]. un système interactif est donc un système ouvert : les entrées fournies au système dépendent de ses sorties, d'une façon qui n'est pas accessible au système. Un système interactif est aussi

différent d'un système algorithmique [2] ou une application de type « batch », lisant au début de leur exécution des données prédéfinies et produisant à la fin de leur exécution des résultats, dans le fait qu'il doit pouvoir réagir à tout moment même lorsqu'il est déjà en train d'effectuer une tâche.

- ***L'interface :*** est l'ensemble de dispositifs matériels et logiciels qui permettent, à un utilisateur, de commander, contrôler, et de superviser un système interactif [2].
- ***L'interface Homme-Machine :*** est la discipline consacrée à la conception, la mise en œuvre, et à l'évaluation des systèmes informatiques interactifs destinés à des utilisateurs humains ainsi qu'à l'étude des principaux phénomènes qui les entourent [2]. L'IHM ne se limite pas aux ordinateurs de bureau, mais elle englobe aussi les périphériques portables (PDA, téléphones,…), distributeurs de billets, appareils électroménager, etc. Comme montré dans la *Fig.1.2*, l'interaction homme-machine est un domaine multidisciplinaire où plusieurs domaines interviennent.
- ***Le degré d'Interactivité :*** le degré d'interactivité d'un système peut se mesurer au nombre et à la nature de ses échanges avec les utilisateurs [2]. Deux éléments importants ont contribué à l'augmentation du degré d'interactivité [2] : (1) la possibilité d'exécution en parallèle de plusieurs tâches, et (2) l'avènement des interfaces graphiques.
- ***L'Utilisabilité :*** l'utilisabilité [2] peut être définie comme le degré selon lequel un produit peut être utilisé, par des utilisateurs identifiés, pour atteindre des buts définis avec efficacité, efficience et satisfaction, dans un contexte d'utilisation spécifié.

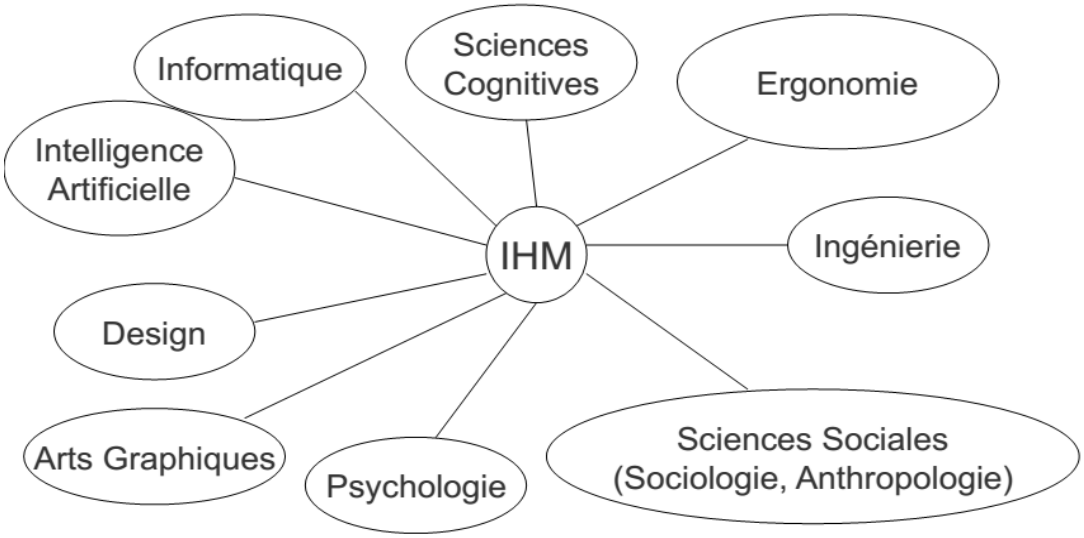

Fig.1.2 La multidisciplinarité de l'IHM.

• ***Le feedback :*** le feedback est l'action par laquelle le receveur d'une consigne renvoi une confirmation de compréhension ou un compte rendu d'exécution à l'émetteur initial. Il s'agit donc d'une action « retour ». ce mécanisme, considéré de nos jours comme un truc d'amélioration pour les systèmes interactifs, est beaucoup plus utilisé dans le domaine de la recherche d'information permettant de poursuivre une recherche d'information basée sur des mots clés.

3. Interfaces et Systèmes Interactifs : Problèmes et Défis

Au-delà de l'aspect esthétique, une bonne interface [2] doit prendre en compte l'utilisabilité en permettant une interaction facile et naturelle tout en incluant les composants ou les commandes nécessaires permettant d'accéder aux différentes fonctionnalités du système. Cependant, une mauvaise interface peut engendrer [2] :

- Apprentissage long et difficile.
- Fatigue, risque d'erreurs.
- Perte de productivité.
- Désorientation.
- Coûts de formation élevés.
- Accidents, désastres.
- Irritation, frustration, gêne, anxiété, insatisfaction.

L'interface a connu plusieurs développements sur le plan chronologique [2] :

• ***Interfaces à ligne de commandes*** *:* donnent accès à une commande (une fonction du système) en utilisant un langage de commandes imposé par le système. Ce genre d'interface est destiné aux utilisateurs experts.

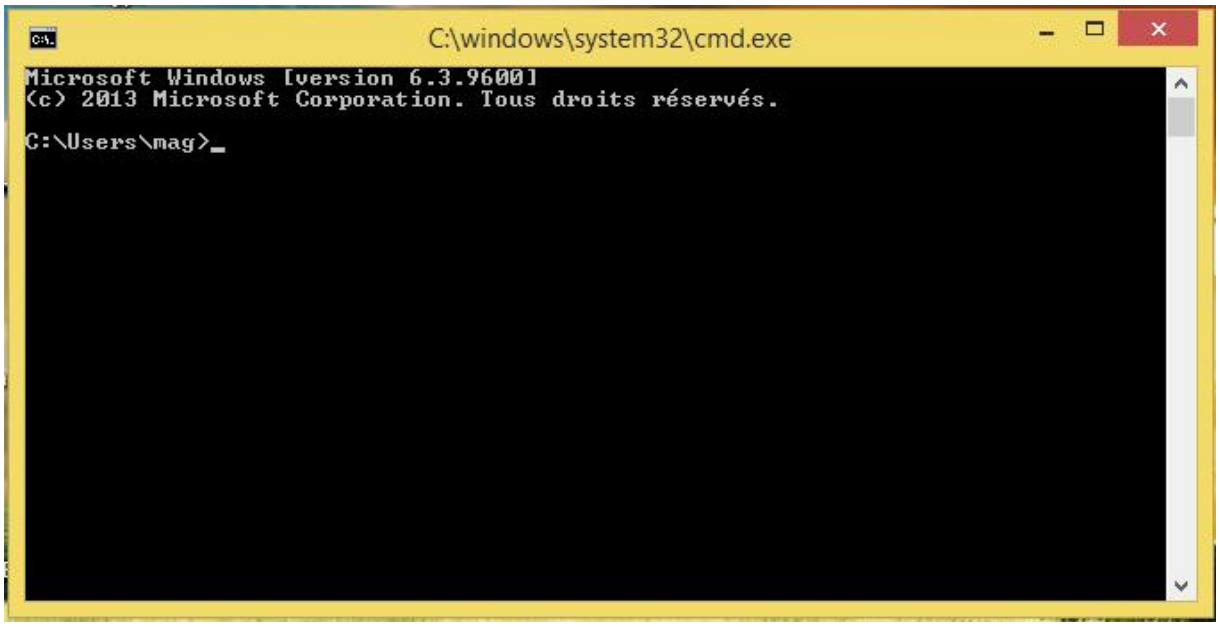

Fig.1.3 Interface à ligne de commandes.

• ***Menu et Écrans de saisie*** : donnent accès à une application (un sous ensemble des fonctions du système). Le dialogue offert par telles interfaces est contrôlé par le système, à titre d'exemple : les interfaces du web.

Fig. 1.4 écran de saisie.

- ***Multifenêtrage, interfaces iconiques et manipulation directe :*** donnent accès à l'ensemble des fonctions du système.

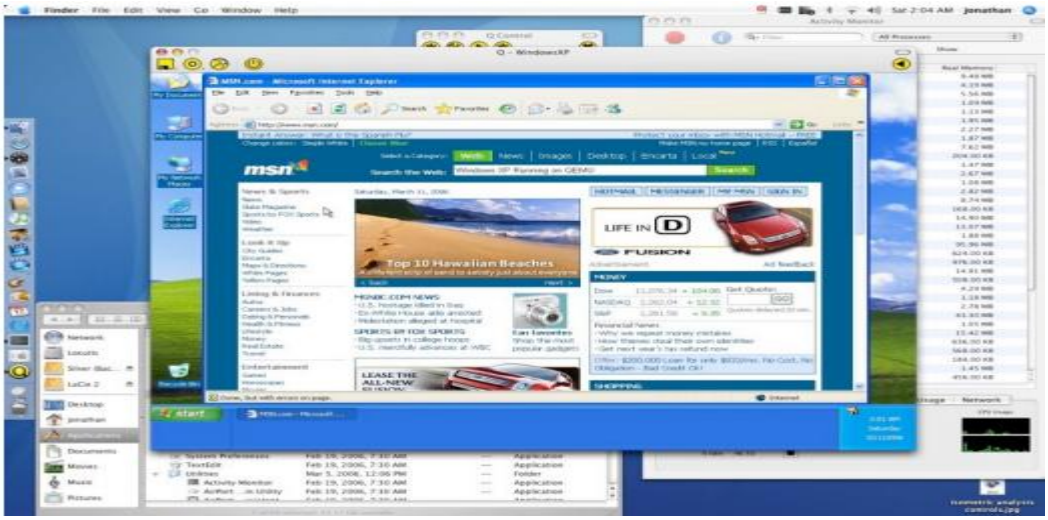

Fig. 1.5 Multifenêtrage et interfaces iconiques.

La conception d'interface est un processus itératif : (1) la définition de l'interface change en même temps que l'interface est conçue. En plus, (2) l'implémentation est influencée par la définition de la tâche et de l'interface.

Les concepteurs d'interface doivent faire face à de nombreux défis [2] :

- Large gamme d'utilisation.
- Différents contextes d'utilisation.
- Différentes configurations matérielles.
- Traduction de l'information dans différentes langues.

Deux problèmes peuvent être rencontrés lors de développement d'un système interactif [2] : (1) Les informaticiens sont souvent formés à l'algorithmique, au calcul, mais pas à la réactivité, et (2) les langages de programmation les plus utilisés ont été conçus pour le calcul, pas pour la réactivité.

Classiquement, un système interactif est divisé en deux parties : l'interface de l'utilisateur et le noyau fonctionnel. L'interface de l'utilisateur contient les éléments logiciels et matériels dédiés à la capture des entrées de l'utilisateur et à la restitution des sorties du système. Le noyau fonctionnel contient le reste du système, c'est-à-dire ses composants de calcul et de stockage de l'information.

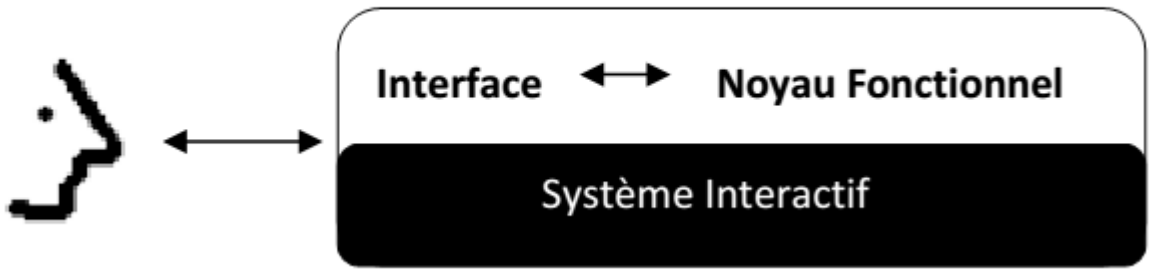

Fig.1.6 Décomposition d'un Système Interactif [3].

Cette décomposition ne doit pas laisser croire que ces deux parties sont forcément des modules logiciels indépendants ni surtout qu'elles peuvent être conçues indépendamment. S'il est vrai que l'on est souvent conduit à développer des interfaces pour des noyaux fonctionnels existants « *Legacy Systems* », la seule façon de développer des systèmes interactifs de qualité est de concevoir l'ensemble du système, en prenant en compte les caractéristiques de l'utilisateur au même titre que les contraintes techniques du noyau fonctionnel.

De nos jours, les systèmes interactifs constituent une part croissante des applications informatiques. La conception, le développement, et l'évaluation des systèmes interactifs nécessitent des méthodes et des techniques particulières afin de prendre en compte l'utilisateur.

Chapitre 2 : Les Systèmes Interactifs : Généralités, Historique, et Notions de Base

1. Introduction

Ce chapitre présente les principes, méthodes et outils pour le développement des systèmes interactifs. L'accent est mis sur la notion de conception centrée sur l'utilisateur, qui implique un style itératif de *conception-développement-évaluation*, et sur les applications utilisant des interfaces graphiques. Quelques éléments issus de la psychologie, pertinents pour la conception de systèmes interactifs, ont été aussi mis en avant.

2. Modèle Conceptuel d'un Système Interactif

Comme toute application informatique, un système interactif contient un certain nombre de fonctionnalités distribuées dans différents modules selon une architecture logicielle. La séparation *interface/noyau* résulte du partage des fonctionnalités d'un système interactif entre fonctions de l'interface et fonctions (internes) du système. Ce découpage fonctionnel peut être aisément raffiné : l'utilisateur doit pouvoir fournir des entrées au système (appelés commandes) et le système doit pouvoir présenter le résultat des commandes à l'utilisateur. Cela conduit au modèle conceptuel « *générique* » (c'est-à-dire indépendant de toute application) de la *Fig.2.1*. Lors de la conception d'un système interactif particulier, ce modèle générique est détaillé en définissant plus précisément chacun de ses composants.

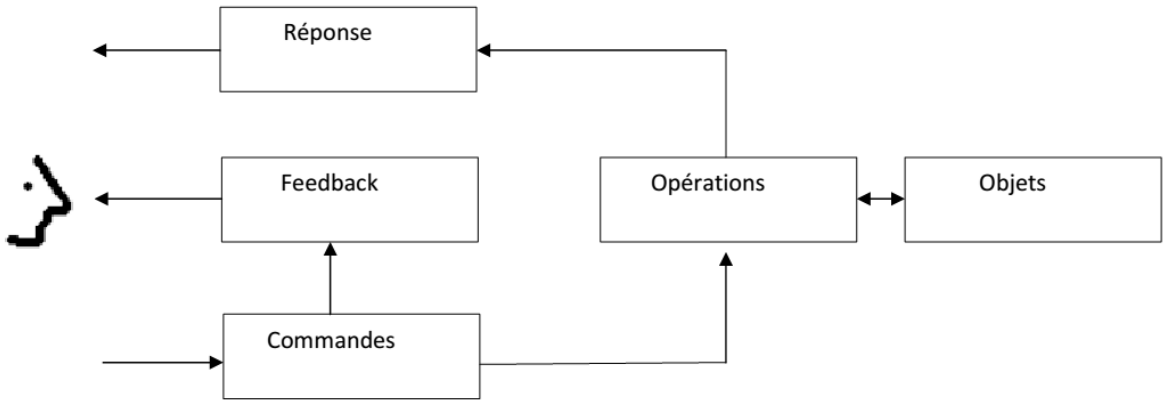

Fig.2.1 Modèle conceptuel générique [3].

Dans ce modèle, les *objets* sont les objets du domaine de l'application : les mots, paragraphes, et sections d'un éditeur de texte, les enregistrements et les relations d'une

base de données, etc. Ces objets sont accédés et modifiés par des opérations internes, activées par l'utilisateur par l'intermédiaire de commandes. Par exemple, la commande de fermeture d'une fenêtre peut déclencher plusieurs opérations : sauvegarde du contenu de la fenêtre, déverrouillage d'enregistrements dans une base de données, etc.

Certaines commandes correspondent à une action instantanée de l'utilisateur : sélectionner un objet par pointage ou utiliser une touche de fonction par exemple. D'autres commandes nécessitent plusieurs actions élémentaires : faire un *cliquer-tirer* d'un icône « *drag-and-drop* », activer une commande dans un menu qui ouvre une boîte de dialogue, taper une commande textuelle suivie de *RETURN*, etc. Dans ce cas, le système peut (doit) produire des sorties informant l'utilisateur qu'il fait bien ce qu'il pense faire : pendant le *cliquer-tirer*, l'icône du fichier suit le curseur de la souris, au fur et à mesure que l'on tape une commande, les caractères s'affichent, etc. Ce retour d'information est appelé *Feed-back*.

Enfin, lorsque les opérations résultant de l'activation d'une commande sont exécutées, le système produit des *réponses* perceptibles par l'utilisateur : le *cliquer-tirer* ouvre une fenêtre, ou fait changer la forme de l'icône de corbeille par exemple.

3. Conception centrée sur l'utilisateur

Des connaissances, réelles ou supposées, des utilisateurs et de l'utilisation du système sont fondamentales pour concevoir l'interface qui sera proposée à l'utilisateur. Une fois le système réalisé, il faudra confronter les hypothèses réalisées lors de la conception avec l'utilisation effective du système. La prise en compte de l'utilisateur est donc nécessaire aussi bien dans les phases de conception que d'évaluation. On se rend souvent compte, lors de l'évaluation, que les hypothèses faites étaient erronées, ou que le style d'interaction choisi ne tient pas ses promesses, ou que les utilisateurs inventent des usages non anticipés du système.

Ceci conduit à intégrer la prise en compte des facteurs humains dans les modèles de développement des systèmes interactifs. On regroupe l'ensemble de ces modèles et des techniques associés sous le terme « *Conception centrée sur l'utilisateur* (*User Centered Design*) ». La *Fig.2.2* illustre le cycle de vie d'un logiciel interactif en mettant l'accent sur la prise en compte des utilisateurs.

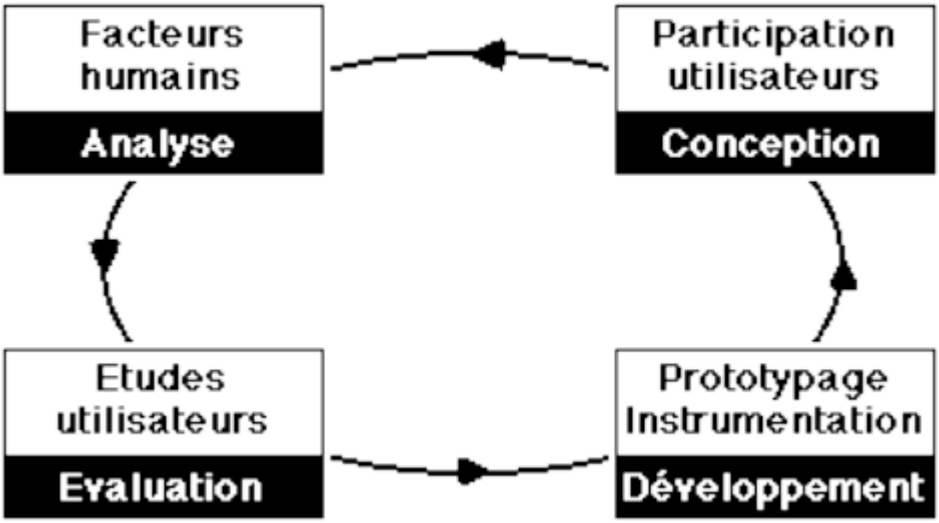

Fig.2.2 Conception centrée sur l'utilisateur [3].

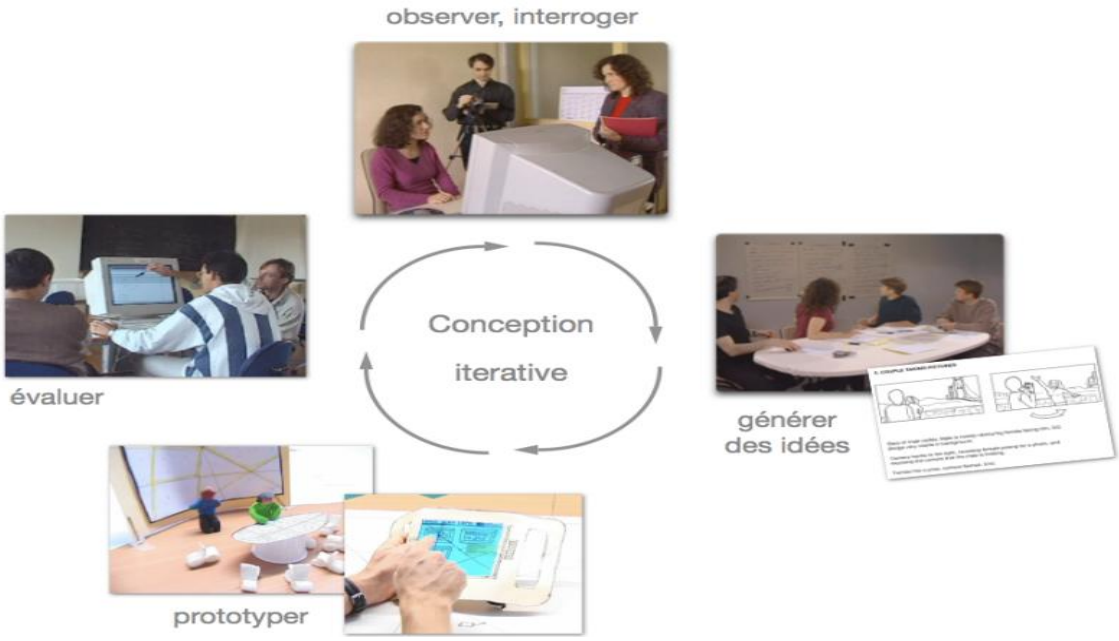

Fig.2.3 Conception itérative centrée sur l'utilisateur [2].

Lors de l'analyse, il s'agit notamment de prendre en compte les connaissances issues de la psychologie, d'analyser les outils existants, d'évaluer l'intérêt d'une solution informatique aussi bien au niveau des utilisateurs que de l'organisation. Au niveau de la conception, il s'agit de définir un modèle conceptuel qui sera aussi facile que possible à appréhender par les utilisateurs, et si possible de faire participer les utilisateurs à la conception. Pendant le développement, il s'agit de pouvoir produire des prototypes permettant d'évaluer rapidement certains choix de conception. Enfin, lors de l'évaluation, il s'agit de vérifier, par

des études avec des utilisateurs, si les hypothèses faites lors de la conception sont vérifiées dans le système final.

Si ce n'est pas le cas, il faut refaire un cycle d'*analyse-conception-développement-évaluation* pour corriger les problèmes rencontrés. Ce cycle de conception itérative est inhérent aux systèmes interactifs pour la simple raison que l'utilisateur humain et les modes d'utilisation ne sont pas formalisables d'une façon telle que l'on puisse les considérer comme fixes ou connus. Quand bien même cela serait possible, l'usage d'un nouveau système par les utilisateurs change leurs modes de travail et donc leurs habitudes, leurs besoins et leurs attentes. Ainsi, on a une coadaptation entre le système et les utilisateurs : le système est conçu en fonction des besoins des utilisateurs, et leurs besoins changent en fonction des caractéristiques du système. La seule façon d'atteindre un point d'équilibre est d'itérer la conception.

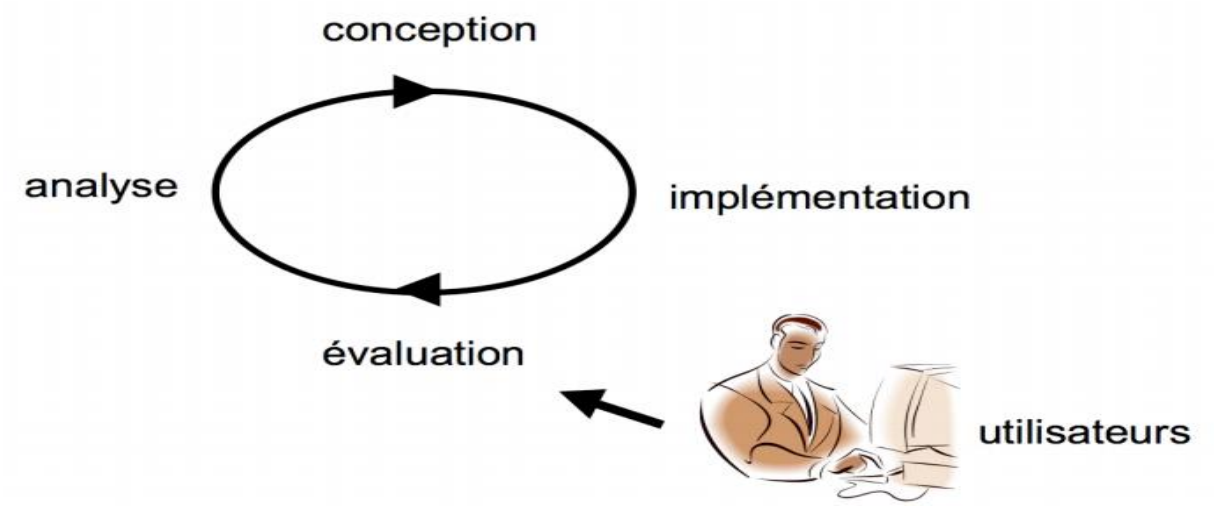

Fig.2.4 conception itérative centrée sur l'utilisateur.

Bien entendu, l'utilisateur n'est pas le seul facteur à prendre en compte lors de la conception. Comme pour tout système informatique, il faut prendre en compte les facteurs organisationnels. Cependant, l'importance de la prise en compte des facteurs humains et le fait qu'ils imposent un cycle de développement itératif sont à l'origine du terme « *conception centrée sur l'utilisateur* ». Pour mettre en œuvre un tel style de conception, il faut un minimum de connaissances sur les utilisateurs. Ces connaissances nous sont fournies par la psychologie expérimentale. D'autres sources d'information sont utiles, comme l'ergonomie des logiciels.

4. Eléments de Psychologie appliqués aux Systèmes Interactifs

L'être humain perçoit son environnement par l'intermédiaire de cinq cens. Il peut agir sur son environnement par l'intermédiaire de son système moteur (action physique) et par l'intermédiaire du langage. Enfin, il est doté d'un système cognitif qui lui permet de mémoriser des informations et de planifier et de contrôler ses actions en fonction de ses perceptions et de sa mémoire [2].

- **Perception**

La perception est l'activité par laquelle un sujet fait l'expérience d'objets ou de propriétés présents dans son environnement. Cette activité repose habituellement sur des informations délivrées par les sens. Chez l'espèce humaine, la perception est aussi liée aux mécanismes de cognition. Le mot « perception » désigne :

- ✓ soit le processus de recueil et de traitement de l'information sensorielle ou sensible (en psychologie cognitive par exemple).
- ✓ Soit la prise de conscience qui en résulte (en philosophie de la perception notamment).

En psychologie expérimentale, chez l'être humain en particulier, on distingue des échelles de perception consciente d'une part, et la perception inconsciente, d'autre part. Celle-ci est qualifiée parfois d'implicite ou subliminale.

La perception d'une situation fait appel tout à la fois aux sens physiologiques d'un organisme et à ses capacités cognitives, à un niveau élémentaire ou conscient.

En psychologie cognitive, la perception est définie comme la réaction du sujet à une stimulation extérieure qui se manifeste par des phénomènes chimiques, neurologiques au niveau des organes des sens physiologiques et au niveau du système nerveux central, ainsi que par divers mécanismes qui tendent à confondre cette réaction à son objet par des processus tels que la représentation de l'objet, la différenciation de cet objet par rapport à d'autres objets.

- **Vue**

Le système visuel a été très étudié par la psychologie expérimentale. Sur le plan physiologique, c'est un système très complexe et très spécialisé. Il est cependant loin d'être

parfait : beaucoup d'animaux ont des systèmes visuels plus performants. Par ailleurs on peut tromper notre système visuel comme le démontrent de nombreuses illusions d'optique.

Le champ de la perception visuelle est de 180°. Cependant, les caractéristiques de la vision sont différentes entre le centre (*focus d'attention*) et la périphérie. Par exemple, la perception périphérique est moins sensible aux couleurs et plus sensibles aux mouvements. Ainsi, il est souvent très difficile de repérer un oiseau sur un arbre, mais on le perçoit immédiatement au moment où il s'envole.

L'œil est capable de percevoir le mouvement grâce à des récepteurs spécialisés et grâce à la fusion entre percepts successifs effectuée au niveau cérébral. Que ce soit au cinéma, à la télévision, ou sur un écran d'ordinateur, le mouvement est produit par l'affichage successif d'images fixes et exploite donc la fusion sensorielle.

- **Ouïe**

L'ouïe a été moins étudiée que la vision. Au plan physiologique, le système auditif est très perfectionné. La partie mécanique de l'oreille capte les vibrations de l'air. Sa bande passante est impressionnante, puisque le rapport entre les variations les plus faibles et les plus importantes qu'elle peut percevoir sans détérioration est de l'ordre d'un million. Aucun système mécanique n'est capable de telles performances.

Contrairement à la vue, on peut entendre sans écouter. En réalité, notre ouïe est toujours active : les signaux sonores sont constamment interprétés, même si un grand nombre ne parviennent pas à notre conscience.

Malgré ses capacités, l'ouïe est très peu sollicitée par les systèmes interactifs actuels, en dehors de « *bips* » plus ou moins variés et de signaux enregistrés. Certains travaux, restés au stade de la recherche, ont montré l'augmentation de performance, pour certaines tâches informatiques, qui peuvent résulter de l'usage du son, à la fois pour décharger le canal visuel, pour faciliter le suivi des activités d'arrière-plan ou pour notifier à l'utilisateur de l'occurrence d'évènements asynchrones.

- **Toucher**

On peut associer au toucher trois composantes : le toucher, la proprioception, et la kinesthésie.

Le toucher concerne la perception de la température, de la pression et de la douleur. C'est en particulier la perception de la pression qui nous permet d'évaluer la texture d'un objet. Les capteurs tactiles sont beaucoup plus denses dans certaines régions du corps (doigts et bouche notamment) quand dans d'autres (dos par exemple). D'autres parts, ces capteurs sont sensibles aux variations de pression. Comme pour la vue et l'ouïe, c'est dans une situation de perception active que l'on peut le mieux exploiter ce sens tactile : pour évaluer la matière d'un objet (bois, plastique, papier, coton, peau, etc.), on touche l'objet en déplaçant ses doigts dessus. C'est dans les systèmes interactifs pour handicapés que l'on a le plus cherché à exploiter le sens tactile, et en particulier pour les aveugles et malvoyants. Divers dispositifs existent (mécaniques, piézo-électriques, pneumatiques,..) permettant de restituer une information tactile.

Une autre composante du toucher est notre capacité à connaître la configuration de notre corps dans l'espace : je sais, sans avoir à la regarder, si ma main est ouverte ou fermée, si je suis debout, accroupi ou allongé, etc. Combiné avec le sens tactile, ce sens dit « *proprioceptif* » nous permet notamment d'appréhender la forme des objets sans les voir. Les systèmes actuels de réalité virtuelle souffrent de l'absence d'utilisation de ce canal : lorsque l'on saisit un objet virtuel, aucune sensation tactile ni proprioceptive ne vient confirmer le contact et ne permet d'évaluer la forme, la texture, le poids de l'objet.

La dernière composante du sens tactile est la kinesthésie (qui, pour certains auteurs, englobe la proprioception). Le sens kinesthésique nous permet de connaître l'effort que font nos muscles par exemple lorsque l'on soulève un objet (poids) ou lorsque on pousse un objet (résistance). Comme la proprioception, l'exploitation de ce sens fait cruellement défaut aux systèmes actuels de réalité virtuelle.

- **Action**

L'homme a deux moyens d'action immédiate sur le monde qui l'entoure : l'action physique et le langage parlé.

✓ ***Action Physique***

L'action physique utilise le système moteur pour agir sur des objets de notre environnement. Chez l'homme, la main est l'organe le plus sophistiqué pour l'action

physique. L'action manuelle, qui est le moyen d'action unique des systèmes interactifs (à des rares exceptions), ne peut être découplée de la perception.

D'une part, c'est dans la main que les récepteurs tactiles sont les plus nombreux. Toute action manuelle sera donc exécutée avec un retour permanent d'information tactile et kinesthésique, même pour un périphérique « passif ». Ainsi, bien que, du point de vue de la machine, une souris, un trackball, un joystick ou une tablette graphique fournissent tous une position *2D* et sont donc équivalents, il n'en va pas de même pour l'utilisateur : il est par exemple bien plus difficile de signer avec une souris qu'avec un stylo sur une tablette, et c'est pratiquement impossible avec un joystick.

D'autre part, beaucoup d'actions manuelles sont guidées par la vue : lorsque l'on attrape un objet, le geste est ajusté, en temps réel, grâce à l'information visuelle qui indique la position relative de la main et de l'objet. Ce guidage peut même se faire si l'objet est dans le champ de vision périphérique : lorsque l'on utilise un clavier et une souris, la main va de l'un à l'autre alors que le regard reste fixé sur l'écran pourtant la main est guidée par la vue.

Dans les tâches d'interaction manuelle, deux actions sont très fréquentes : la saisie d'un objet, et la manipulation d'un objet. La saisie a été étudiée empiriquement afin de déterminer s'il existait une loi permettant d'évaluer le temps d'une saisie. *Paul Fitts* a découvert une loi empirique pour les tâches de pointage. Il s'agit de pointer, avec le doigt, une cible circulaire de taille ***L*** à une distance ***D***. intuitivement, le temps de pointage doit augmenter si ***D*** augmente et si ***L*** diminue : le temps de pointage est plus long si la cible est plus éloignée ou si elle est plus petite. La loi de *Fitts* s'exprime sous la forme suivante :

$$t = 0.1log2D/L$$

Le temps t (en seconde) de pointage d'une cible de taille L à une distance D est proportionnel au logarithme de *2D/L*. Ainsi, si la distance se double ou si la taille de la cible se diminue de moitié, le temps de pointage ne se double pas mais croît selon une loi logarithmique. Cette loi empirique s'applique également au pointage avec un dispositif tel que souris, joystick ou tablette.

La loi de ***Fitts*** est l'une de très rares lois quantitatives que nous fournit la psychologie pour la réalisation des systèmes interactifs. Elle montre qu'un temps de pointage typique est

compris entre ½ et 1 secondes. Si l'on accumule les temps de pointage sur une longue période d'interaction, on découvre que tout gain sur ces temps de pointage est appréciable. Dans les logiciels actuels, l'utilisation d'équivalents-claviers et accélérateurs divers est souvent un moyen d'éviter des pointages longs et il est caractéristique d'observer que les utilisateurs experts, qui optimisent l'utilisation du logiciel, exploitent au maximum ces raccourcis. Certaines techniques d'interaction peuvent également faire gagner du temps en diminuant le nombre ou le degré de difficulté des pointages, sans recourir à des raccourcis que l'utilisateur doit mémoriser.

- **Le Langage**

L'étude du langage est un volet très important de la psychologie (complété par la linguistique qui s'intéresse plutôt aux langues). En informatique, les langages de programmation reprennent certaines caractéristiques des langages humains, et les interfaces conversationnelles sont fondées sue des langages de commandes qui sont des langages de programmation simplifiés. D'autre part, des systèmes ont été développés, depuis longtemps, permettant une communication en langue naturelle entre l'utilisateur et le système informatique.

L'utilisation d'un langage (parlé, écrit, gestuel, voire olfactif chez certains espèces) suppose que les acteurs de communication partagent ce langage. Chez l'homme, l'apprentissage d'une langue dure plusieurs années, et est en grande partie indissociable de l'acquisition d'une grande quantité de traits sociaux et culturels et de ce que l'on appelle « *le sens commun* ». Compte tenu de la complexité des langues humaines et de l'imbrication entre langage et environnement social et culturel, et malgré l'augmentation de la puissance de calcul des ordinateurs, les interfaces en langue naturelle sont restées confinées à des domaines d'application spécialisés, dans le quel on peut simplifier le vocabulaire, la syntaxe et le domaine du discours de façon suffisamment importante pour pouvoir donner au système informatique le rôle d'un interlocuteur.

5. Le Couplage Action-Perception

La perception est presque toujours un phénomène actif, et l'on ne peut complètement séparer la perception de l'action ; nos actions sont constamment régulées par notre perception et notre perception dirige constamment nos actions. Lorsque par exemple on

saisit un objet, la trajectoire de la main est constamment réajustée en fonction de la perception d'abord visuelle puis tactile (lorsqu'il y a un contact). On parle de *boucle perception-action*.

Selon certains psychologues (l'approche dite « *écologique* » de *Gibson* », l'être humain ne perçoit pas des unités d'information (image, son, …etc.) mais il perçoit des flux sensoriels. C'est l'extraction des invariants de ces flux qui lui permet d'appréhender le monde qui l'entoure, qu'il s'agisse de flux visuels, auditifs, tactiles, gustatifs ou olfactifs. La perception active consiste donc à agir sur ces flux de façon contrôlée (par exemple en déplaçant la tête), afin de faciliter l'extraction des invariants.

Sur le plan des systèmes interactifs, de nombreuses expérimentations ont montré que les techniques d'interaction qui exploitent la boucle *perception-action* sont plus performantes que les techniques « *passives* ». Malheureusement, ce message n'est pas encore passé dans l'industrie et la plupart des systèmes interactifs commerciaux sont « *passifs* ».

6. Historique de L'interaction Homme-Machine

L'histoire de l'interaction homme-machine est presque aussi vielle que l'histoire de l'informatique. En *1945*, *Vennevar Bush*, dans son célèbre article « *As We May Think* » décrivait un système électronique (imaginaire) permettant le stockage et la recherche d'informations, et inventait les concepts de l'hypertexte : navigation, indexation, et annotation. Au début des années *1960*, alors que les systèmes à temps partagé font leurs débuts, *Ivan Sutherland* crée le système *SketchPad* (*1963*), qui utilise comme écran un oscilloscope. *SketchPad* est un outil de dessin qui permet de créer des schémas par instanciation de modèles prédéfinis (à l'instar du modèle *classe-instance* des langages à objets. Simula date de la même époque). *SketchPad* permet la construction de structures hiérarchiques et utilise des contraintes pour définir les objets. L'interaction utilise un crayon optique. A la fin des années *1960*, *Douglas Engelbart* développe le système *NLS-Augment*. En *1968*, il présente une démonstration à *San Francisco* où l'on voit un système de traitement de texte hiérarchique, multimédia (texte et graphique) et hypertexte. C'est un système permettant le travail coopératif : on peut partager des fichiers, annoter des documents, utiliser une messagerie, et même un système de visioconférence avec partage d'écrans

informatiques et télépointeurs. *Engelbart* a inventé la souris, ainsi que le clavier à une main, les fenêtres, etc.

En *1969*, *Alan Kay* présente sa vision d'un ordinateur de poche qui préfigure les *PDAs* (*Personal Digital Assistants*). Dans les années *1970* et *1980*, les laboratoires *Xerox PARC* (*Xerox Palo Alto Research Center*) sont un creuset sans équivalent pour le développement de systèmes interactifs novateurs. Pendant les années 70, le projet *Smalltalk*, qui lancera les langages à objets, conduit à la réalisation de la première station de travail à écran graphique et souris, l'*Alto*, qui dispose également d'un réseau local (*Ethernet* a été inventé aussi à *Xerox PARC*). L'interface contient déjà des fenêtres, menus, barres de défilement et utilise la sélection directe à la souris. En *1981*, une nouvelle machine, le *Star*, utilise la métaphore du bureau avec des icônes représentant le système de fichiers et un certain nombre de commandes génériques. Le *Star* invente aussi la présentation *WYSIWYG* (*What You See Is What You Get*) : les documents apparaissent à l'écran sous la même forme que lorsqu'ils sont imprimés. Le *Star* sera un échec commercial mais inspirera *Apple* : en *1983*, l'*Apple Lisa* est une copie du *Star* mais, trop cher et trop étriqué, c'est aussi un échec commercial. En *1984*, le *Macintosh*, une évolution du *Lisa* vendu à un prix plus bas est un succès commercial sans précédent. Il faudra plus de *10 ans* pour que *Microsoft* intègre, à son *interface graphique Windows*, des fonctionnalités comme l'*interaction iconique* ou le *cliquer-tirer*.

Dans la seconde moitié des années *1980*, au *Massachusetts Institute of Technology* (*MIT*) à *Boston*, un grand projet, financé notamment par *Digital Equipement Corp.* (*DEC*), *IBM* et *Tektronix,* a pour objectif de fournir, à tous les étudiants du campus, des postes de travail accessibles librement. C'est le projet *Athena*, qui donnera naissance notamment au *X Window System*, qui deviendra l'interface graphique standard des stations *Unix*. A la fin des années *1980*, on voit apparaître les premiers systèmes de réalité virtuelle qui immergent l'utilisateur dans un monde synthétique. Au début des années *1990*, les physiciens du *CERN* inventent un système hypertexte qui s'appellera plus tard le *World Wide Web* et envahira la planète.

Malgré cette profusion d'inventions, qu'il faudrait compléter par tous les travaux, moins spectaculaires mais indispensables, de nombreux chercheurs et praticiens, il faut reconnaître que les systèmes interactifs actuels sont souvent loin des attentes des utilisateurs. Il n'a pas

besoin d'études pour constater que l'informatique n'a pas réduit la quantité de papier que l'on est amené à manipuler, mais au contraire l'a fait augmenter. C'est la preuve que la conception de systèmes interactifs de qualité est encore un domaine mal maîtrisé et souvent sous-estimé.

7. Historique des Styles d'Interaction

On peut classifier la majorité des systèmes interactifs en trois catégories selon le style d'interaction qu'ils emploient. Le terme «*Style d'interaction* » recouvre l'ensemble des méthodes, techniques, et règles d'interaction qui sont employées dans un système.

- **Systèmes Conversationnels**

Historiquement, le style conversationnel est le plus ancien : il correspond à un mode d'interaction langagière inspiré de la conversation entre humains. Le langage humain a également inspiré les langages de programmation, il n'est donc pas surprenant de le retrouver dans l'interaction.

Dans un système conversationnel, les commandes sont fournies en tapant un texte au clavier, généralement une ligne terminée par un *retour-chariot*. Un *feedback* permet à l'utilisateur de voir ce qu'il tape (et de le corriger), mais la commande n'est exécutée que lors de sa validation. A ce moment là, la commande est interprétée et transformée en opérations sur les objets internes du système (voir le modèle conceptuel générique de la *Fig.2.1*). Le système fournit alors une réponse qui est soit un message d'erreur, soit le résultat, éventuellement vide, de la commande. L'utilisateur peut alors entrer une nouvelle commande, etc. les invites de commandes comme ceux d'*Unix* sont des exemples des systèmes conversationnels.

L'inconvénient principal d'un système conversationnel est que l'utilisateur doit apprendre le langage de commandes et que les réponses du système sont parfois difficiles à interpréter. Par contre, pour des utilisateurs experts, ce mode d'interaction peut être plus efficace que tout autre notamment par la possibilité d'étendre le système en programmant ses propres commandes (comme par exemple les *Shell-scripts* sous *Unix*). D'autre part, les systèmes conversationnels impliquent un séquencement strict entre actions de l'utilisateur et réponses du système (à l'exception du fait que l'utilisateur peut interrompre le système si

celui-ci tarde à répondre). Il est donc difficile de mener plusieurs tâches simultanément, alors que c'est une activité courante dans la vie quotidienne.

- **Menus, Formulaires**

Les systèmes à base de menus et de formulaires sont une extension du modèle conversationnel destinée à faciliter l'interaction en rendant le langage de commandes explicite. Ainsi, la sélection d'une commande dans un menu nécessite de reconnaître cette commande dans une liste, alors qu'une interface conversationnelle oblige à se souvenir du nom de la commande. Les menus hiérarchiques permettent d'accéder à un grand nombre de commandes, au prix de commandes de navigation qui permettent de passer d'un menu à l'autre.

La syntaxe disponible avec un simple système de menus est très primitive. C'est pourquoi ce style d'interaction est complété par l'usage de formulaires, similaires dans leur principe aux formulaires papier : lorsque le formulaire lui est présenté à l'écran, l'utilisateur doit remplir ses champs (dans un ordre quelconque), puis valider le formulaire. C'est la validation qui déclenche l'exécution d'une commande par le système en générant une réponse sous forme d'un message d'erreur, d'un nouveau formulaire, d'un menu ou d'un document. La plupart des services *Minitel* sont des systèmes à base de menus et de formulaires, de même que beaucoup d'applications de gestion au sens large du terme comme par exemple les systèmes de réservation d'avions ou de trains.

Les systèmes à base de menus et de formulaires imposent, comme les systèmes conversationnels, un dialogue strict entre l'utilisateur et le système. Cependant, ces systèmes assistent l'utilisateur en lui fournissant des informations permettant de guider ses actions ; liste des commandes disponibles dans les menus, noms des champs dans les formulaires, etc. De plus, l'interaction avec un formulaire n'impose pas d'ordre de remplissage et en cas d'erreur, il n'est en général pas nécessaire de tout re-saisir. Le fait de présenter les objets informatiques sous une forme graphique et la possibilité d'agir directement sur ces objets (cliquer dans un champ pour le remplir, sur un bouton *OK* pour valider) renforce la sensation d'engagement de l'utilisateur. L'inconvénient de ce style d'interaction est qu'il ne peut convenir qu'à des tâches prédéfinies. Comme pour le style conversationnel, il n'est en général pas possible de traiter plusieurs tâches en parallèle ou

même hiérarchiquement. Par exemple, si l'un des champs à saisir est un numéro de client et que l'on ne connaît que son nom, il faudrait pouvoir utiliser le formulaire d'accès à la base de données des clients, sans avoir à abandonner le formulaire en cours. Enfin, les systèmes à base de menus et de formulaires ne sont en général pas programmables par l'utilisateur ; si l'on utilise souvent le même formulaire avec les mêmes valeurs de champs, il est probable que l'on doit les faire entrer à chaque fois.

- **Navigation (*Hypertexte*)**

Un *système hypertexte* permet la navigation dans un ensemble de *nœuds*, reliés entre eux par un ensemble de *liens*. Un nœud se présente typiquement comme une page à l'écran. Les liens peuvent apparaître dans le corps de la page (mots soulignés ou encadrés) et/ou dans des menus/palettes. L'interaction consiste simplement à sélectionner des liens. D'autres primitives de navigation complètent les liens : navigation avant et arrière dans les nœuds visités récemment, navigation directe vers un nœud par donner un nom, etc.

Des exemples des systèmes de navigation sont les navigateurs du *World Wide Web* et le système d'information d'*Emacs*. On trouve également des systèmes de navigation comme sous-systèmes d'autres systèmes interactifs, notamment les systèmes d'aide en ligne.

- **Manipulation directe**

L'avènement des stations de travail à écran graphique a amené les concepteurs d'interfaces à rapprocher l'interaction avec les objets informatiques de l'interaction avec les objets physiques. Ainsi, ils ont cherché à présenter les documents à l'écran sous une forme comparable à leur forme imprimée, tout en permettant de modifier ces documents en agissant directement dessus. Le développement de la métaphore du bureau procède de la même logique ; en représentant des objets informatiques tels que fichiers et dossiers par des images évocatrices (les icones), et en permettant l'action sur ces icones par l'intermédiaire de la souris, en espérant rendre les commandes possibles suffisamment intuitives pour éviter à l'utilisateur un apprentissage long et fastidieux.

Le terme « *manipulation directe* » a été inventé par *Ben Shneiderman* en *1983*, inspiré par certains logiciels de l'époque, notamment les premiers tableurs et les jeux vidéo. *Shneiderman* caractérise les applications à manipulation directe par 4 principes :

- Affichage permanent des objets d'intérêt (les objets d'intérêt sont les objets qui ont un sens pour l'utilisateur, qui appartiennent à son modèle mental : par exemple les mots, les lignes et les paragraphes d'un texte).
- Action directe (via un dispositif de désignation comme la souris) sur les objets d'intérêt plutôt que syntaxe complexe.
- Actions incrémentales et réversibles dont l'effet sur les objets d'intérêt est immédiatement visible.
- Structuration de l'interface en couches afin de faciliter l'apprentissage.

Comparé aux styles d'interactions précédents, la manipulation directe permet de traiter des tâches non prédéfinies, en particulier les tâches créatives : édition de texte, de dessins, de schémas, de partitions, etc. l'invention des systèmes de multifenêtrage, qui a accompagné celle de la manipulation directe, permet de mener de front plusieurs tâches, de façon entrelacée. La manipulation directe permet non seulement d'interagir avec plusieurs applications, mais aussi de transférer des données entre applications (par *cliquer-tirer* ou *couper-coller*, notamment).

L'inconvénient principal de la manipulation directe est de ne pas permettre aisément la programmation par l'utilisateur ; par exemple si l'on doit copier tous les fichiers qui ont un nom se terminant par « *.c* », il est probable que l'on doit faire cette opération à la main. Un autre problème, qui n'est pas inhérent au style d'interaction lui-même, est que de nombreuses applications n'exploitent pas la manipulation directe correctement ; beaucoup de commandes sont déclenchées par la combinaison d'une commande de menu et d'une (ou plusieurs) boîtes de dialogue. Il n'y a alors ni action directe sur les objets d'intérêt (la manipulation se fait dans la boîte de dialogue), ni effet immédiat de l'action (il faut valider pour voir l'effet). On peut alors parler de manipulation indirecte (proche en réalité du style *menus/formulaires*), et non de manipulation directe. A raison de cet état de fait tient en grande partie aux outils de développement aujourd'hui disponibles, qui simplifient le développement des parties de l'interface utilisant la manipulation indirecte sans apporter de solution satisfaisante pour mettre en œuvre les principes de la manipulation directe.

Deux catégories très répandues d'interfaces à manipulation directe sont : l'édition de documents et l'interaction iconique.

1) Édition de documents

Le principe du *WYSIWYG* (*What You See Is What You Get*) est de présenter un document à l'écran sous une forme la plus proche possible de sa forme imprimée : texte, schéma, partition musicale, etc. Ainsi, dans un système de traitement de texte *WYSYWYG*, l'équation $d = vb^2 - 4ac$ apparaît telle quelle, tandis que dans un système non *WYSIWYG*, elle aurait par exemple la forme

$$\S\backslash equal\backslash sqr\{b^2 - 4ac\}\S$$

L'interaction avec un document *WYSIWYG* utilise en général : (1) des techniques de manipulation directe, et (2) des techniques de manipulation « indirecte » (boîtes de dialogues, menus, palettes).

Pour mettre en œuvre les techniques de manipulation directe, il peut être nécessaire de violer le principe de *WYSIWYG*. Par exemple, dans un logiciel comme *Word*, on peut visualiser le texte en mode pleine page ou en mode défilement, on peut afficher les caractères normalement invisibles (tabulations, espaces, retour-chariot, etc.). Cependant, la notion de *WYSIWYG* est indépendante de celle de manipulation directe. On peut faire de la manipulation directe de représentations non *WYSIWYG*, et on peut éditer un document *WYSIWYG* par un langage de commande.

2) Interaction iconique

Une icône est une représentation graphique d'un objet ou d'un concept. Les icones doivent être conçus pour être aisément reconnaissables, mémorisables, et différentiables les uns des autres. La stratégie la plus efficace consiste à créer des familles d'icônes, comme par exemple sur le *Macintosh* ; la forme générale de l'icône indique s'il s'agit d'un document ou d'une application, le contenu indique de quel type de document il s'agit (texte, graphique…).

Fig.2.5 Présentation iconique.

L'interaction iconique consiste à effectuer des commandes par manipulation directe des icônes (sélection, déplacement) et les techniques usuelles de manipulation indirecte (menus, boîtes de dialogues).

- **Interaction par reconnaissance de traces**

Ce style d'interaction consiste à reconnaître les mouvements du périphérique de localisation par rapport à un vocabulaire gestuel prédéfini. Un geste définit une commande et éventuellement certains de ses paramètres (taille, portée du geste, point de départ et/ou d'arrivée). La position du geste détermine son objet. L'interaction est donc concise, mais invisible ; il est difficile de savoir quel est le vocabulaire.

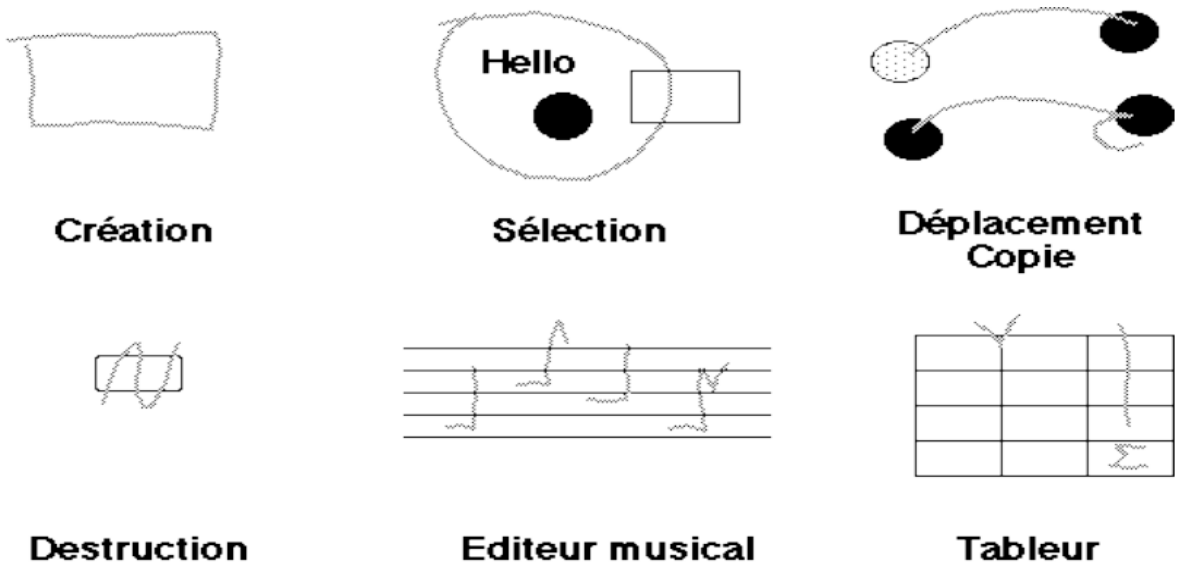

Fig.2.6 Interaction par reconnaissance de traces.

- **Autres Styles**

D'autres styles d'interaction existent ou émergent. On peut notamment citer la réalité virtuelle, la réalité augmentée, l'interaction augmentée, l'interaction multimodale, le collecticiel.

La réalité virtuelle immerge l'utilisateur dans un monde synthétique ; tout ce que l'utilisateur perçoit (vue, ouïe, et idéalement toucher) est produit par le système et inversement toutes ses actions (actions physiques comme parole) sont interprétées par le système. Issue des travaux militaires dans le domaine des simulateurs de vols, la réalité virtuelle trouve aujourd'hui des applications dans des domaines très spécialisés : médecine, télé-opération en milieu hostile, etc.

La réalité augmentée procède d'une approche opposée à la réalité virtuelle : au lieu d'immerger l'utilisateur dans un monde synthétique, on intègre l'interface du système informatique dans les objets et l'environnement quotidiens. Par exemple, une feuille de papier peut servir d'interface ; une caméra ou une tablette graphique permet de capturer ce que l'on écrit dessus, et un projecteur permet d'afficher des informations sur la feuille. La frontière entre les mondes physique et informatique s'efface, rendant l'interface invisible.

L'interaction multimodale cherche à exploiter, aux mieux, les capacités d'action et de perception de l'être humain, en s'intéressant notamment aux combinaisons entre modes d'interaction. Par exemple, en entrée la combinaison du geste et de la parole est plus puissante que chacune des modalités prises indépendamment ; je peux dire « mets ça ici » en désignant du doigt le « ça » et le « ici » alors qu'il est difficile d'exprimer la même commande seulement par la voix ou par le geste. En sortie, la combinaison de l'image et du son permet par exemple d'attirer l'attention (grâce au son) et de communiquer une information sous forme graphique.

Le collecticiel est une extension de la notion de système interactif dans laquelle plusieurs utilisateurs interagissent avec le système afin de communiquer entre eux via le système. Un exemple d'application *collecticiel* est un éditeur partagé qui permet à plusieurs personnes de modifier, simultanément, le même document, et de voir, en temps réel, les modifications effectuées par les autres personnes. L'intérêt croissant pour le collecticiel vient de la constatation que la plupart de nos activités sont des activités de groupe, alors que l'usage de l'ordinateur est essentiellement individuel.

Chapitre 3 : Conception de Systèmes Interactifs

1. Introduction

Une application interactive ou système interactif est un logiciel qui communique avec un utilisateur (*Fig.3.1*). Les échanges s'effectuent grâce à une interface homme-machine (IHM), qui prend en charge :

- La communication de l'utilisateur humain vers le noyau fonctionnel de l'application (entrée d'information).
- La communication du noyau fonctionnel vers l'utilisateur humain (sortie d'information).
- La traduction entre les variables informatiques du monde de l'application et celles psychologiques et/ou réelles de l'utilisateur.

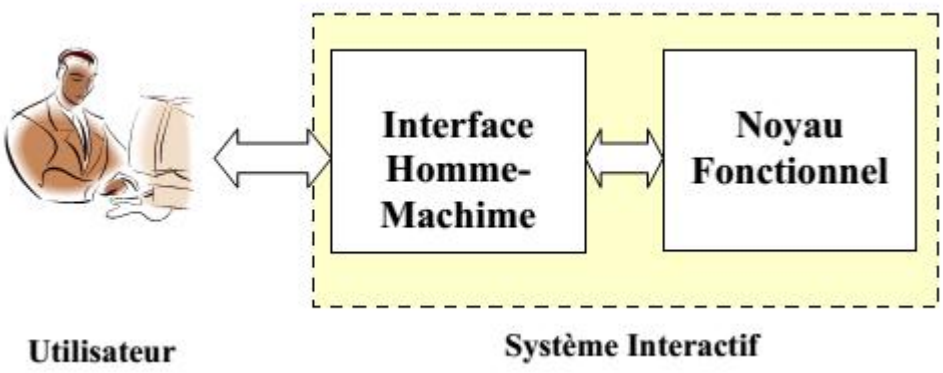

Fig. 3.1 Vue générique d'un Système Interactif [4].

La qualité d'une IHM est primordiale car plus la distance entre les variables du monde de l'application et celles de l'utilisateur est grande, plus le risque d'erreurs et d'incompréhension est grand [5]. Les concepteurs d'IHM cherchent donc à produire une représentation perceptible du système qui soit la plus proche possible de la représentation mentale de l'utilisateur. Très tôt, les ergonomes ont aussi montré que qu'il était important de tenir compte des différences interindividuelles. Ainsi, l'adaptabilité est, avec la compatibilité, un critère utilisé par l'ergonomie des logiciels pour apprécier l'adéquation entre les exigences de l'activité, les ressources de l'usager et les caractéristiques du système [6], [7].

Parce que les IHM impliquent l'humain, elles obligent les chercheurs à adopter un regard pluridisciplinaire mêlant l'ergonomie, l'automatique, l'informatique, la psychologie ou la sociologie. La tendance consiste ainsi à mettre en relation les spécificités de ces différentes disciplines pour aboutir à des modèles, des méthodes et des outils qui couvrent les différents aspects de l'IHM. L'analyse de la littérature met en évidence plusieurs types de modèles mettant l'accent sur différents aspects de la conception des interfaces et faisant intervenir plusieurs acteurs dans le processus de construction des systèmes interactifs [4] :

- *Les modèles de tâches* : destinés au concepteur d'application, auquel ils fournissent un schéma général de l'activité de l'utilisateur du système.
- *Les modèles de dialogue* : destinés au concepteur de dialogue pour décrire la structure des échanges entre l'homme et l'application. Ils identifient les objets du dialogue.
- *Les modèles d'architecture* : ils fournissent, aux réalisateurs, une structure générique de l'application à partir de laquelle il est possible de construire un système interactif particulier.

Dans ce chapitre, nous parcourons les modèles, les architectures, ainsi que les techniques et les outils employés pour la conception et le développement des systèmes interactifs.

2. Les modèles de tâches

Le concept de tâche peut être défini comme « une activité dont l'accomplissement par un utilisateur produit un changement d'état significatif d'un domaine d'activité donné dans un contexte donné » [8]. Les modèles d'analyse de tâches sont utilisés pour analyser et décrire une tâche particulière de l'utilisateur, le plus souvent par décomposition hiérarchique en sous-tâches. Ils sont typiquement utilisés pour fournir un cadre de base au processus de conception d'une IHM particulière. Ils identifient les types de connaissances que l'utilisateur a ou doit avoir pour réaliser cette tâche.

La modélisation des tâches a bénéficié de la contribution de plusieurs domaines de recherche :

- Les chercheurs en sciences cognitives proposent des modèles pour l'identification et la caractérisation des tâches par exemple *GTA* ou *MAD*.

- Les informaticiens proposent des approches et des notations pour la représentation des tâches et de leurs relations telles que *UAN* et *CTT*.

Etant donné le grand nombre de propositions de modèles de tâches, différentes classifications ont été élaborées. Certains classent les modèles de tâches dans trois catégories : les modèles linguistiques, les modèles hiérarchiques orientés tâches et les modèles de connaissance. Certains effectuent leur classement selon le type de formalisme utilisé pour décrire le modèle, l'information qu'il contient et le support qu'il fournit. Certaines classifications distinguent but, discipline, relations conceptuelles et pouvoir d'expression.

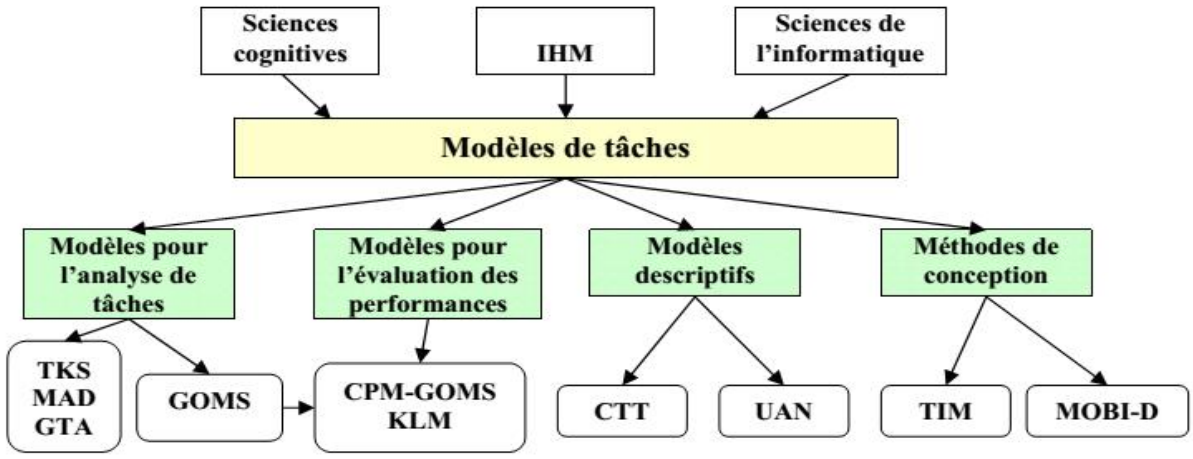

Fig.3.2 Différentes approches de la modélisation des tâches [9].

✓ **GOMS (GOAL, OPERATOR, METHOD, SELECTOR)**

Le modèle *GOMS* [10] est issu du domaine de la *Résolution de Problèmes*. *GOMS* propose une description des activités de l'usager sous forme de *buts* (*Goal*), d'*opérateurs* (*Operator*), de *méthodes* (*Method*) et de *règles de sélection* (*Selector*). Un **but** est un état du système à atteindre, un **opérateur** est une opération élémentaire, une **méthode** est un algorithme utilisable pour atteindre un but, et une **règle de sélection** décrit comment choisir une méthode parmi plusieurs susceptibles d'atteindre le même but.

Il existe différentes versions de *GOMS* :

• *KLM* (*Keystroke-Level Model*) [11] (un modèle au niveau des touches) : est une version simplifiée de *GOMS*, qui n'utilise que les opérateurs concernant les touches, sans utiliser de buts, de méthodes ou de règles de sélection. Ce modèle ne fait que dresser la liste des

touches, des mouvements de souris et des clics souris accomplis pour effectuer une tâche, afin d'évaluer le temps nécessaire à son accomplissement.

- *NGOMS* [12] : est une extension de *GOMS* dans laquelle ont été ajoutés : (1) la possibilité d'identifier les composants *GOMS*, (2) des informations comme le nombre d'étapes dans une méthode, (3) la façon dont un but est atteint, (4) les informations à retenir au cours de l'accomplissement d'une tâche.
- *CPM-GOMS* [13] : ajoute (partiellement) à *GOMS* la prise en compte des erreurs et la concurrence des tâches, implémentant une méthode de description de chemin critique, basée sur les diagrammes *PERT*.

GOMS travaille par niveau de modélisation, c'est-à-dire en considérant des opérateurs ayant des temps d'exécution du même ordre de grandeur. Ce modèle peut, ensuite, être affiné en considérant que les buts composés de sous-buts atteints par des opérateurs ayant des temps d'exécution inférieurs. *GOMS* introduit donc la notion classique d'analyse descendante et ascendante, ainsi que l'analyse prédictive du temps d'exécution d'une tâche. Il fait partie des modèles hiérarchiques de tâches. En revanche, *GOMS* manque d'opérateurs pour décrire explicitement l'activation et la désactivation dynamique des tâches, ce qui limite grandement son usage surtout dans le contexte d'interfaces avancées.

✓ **UAN (User Action Notation)**

La technique de spécification *UAN* [14] est un modèle linguistique pour la description de l'*IHM*. Il repose sur une notation textuelle orientée tâche et centrée sur l'utilisateur, qui peut ainsi décrire une tâche dans un tableau à trois colonnes :

- *Les actions utilisateur* : décrivant les actions physiques exécutées par l'utilisateur quand celui-ci agit sur les dispositifs (enfoncer le bouton gauche de la souris par exemple).
- *Les retours d'information* : fournis par le système (élément sélectionné).
- *L'état de l'interface :* qui donne la nouvelle situation de l'interface (élément sélectionné).

L'exemple suivant (*Fig.3.3*) décrit la tâche de déplacement d'une icône de fichier dans une interface.

Tâche : Sélectionner Fichier		
Actions Utilisateur	Retour d'information	Etat de l'interface
~ [File icon] Mv [File icon] M^	[File icon]'!	selected=file

Fig.3.3 Sélectionner une icône de fichier exprimé avec UAN [4].

Dans cet exemple [4], « ~[File icon] »signifie le déplacement du curseur sur une icône de fichier non sélectionnée. « Mv » signifie presser le bouton de la souris. Quand ces deux actions sont réalisées, l'interface doit faire apparaître en surbrillance l'icône de fichier « [file icon]' ! » et positionner la variable « selected » à la valeur « file ». « M^ » signifie relâcher le bouton de la souris et complète la tâche.

La capacité de description de la version initiale de *UAN* était limitée au niveau de la tâche élémentaire. Une extension *XUAN* fut proposée par *Gray*. Dans *XUAN*, la structuration des tâches est plus complète avec des variables locales et externes à la tâche, des pré- et post-conditions et quelques notions temporelles plus fines.

La structuration du modèle de tâches proposé par *UAN/XUAN,* sous forme de tableau, donne une certaine compréhension de ce modèle. Cependant, une représentation graphique hiérarchisée offre une meilleure lisibilité des modèles de tâches.

- ✓ **CTT (ConcurTaskTrees)**

CTT [15] est une notation basée sur des diagrammes hiérarchiques pour la description des modèles de tâches. Contrairement aux approches précédentes, comme *UAN*, *CTT* offre un nombre important d'opérateurs, qui ont chacun un sens précis issu principalement de la notation *LOTOS* (*Language Of Temporal Ordering Specification*). Ces opérateurs servent à décrire de nombreuses relations temporelles (la concurrence, l'interruption, la désactivation, l'itération, etc). Le concepteur peut ainsi représenter de façon concise les évolutions possibles d'une session utilisateur.

La *Fig. 3.4* montre la notation des opérateurs pouvant lier deux tâches *T1* et *T2*. La priorité des opérateurs est, dans un ordre décroissant : opérateur unaire, [], |=|, |||, |[]|, [>, |>, >>, []>>.

Notation	Signification
T1 [] T2	Le choix
T1|=|T2	Ordre indépendant
T1|||T2	Imbrication (interleaving)
T1|[]|T2	Synchronisation
T1>>T2	Composition séquentielle
T1[]>>T2	Composition séquentielle avec transfert d'information
T1[>T2	Neutralisation (disabling)
T1*	Itération infinie (opérateur unaire)
[T1]	Exécution optionnelle (opérateur unaire)
T1|>T2	Suspendre/Reprendre

Fig.3.4 Les notations CTT des opérateurs entre deux tâches T1 et T2 [4].

CTT possède une notation graphique relativement intuitive supportée par un outil de modélisation appelé *CTTE* (*ConcurTaskTrees Environment*) qui rend l'approche facilement utilisable. Il présente des fonctionnalités avancées comme la scénarisation et l'ordonnancement des tâches en des ensembles de tâches de présentation *PTS* (*Presentation Task Sets*).

La *Fig.5* représente une modélisation *CTT* d'une tâche de consultation des informations concernant un étudiant. Cette tâche se divise en plusieurs sous-tâches utilisateurs permettant la formalisation de la demande (*saisir_paramètres*, *envoyer_demande*) et d'une tâche machine qui assure l'affichage des résultats.

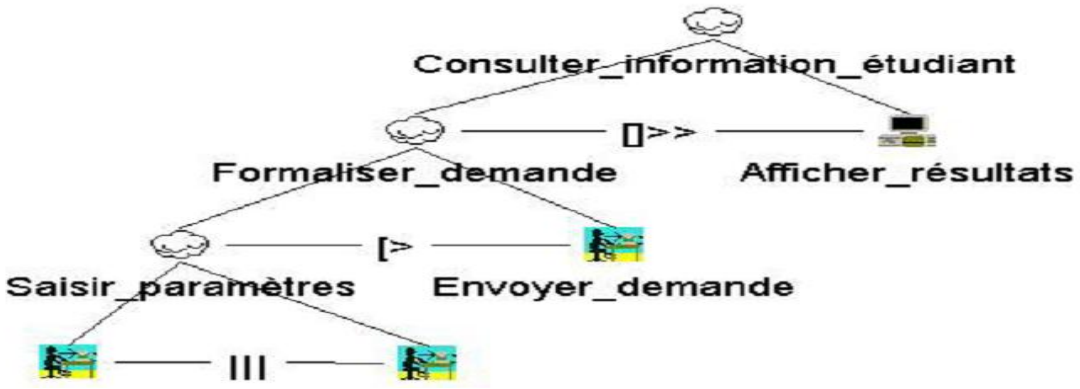

Fig. 3.5 La modélisation CTT d'une tâche de consultation des informations concernant un étudiant (adoption d'un exemple fourni avec l'environnement CTTE) [4].

Bien que la notation *CTT* ne soit pas une notation formelle, appuyée par son éditeur graphique (téléchargeable gratuitement), elle s'est bien répandue en particulier dans le milieu universitaire.

3. Les formalismes du modèle de dialogue

Green [16] définit un modèle de dialogue comme un modèle abstrait utilisé pour décrire la structure logique du dialogue entre un utilisateur et un système interactif. Pour exprimer le modèle de dialogue, plusieurs approches et formalismes sont utilisés. Ces modèles et formalismes sont issus de domaines variés tels que *la théorie des langages* ou *la théorie des graphes*.

Il existe un grand nombre de notations utilisées pour décrire des comportements dits « *orientés contrôle* ». Ces notations graphiques et leurs variantes sont couramment employées en interaction *homme-machine*. Le modèle de dialogue exprimé avec un système de transition décrit un état spécifique du système qui autorise une séquence d'états pour l'utilisateur.

- ✓ **Les automates à états finis**

Les automates à états finis [4] ont été employés pour spécifier le comportement dynamique de certains systèmes, et sont notamment utilisés pour décrire certaines parties des interfaces utilisateur.

Les automates à états finis sont représentés graphiquement par des diagrammes de transition d'états où les nœuds sont des états. Ces diagrammes manipulent cinq éléments : un ensemble fini d'états, un ensemble fini d'entrées, un ensemble fini de sorties, une fonction pour passer à l'état suivant (dépend de l'état précédent et de l'entrée), et une fonction de sortie pour envoyer les résultats. Un **état** est un ensemble de valeurs qui caractérise le système à un instant donné. Une **transition d'état** est une relation indiquant un changement possible entre deux états.

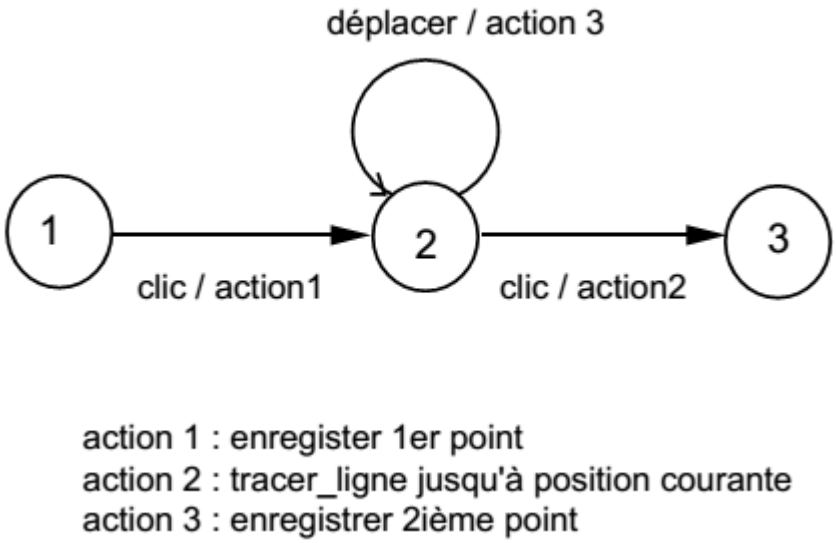

Fig.3.6 Diagramme de transition d'états décrivant le tracé d'un segment avec une souris [4].

Les automates à états finis ont rapidement montré leurs limites pour modéliser des systèmes complexes ou parallèles. Des extensions telles que *les machines à états hiérarchiques* et *les state charts* ont cherché à y trouver remède.

- ✓ **Les réseaux de Petri**

Les réseaux de Petri [17] sont une généralisation des automates à états. Ils ont été largement utilisés pour modéliser la concurrence et la synchronisation dans les systèmes distribués. *Un réseau de Petri* se compose de places, représentés par des cercles, et de transitions, représentées par des barres horizontales. Les places et les transitions sont connectées à l'aide de flèches orientées. *Un réseau de Petri* est donc un graphe orienté bipartie qui possède deux types de nœuds : les places (cercles) et les transitions (rectangles). Toute place est connectée à au moins une transition et inversement. Chaque place possède un nombre de jetons. Une transition est franchissable sous certaines conditions, notamment lorsque suffisamment de jetons sont présents dans ses places d'entrée. Le franchissement d'une transition se traduit par une modification du marquage consistant la plupart du temps en un déplacement de jetons.

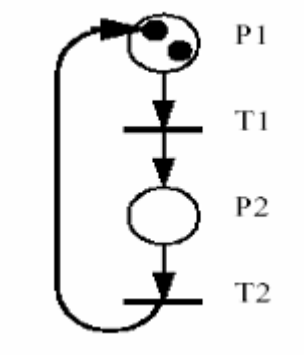

Fig. 3.7 Un réseau de Petri discret [4].

Les réseaux de Petri ont connu de nombreuses extensions, comme *les réseaux de Petri colorés* [18], *temporisés* [19], et *hybrides* [20].

Le modèle de dialogue est insuffisant pour décrire un système interactif [4]. Cependant, ce modèle constitue souvent un composant fondamental dans des méthodes plus globales de description des systèmes interactifs. Notons que certains travaux proposent de faire cohabiter les formalismes de dialogue et les approches orientées objet pour la description des systèmes interactifs.

- ✓ **Les objets coopératifs interactifs (ICO : Interactive Cooperative Object)[8]**

Les *ICO* sont *des réseaux de Petri* à objets, dédiés à la spécification formelle des systèmes interactifs. Dans ce formalisme, un système est composé d'objets dont on décrit le comportement et les communications à l'aide de *réseaux de Petri*. Chaque objet est lui-même composé de quatre parties : comportement, services, état et présentation. Le **comportement** d'un objet est sa façon de réagir aux stimuli extérieurs en fonction de son état interne. Ce comportement est décrit à l'aide d'*un réseau de Petri* de haut niveau appelé *structure de contrôle de l'objet* (*ObCS* : *Object Control Structure*). Les **services** composent l'interface offerte par l'objet à son environnement, c'est-à-dire l'utilisateur, ou à d'autres objets via un dispositif d'entrée. L'**état** d'un *ICO* est représenté par le marquage des places de l'*ObCS*. Enfin, la **présentation** est l'apparence extérieure de l'objet.

Ce formalisme est supporté par un environnement appelé *PetShop* qui est un éditeur/interpréteur d'*ICO* servant à spécifier le modèle du domaine à travers un ensemble de classes d'objets coopératifs appelé les *CO-classes*.

Dans le même esprit, la méthode *TOOD* utilise conjointement l'approche objet et *les réseaux de Petri* dans un modèle générique couplé avec un modèle de tâches. L'idée de base dans cette approche hybride consiste à utiliser *les réseaux de Petri* à objets pour décrire le comportement interne des objets et les communications qu'ils entretiennent, pour pouvoir exprimer la concurrence aussi bien entre les différents objets qu'au sein même d'un objet.

4. Les modèles d'architecture

Les modèles d'architecture logicielle constituent un des éléments indispensables sur lequel s'appuie le génie logiciel moderne. Il systématise, à la fois, l'élaboration et l'utilisation des logiciels et joue un rôle primordial dans leur qualité et leur efficacité. *Garlan* et *Perry* soulignent le fait que l'utilisation des modèles d'architecture peut avoir des effets bénéfiques sur au moins cinq aspects du développement des logiciels [21]:

- La **compréhension** est simplifiée en présentant la structure de grands systèmes à un niveau d'abstraction adapté. Les contraintes et choix de conception peuvent ainsi être mieux explicités et expliqués.
- La **réutilisation** est favorisée. Bien sûr, la réutilisation des composants est largement encouragée par la définition d'architectures, mais il est également possible de réutiliser des structures plus complexes définies sous forme de *motifs de conception* ou *design patterns*.
- L'**évolution** et la **maintenance** profitent largement de la définition d'architectures logicielles. En comprenant les ramifications et les connexions de chaque composant, il est beaucoup plus facile de déterminer, a priori, les coûts potentiels de toute modification.
- L'**analyse** des systèmes est simplifiée. Suivant les modèles utilisés, on peut envisager des formes de vérification de cohérence, de respect de styles, de satisfaction de critères de qualité voire d'adaptation à des domaines spécifiques d'application.
- La **gestion** des projets industriels doit s'appuyer sur cet élément essentiel pour déterminer au plus tôt la viabilité des options choisies, les pré-requis indispensables et les perspectives d'évolution.

On peut spécifier quatre objectifs pour le modèle d'architecture :

- Servir de support lors de la spécification (en tant que formalisme).

- Constituer l'ossature de la réalisation (en tant que *framework*).
- Assurer la cohérence de fonctionnement à l'exécution (au *runtime*).
- Servir de guide d'utilisation et/ou de support à la reconfiguration par l'utilisateur.

Les modèles d'architecture partent du principe qu'un système interactif comporte une partie **interface** et une partie **noyau fonctionnel** qui se réfère au domaine de l'application. Le noyau fonctionnel est considéré comme préexistant, et les modèles des systèmes interactifs décrivent essentiellement la partie interface, ainsi que ses relations avec les objets du domaine.

La plupart des modèles identifient au moins trois types d'éléments.

- Les éléments en contact direct avec l'utilisateur (présentations).
- Les éléments en contact direct avec le noyau fonctionnel ou qui en font partie (interfaces du noyau fonctionnel, abstractions, etc.).
- Les éléments de communication entre les deux premiers éléments (contrôleurs, adaptateurs).

Malgré ces points communs, certains modèles procèdent d'approches différentes, et ne se limitent pas à ce découpage minimal.

Les modèles d'architecture des applications interactives peuvent être classifiés en deux grandes catégories. Les **modèles à couche** décrivent la structure globale d'une application interactive sous forme de couches logiques. Ces modèles sont généralement appelés modèles logiques, ou sont qualifiés d'abstraits. La seconde catégorie contient les **modèles à agents**, ou encore modèles orientés objet. Ces modèles proposent une décomposition modulaire de l'interface en un ensemble d'agents communicants. Par ailleurs, des modèles mixtes tels que *PAC-Amodeus* [22] ou *H4* [23] tentent de tirer partie des avantages respectifs de ces deux approches en les combinant.

✓ **Les modèles d'architecture en couches**

- ***Modèle de Seeheim***

Le premier modèle architectural de référence pour les applications interactives a été proposé lors d'un *workshop* à *Seeheim*[24]. Il distingue trois composants logiques (*Fig. 3.8*) :

- Le **Noyau Fonctionnel** regroupe les concepts et fonctions du domaine.
- La **Présentation** est chargée de rendre perceptible l'état pertinent des concepts du domaine et d'en supporter la manipulation par l'utilisateur.
- Le **Contrôleur de Dialogue** gère les communications entre le Noyau Fonctionnel et la Présentation.

Le Noyau Fonctionnel et la Présentation définissent deux perspectives d'un même domaine, mais chaque perspective possède un rôle précis à remplir. Dans le Noyau Fonctionnel, la modélisation du domaine est guidée par des considérations de calcul alors que dans la présentation, le modèle est conçu pour manipuler et rendre perceptible l'état du Noyau Fonctionnel.

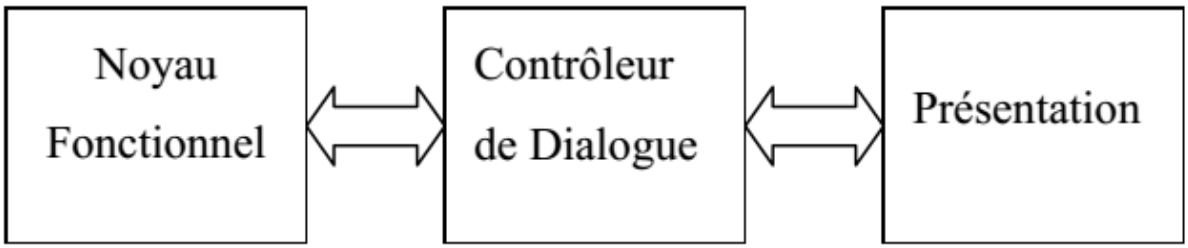

Fig.3.8 Le modèle de Seeheim [4].

En conséquence, le Noyau Fonctionnel et la Présentation utilisent des formalismes et des techniques de représentation différents. Le pont entre les deux mondes est assuré par le Contrôleur de Dialogue au moyen de deux protocoles de communication, l'un adapté aux besoins du Noyau Fonctionnel et l'autre à ceux de la Présentation.

- ***Le modèle Arch***

Le modèle Arch [25] s'appuie sur les composants conceptuels du *modèle Seeheim* en rajoutant deux composants intermédiaires. Il reprend *le contrôleur de dialogue de Seeheim* comme centre de l'arche et il raffine la décomposition des deux autres piliers. (*Fig.3.9*).

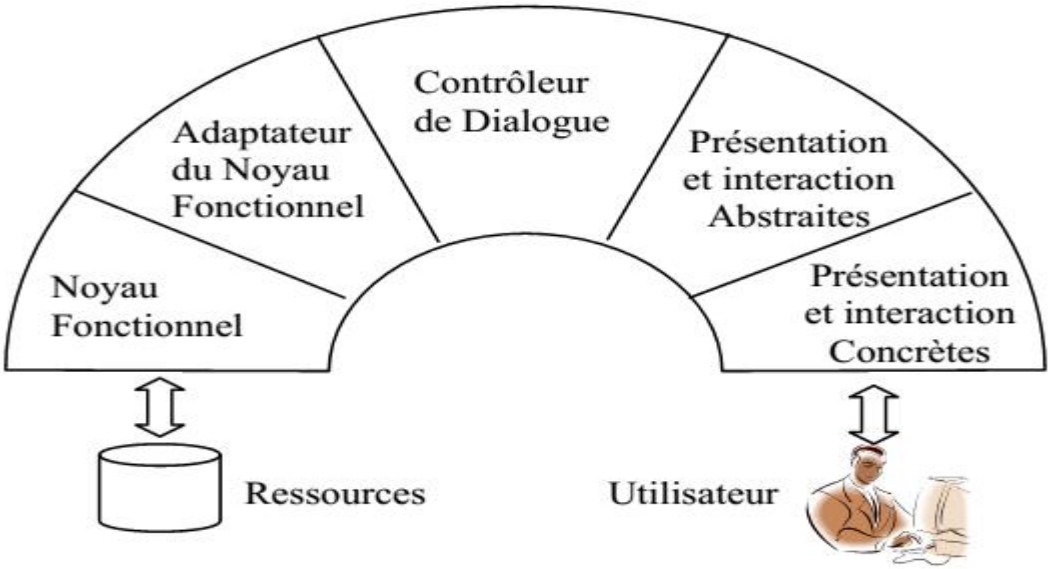

Fig.3.9 Les composants du modèle Arch [4].

Du côté du pilier applicatif, il distingue 2 niveaux :

- Le **noyau fonctionnel** de l'application qui gère les données internes et leur rémanence.
- L'**adaptateur du noyau fonctionnel** est un composant médiateur entre le contrôleur de dialogue et le noyau fonctionnel. Il est responsable de la réorganisation des données du domaine dans des buts de manipulation interactive, ou de la détection des erreurs sémantiques.

Du coté du pilier présentation, il décompose l'utilisateur en 2 niveaux :

- Un **niveau abstrait**, comportant les objets de présentation et d'interaction indépendant de l'interface concrète et donc des dispositifs d'interaction avec l'utilisateur.
- Un **niveau concret**, lié à une *Toolkit* précise gérant le dialogue avec l'utilisateur (et donc lié à des dispositifs d'entrée/sortie précis).

✓ **Les modèles d'architecture multi-agents**

Ferber a définit l'*agent* comme une entité informatique, réelle ou abstraite, indépendante à durée de vie limitée, capable d'agir sur elle-même et son environnement [26]. *Un agent* communique avec ses pairs et son comportement est la conséquence de ses observations, de sa connaissance et de ses interactions avec les autres agents.

En réalité, au sein de la communauté informatique, il existe deux approches assez différentes des univers multi-agents. Ainsi, les chercheurs qui travaillent dans le domaine de l'intelligence artificielle construisent des mondes composés d'**agents cognitifs**. Chaque agent possède des connaissances sur un domaine particulier, et la résolution d'un problème complexe se traduit par la coopération de ces agents (et éventuellement celle de l'utilisateur). Les agents cognitifs sont généralement des entités de taille assez importante. A l'opposé, les chercheurs spécialisés dans la modélisation des interfaces homme-machine considèrent les applications interactives comme étant des mondes composés d'**agents réactifs**. Un agent réactif ou interactif est alors défini comme un agent dont le comportement est la conséquence de ses observations et de ses interactions avec l'utilisateur. A la différence des agents cognitifs, la granularité des agents réactifs peut être très petite (un agent réactif peut être réduit à un simple bouton).

- ***MVC***

Le modèle MVC (*Modèle*, *Vue*, *Contrôleur*) [27] a été introduit comme architecture de référence dans l'implémentation des interfaces utilisateur du langage *Smalltalk*. L'approche de *MVC* inspirera toute une lignée de modèles à base d'agents, dont les principales motivations sont la **modifiabilité** et la **conception itérative**, ainsi que la **compatibilité avec les langages à objets**.

Dans cette architecture, trois types d'objets collaborent :

- Le **Modèle** est l'objet sémantique que l'on veut visualiser et modifier.
- La **Vue** est l'objet graphique, responsable de la visualisation et de l'édition graphique d'une facette du modèle.
- Le **contrôleur** prend en charge la gestion des évènements de synchronisation pour mettre à jour la vue ou le modèle. Il n'effectue aucun traitement, il analyse la requête du client et se contente d'appeler le modèle adéquat et de renvoyer la vue correspondant à la demande.

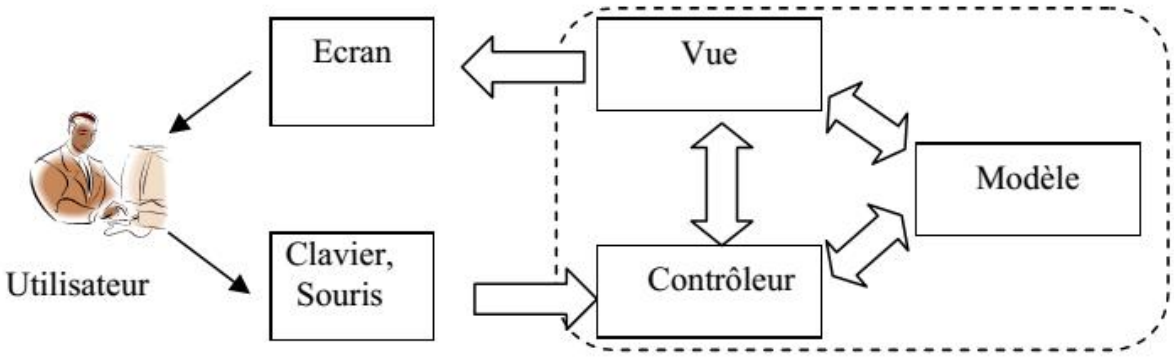

Fig.3.10 Le modèle MVC [4].

- ***PAC***

Le modèle PAC [28] décompose l'application interactive en une hiérarchie d'agents interactifs (*Fig.3.11*) structurés en trois facettes :

  - Le composant **Présentation** est chargé de la transmission (visuelle, sonore, etc.) des informations à l'utilisateur et de l'acquisition des entrées de l'utilisateur.
  - Le composant **Abstraction** contient les données et les traitements sémantiques.
  - Le composant **Contrôle** est le moteur des dialogues qui s'établissent entre l'agent et l'utilisateur, et entre l'agent et les autres agents.

PAC gère un dialogue multi-fils (plusieurs dialogues en parallèle) grâce à l'autonomie relative des agents dans l'interprétation des évènements et dans la gestion de leur état.

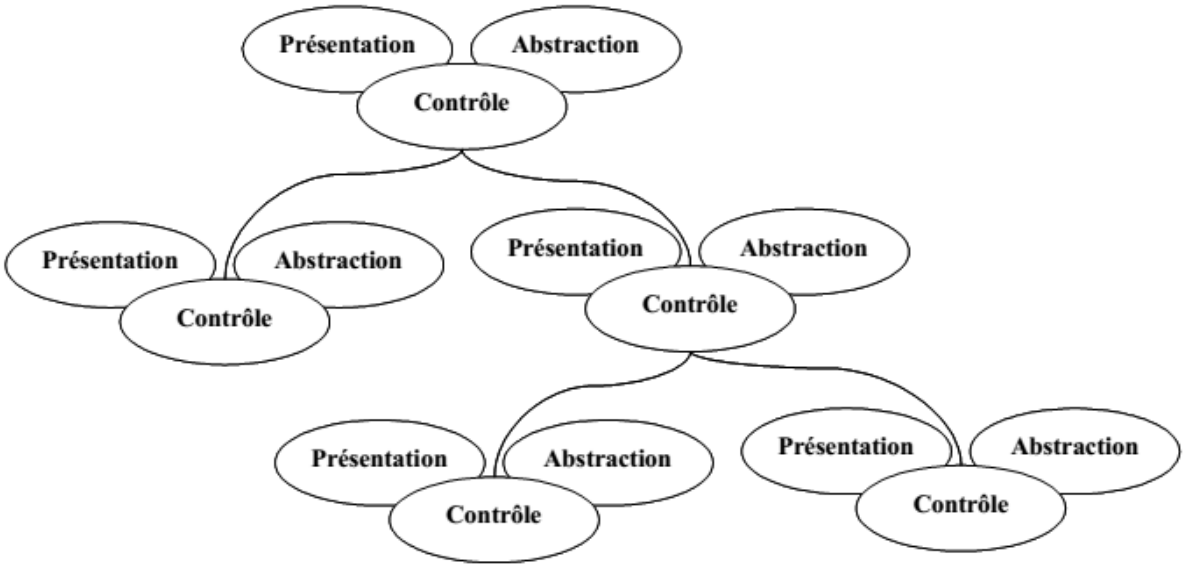

Fig.3.11 Structure d'une application interactive PAC [4].

- ***Le modèle AMF basique***

AMF [29] est un modèle d'architecture multi-agents et multi-facettes utilisé pour spécifier l'architecture logicielle d'une application interactive. *Le modèle AMF* propose une décomposition des agents interactifs plus fine que *PAC*. Chaque **agent** est structuré en facette. Sur le plan conceptuel, une **facette** est un composant contenant l'ensemble des données et traitements relevant d'une même thématique fonctionnelle. Leur nombre n'est pas limité et aucune facette n'est obligatoire (lorsque la granularité de décomposition des agents est très fine, certains agents peuvent ne contenir qu'une seule facette spécialisée).

En général, parmi les facettes des *agents AMF*, on retrouve les composants classiques de présentation et d'abstraction complétés par de nouvelles facettes pouvant provenir de :

- L'identification de fonctions spécifiques des agents comme la gestion du modèle de l'utilisateur ou la communication dans un contexte multiutilisateurs.
- Une duplication des facettes classiques.
- Une décomposition des facettes Contrôle des autres modèles qui sont souvent des « *fourre-tout* » contenant tout ce qui ne relève pas des autres composants.

Les agents sont organisés hiérarchiquement selon des règles de composition qui traduisent le fait qu'un agent parent contient un certain nombre d'agents enfant. De fait, pour toute application modélisée avec *AMF*, il existe un agent racine ancêtre commun de tous les agents de l'application.

Pour représenter les concepts et les mécanismes de contrôle dans *une architecture AMF*, le modèle repose sur un formalisme s'appuyant sur des éléments situés à deux niveaux. Au niveau des facettes, les **ports** de communication définissent les services proposés et les services utilisés. Au niveau des agents, les composants de contrôle sont définis par le regroupement d'entités de contrôle élémentaires appelées **administrateurs de contrôle**.

Les ports de communication constituent les interfaces des facettes. A ce titre, ils expriment non seulement les services proposés par chaque facette, mais aussi les services indispensables à leur fonctionnement. En fait, on distingue trois types de ports (*Fig.3.12*) :

- Port d'**entrée** (service offert).
- Port de **sortie** (service nécessaire).

- Port d'**entrée/sortie** (service offert, dont l'activation se conclut par l'invocation d'un autre service distant créant ainsi un phénomène de diffusion).

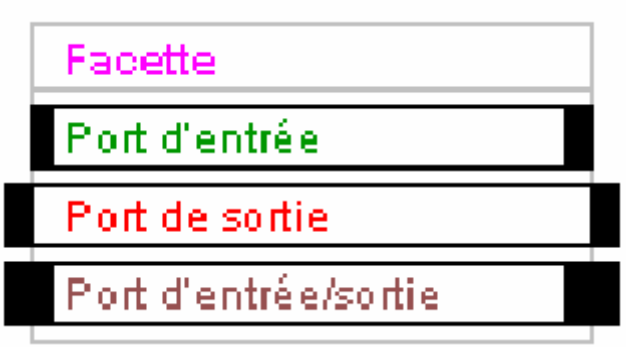

Fig.3.12 Représentation des ports de communication d'une facette AMF [4].

A chaque port de communication est associée une fonction particulière appelée **démon**. Ce démon se déclenche automatiquement à chaque activation du port.

Les administrateurs de contrôle des agents gèrent les liens entre les ports de communication des facettes. Un administrateur de contrôle joue trois rôles :

- Un rôle de connexion qui consiste à gérer les relations logiques pouvant exister entre les ports de communication qui lui sont attachés.
- Un rôle comportemental qui exprime les règles d'activation de l'administrateur, c'est-à-dire sous quelle condition et à quel moment les messages émis par les ports sources seront transmis aux ports cibles.
- Un rôle de traduction qui consiste à transformer les messages émis par les ports sources en messages compréhensibles par les ports cibles.

La *Fig.3.13* illustre le formalisme graphique de *l'architecture AMF* [4]. Elle modélise une relation élémentaire entre deux ports de deux facettes de l'agent *A1*. Cette relation se traduit par l'utilisation d'un administrateur simple entre le port source et le port cible.

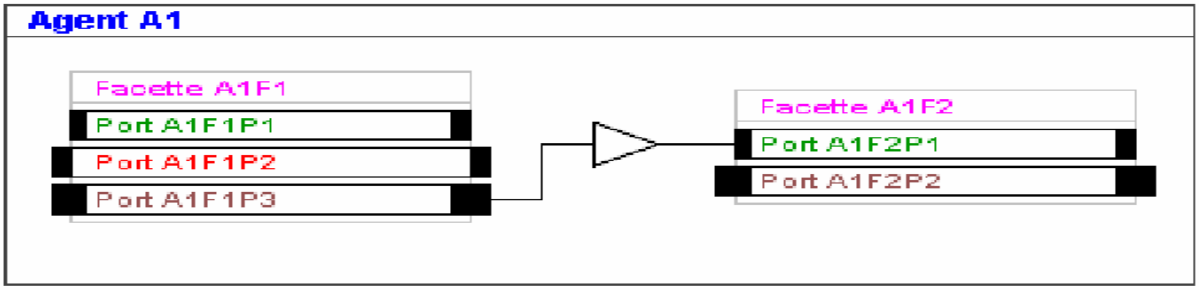

Fig.3.13 Modélisation des éléments de base de l'architecture AMF [4].

- ✓ **Les modèles d'architecture mixte**

  - ***PAC-Amodeus***

Dans certains cas, il apparaît souhaitable de combiner les modèles à couches avec les modèles multi-agents pour essayer de tirer partie de leurs avantages respectifs. Ainsi, pour la modélisation des applications multimodales, *les modèles PAC-Amodeus* [22] reprennent le découpage en couches de *Arch* tout en exprimant le composant contrôleur de dialogue sous la forme d'*agents PAC*. Il peut ainsi facilement exprimer le parallélisme de contrôle des différentes modalités.

D'une façon plus générale, les composants **Abstraction** et **Présentation** du *modèle PAC* peuvent être rapprochés des composants Adaptateur de domaine et Présentation du *modèle Arch*. Dans ce cas, ils constituent des adaptateurs combinant une modélisation multi-agents de l'application et l'utilisation de boîtes à outils spécialisés (pour le traitement du noyau fonctionnel ou pour l'interface utilisateur). Ainsi, il est possible de concevoir des logiciels hétérogènes utilisant, à la fois, l'approche multi-agents et l'approche en couches et faisant appel à différents langages de programmation ou boîtes à outils.

  - ***Le modèle AMF hybride***

Le but d'*AMF* est de servir, à la fois, à la conception, la réalisation et l'utilisation. La dualité des vues, entre la segmentation thématique en couches (présentation, abstraction, etc.) et le rapprochement de tous les aspects concernant le même agent nous conduisent à privilégier la vue « *agent* » dans la conception et la vue « *en couches* » dans la réalisation. Pour marier ces deux approches, une extension du *modèle AMF* a été proposée. Ce modèle est appelé : **AMF hybride** [30].

Ainsi, les agents d'une application peuvent être considérés comme des entités duales : une partie *AMF* pour les connexions et les communications gérant la dynamique d'interaction, et une partie application, pour la gestion des objets interactifs de la présentation ainsi que les données et les traitements applicatifs. On retrouve ainsi les 5 niveaux du modèle *Arch*, comme représenté sur la *Fig.3.14*.

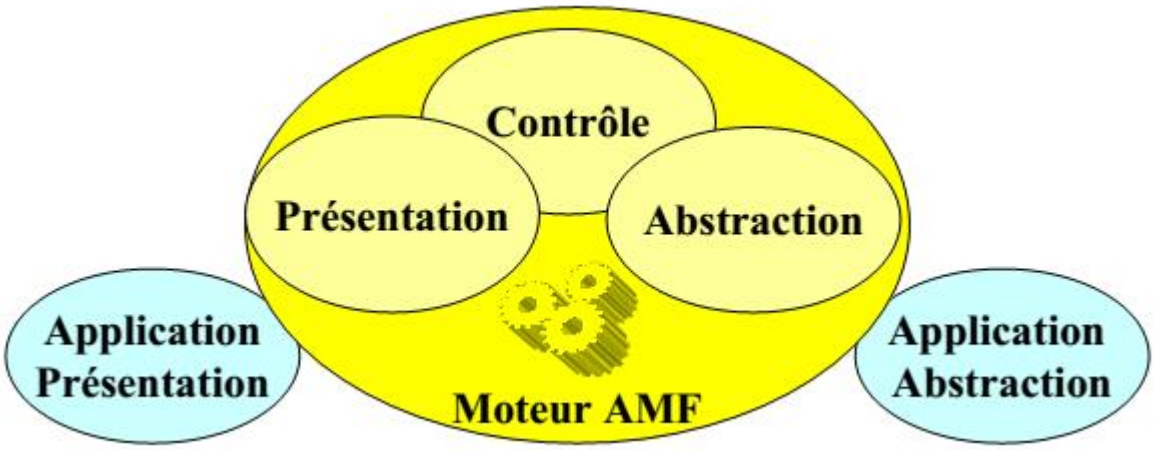

Fig.3.14 Dualité des objets de l'application [4].

Toute la modélisation *AMF* (*agents*, *facettes*, *ports* et *administrateurs*) et les échanges dynamiques de messages (communications inter et intra agents) est géré par ce que nous appelons le **Moteur AMF**. Ce rapprochement d'*AMF* avec *Arch* permet d'aller au-delà d'un simple modèle d'architecture. En effet, l*e modèle AMF* est directement instancié en un contrôleur de dialogue et *le moteur AMF* pilote le dialogue de l'application en gérant les échanges entre facettes selon le contrôle exprimé dans *le modèle AMF*.

Pour qu'une application puisse être exécutée, les objets de l'application (issus soit de classes de présentation gérant les objets graphiques (*Widgets*) soit du noyau fonctionnel) doivent être liés à leurs correspondants *AMF* par une relation d'association. C'est la raison pour laquelle chaque agent du *modèle AMF*, ainsi que chacune des facettes a une entité duale : une partie du côté application, une autre complémentaire du côté *moteur AMF*.

5. Techniques et outils pour la génération de systèmes interactifs

Pour que la charge de travail des développeurs des systèmes interactifs soit moins importante, un grand nombre de techniques et d'outils d'aide au développement ont été proposés.

Dans cette section, nous présentons, tout d'abord, la notion de patterns, puis, nous explorons les deux grandes familles d'outils utilisées pour la conception d'interfaces [28], [22], [4] à savoir : (1) les **approches ascendantes** qui produisent automatiquement le code d'une interface à partir d'une spécification externe de l'interface, et (2) les **approches**

descendantes qui partent de la description des concepts et des tâches pour aller jusqu'aux objets de présentation et au code exécutable de l'interface.

✓ **Les patterns**

L'origine des patterns remonte à l'architecte en bâtiment *C. Alexander* en *1977* [31]. Celui-ci a réuni, dans un catalogue, la description d'un ensemble de problèmes types pouvant avoir lieu lors de la conception d'un édifice, en associant à chaque problème les éléments d'une solution générale permettant de le résoudre. *Alexander* nomme ce catalogue un langage de patterns (*a Pattern Language*), dans lequel tous les couples (*problème-solution*) sont des patterns.

Le concept de patterns a, ensuite, envahi de nombreux autres domaines dont celui du logiciel. L'avantage de l'utilisation des patterns est de diminuer le temps nécessaire au développement d'un logiciel, notamment en apportant des solutions validées déjà existantes à des problèmes courants de conception. C'est surtout les notions de capitalisation et de réutilisation apportées par les patterns qui ont séduit les concepteurs des systèmes informatiques.

Ainsi, en *1987*, *Cunningham* et *Beck* ont utilisé certaines idées d'*Alexander* pour développer des patterns dans le domaine de la conception des logiciels. Leur but était de fournir un ensemble de solutions réutilisables à des problèmes récurrents dans le développement des logiciels, plus particulièrement en ce qui concerne les interfaces utilisateur.

L'usage des patterns est, aujourd'hui largement, répandu dans la conception des systèmes interactifs, notamment en ce qui concerne les patterns dits de « *conception* » (*design patterns*).

✓ **Conception ascendante**

Les outils des approches ascendantes pour le développement d'interface peuvent être classifiés en trois catégories principales :

- *Les boîtes à outils*.
- *Les squelettes d'application*.
- *Les éditeurs graphiques*.

Une **boîte à outils** met, à disposition du programmeur d'interfaces, une collection de techniques d'interaction sous forme de composants logiciels réutilisables de petite taille [4]. Elles correspondent à une façon de manipuler les périphériques physiques (souris, tablette à digitaliser, clavier, écran…) pour faire le choix d'une commande (exemple : menu), une acquisition de valeur (exemple : Scrollbar), accéder à des fonctionnalités, effectuer un paramétrage (exemple : boutons), ou réaliser des affichages graphiques.

Les boîtes à outils fournissent un ensemble de modules dotés de fonctions communes (*API* : Application Programming Interface) produisant automatiquement des applications. Une grande variété de boîtes à outils est proposée au développeur comme *AWT* [32], *TCL*/*TK* [33] ou *MFC* [34].

Les boîtes à outils présentent plusieurs avantages. D'abord, de pouvoir réutiliser des outils d'interaction de plus haut niveau d'abstraction que les procédures fournies par les pilotes des périphériques d'entrées-sorties. Ensuite, elles standardisent l'aspect de toutes les applications construites avec la même boîte à outils ainsi que le comportement des objets de présentation (*look and feel*).

En revanche, les boîtes à outils ne disposent d'outils pour spécifier le séquencement des tâches ou le contrôle de dialogue, les échanges entre l'interface et l'application. Il en résulte des coûts importants de développement et de mise au point et souvent des difficultés d'utilisation de l'interface dues à des comportements mal implémentés.

Le principal problème résultant de l'utilisation des boîtes à outils est donc l'absence d'architecture guidant la construction de l'interface.

Les **squelettes d'application** sont des assemblages logiciels réutilisables et extensibles qui réalisent une large part de fonctions de l'interface d'une application. Ils remédient à certains des problèmes liés aux boîtes à outils, en particulier la duplication d'efforts et à un degré moindre d'apprentissage. Ainsi, la tâche du développeur consiste à remplir les trous du squelette et à récrire les parties prédéfinies du système inadaptées au besoin particulier de son application. La réutilisation, l'extensibilité et la surcharge de logiciel se traduisent directement dans les termes de la programmation par objets. La plupart des squelettes d'application utilisent donc cette technique et sont construits au dessus d'une boîte à outils.

Les squelettes les plus connus sont actuellement *Java Swing,* les squelettes associés à *MFC*, la boîte à outil de *Microsoft*, et *OpenStep* d'Apple et son clone libre *GNUstep*.

L'avantage des squelettes d'application est qu'ils fournissent une architecture logicielle. Le développeur n'a plus à déterminer l'assemblage correct des briques logicielles. De plus, le niveau de service procuré est plus proche des besoins du développeur. L'inconvénient majeur tient, essentiellement, aux difficultés de la réutilisation logicielle. Pour réutiliser de façon optimale un squelette d'application, il faut bien comprendre son fonctionnement et il faut aussi que le type d'application à développer rentre dans le moule. L'utilisation d'un squelette d'application nécessite que le concepteur connaisse la boîte à outils sous jacente et l'architecture sur laquelle le squelette est basé. De plus, les squelettes d'application sont par nature stéréotypés, ce qui limite leur domaine d'application. Plus le squelette d'application est proche de l'application finale, moins le développeur a de travail, mais moins ce squelette est adaptable à un grand nombre d'applications.

Les éditeurs graphiques d'interfaces, ou *GUIBuilders* (*Graphic User Interface Builders*), sont des logiciels donnant, aux concepteurs, la possibilité de créer la couche de présentation des interfaces graphiques. Cette création d'interface, par les moyens visuels, exclut toute forme de programmation graphique complexe par leur utilisateur [4]. La plupart des boîtes à outils, actuellement disponibles, fournissent des menus, des boîtes de dialogues, la gestion des fenêtres et de boutons, prêts à être utilisés. En revanche, ils offrent peu de supports pour des interfaces, représentant les données de manière spécifique au domaine, et utilisant des styles d'interaction variés [4].

Des éditeurs graphiques sont proposés par la plupart des boîtes à outils. On peut citer à titre d'exemple : *Visual C++* et *Visual Basic* de *Microsoft* pour les *MFC*, *JBuilder* pour *Java* de *Borland* [4].

- ✓ **Générateurs descendants**

D'autres outils ont été développés dans le but d'assister le concepteur dans le processus de développement d'interfaces. Ces outils proposent d'utiliser des langages spécialisés de haut niveau d'abstraction pour la description d'une application. La finalité d'une génération descendante devrait être la génération de l'interface à partir de ces spécifications.

Le terme *UIMS* (*User Interface Management System*) est principalement associé à la partie exécutive de l'interface construite selon le modèle d'architecture de *Seeheim*.

Les *UIMS* utilisaient déjà certains modèles. Depuis, les langages de spécification ont, considérablement, évolué, prenant en compte de plus en plus d'éléments et de modèles, pour générer des interfaces de plus en plus complexes. Cette évolution, que ce soit en nombre ou en langages de description des nouveaux modèles, conduit à préférer au terme *UIMS* le terme d'approche à base de modèles ou *MBA* (*Model Based Approach*) [4].

Un modèle représente la description d'un aspect caractéristique de l'interface d'une application interactive. Ceci inclut aussi bien la description du dialogue, du domaine que celui de la présentation [4].

Chapitre 4 la spécification formelle et les nouvelles interfaces

1. Introduction

L'objectif de ce chapitre est de mettre l'accent sur la spécification formelle ainsi que les nouvelles interfaces apparues dernièrement basées principalement sur le mode tactile.

2. La spécification formelle

Afin de permettre des vérifications, l'utilisation de techniques formelles pour les développements d'*IHM* a été préconisée [35]. Ces techniques ont été appliquées aussi bien pour les modèles d'architecture que pour les modèles de tâches [35]. Notons que l'application des techniques formelles dans les *IHM* a suivi l'évolution historique de ces techniques et de leur utilisation dans le génie logiciel [36].

Les premiers modèles ont utilisé des automates et/ou des extensions d'automates tels que les *Statecharts* [37] ou les *ATN* [38].

Une des notions importante dans la description formelle de systèmes interactifs est la notion d'interacteur. Un interacteur est un composant logiciel fournissant des services particuliers dans une *IHM*. Il est doté d'un état, il reçoit des évènements en entrée et en renvoie en sortie.

Différentes définitions des interacteurs ont été proposées suivant la nature de l'approche employée [39]. La notion d'interacteur repose sur un principe commun : la description d'un système interactif par composition de processus abstraits indépendants [4]. Ceux-ci ressemblent fortement aux modèles d'architecture multi-agents, par exemple *PAC*, dans le sens où l'interface est répartie en une multitude d'agents actifs, réagissant à la réception de certains évènements. Toutefois, à la différence d'une architecture multi-agents, les approches à base d'interacteurs explicitent formellement la communication externe et le comportement interne de ces interacteurs [4].

Au fil des approches, les interacteurs sont devenus de plus en plus expressifs pour aboutir à une modélisation plus approfondie d'un système interactif.

✓ **Interacteurs de CNUCE**

Les interacteurs de *CNUCE* [40] ont été développés dans le cadre de la formalisation du projet *GKS* [41]. Ce projet avait pour but de concevoir des composants de base (les interacteurs) pouvant être modélisés, construits et vérifiés.

Un interacteur est un composant de l'interface utilisateur. Son comportement est réactif et peut fonctionner en parallèle avec d'autres interacteurs.

Au niveau le plus abstrait, l'interacteur est vu comme une boîte qui sert d'intermédiaire entre un côté utilisateur et un côté application. Il peut émettre, recevoir des évènements des deux côtés de cet interacteur, et traiter, en interne, les données qui transitent. A un niveau moins abstrait, l'interacteur est vu comme une boîte blanche qui permet alors de comprendre son fonctionnement interne.

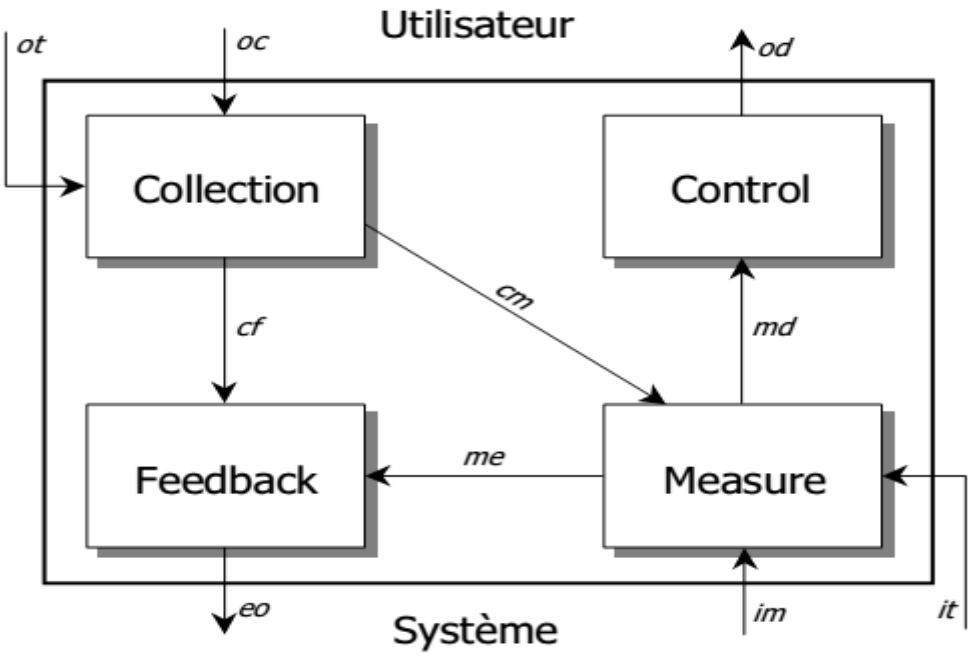

Fig.4.1 Boîte blanche de l'interacteur de CNUCE, adapté de [40].

✓ **Interacteurs de York**

L'idée de modéliser un système interactif au moyen d'interacteurs a été reprise par les travaux réalisés à l'université de *York* [42]. A la différence de ceux de *CNUCE*, les interacteurs de *York* permettent de modéliser, plus finement, le dialogue du système interactif dans le sens où l'état de l'interacteur est explicitement décri. Deux évolutions ont été proposées : les objets abstraits d'interaction [43], basés sur un modèle à états, et une deuxième approche basée sur un modèle événementiel [44].

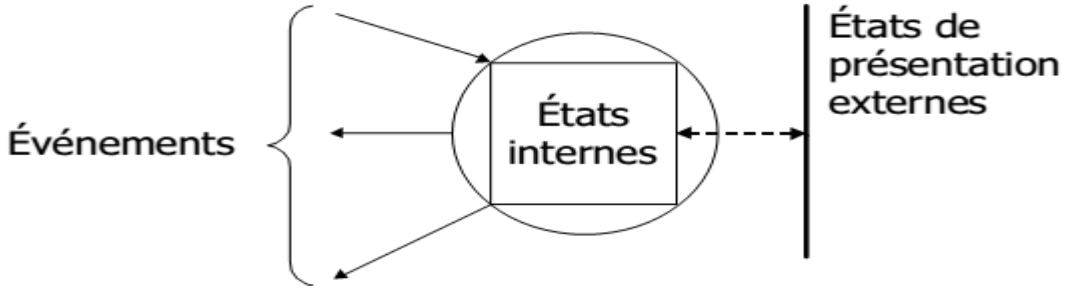

Fig.4.2 Schéma d'un interacteur de York [35].

Dan la version initiale [42], l'interacteur de *York* est décrit sous la forme d'un système à états de type automate, définissant l'état interne de l'interacteur. Lorsque cet interacteur reçoit des évènements ou des actions d'entrée (commandes qui proviennent de l'utilisateur ou de l'application), il modifie son état interne et associe à cette modification un changement d'état de présentation externe. Il peut aussi transmettre des évènements de sortie, encore appelés actions de sortie [35].

3. Les nouveaux dispositifs

Interfaces tactiles

Les interfaces tactiles sont des dispositifs permettant à l'utilisateur d'interagir avec son environnement via le toucher. Ces dispositifs agissent sur l'utilisateur en appliquant des forces, des vibrations ou des mouvements, permettant à quelqu'un de sentir des températures, des textures, des formes, des viscosités ou des forces. Pour stimuler la sensation des organes lorsqu'ils sont manipulés, il est nécessaire d'avoir une communication à double sens entre l'utilisateur et l'ordinateur rendue possible par une interface tactile. Au contraire de la vision et de l'audition, le toucher est un sens qui permet ce transfert bidirectionnel d'information entre l'utilisateur et un environnement virtuel [3].

Les interfaces tactiles ont plusieurs caractéristiques spécifiques. Une première caractéristique concerne l'actualisation des informations en fréquence et en temps réel. Par exemple, lorsque l'utilisateur manipule l'outil (stylo, joystick, etc.) d'un dispositif haptique, la nouvelle position et l'orientation de l'outil haptique sont acquises et les contacts avec l'objet virtuel sont détectés. Si un contact est détecté, les forces interactives sont calculées en utilisant les règles préprogrammées pour une réponse de collision. Puis, ceci est transmis à l'utilisateur à travers le dispositif haptique pour lui fournir une représentation tactile des objets en 3D et de leurs détails de surface. Par conséquent, il faut une boucle qui actualise

les forces à environ *1KHZ* car autrement, il semble que le dispositif haptique vibre [4]. Ces deux composantes : détection de contact et réponse de collision définissent généralement un algorithme de rendu haptique [5]. Malgré la puissance croissante des ordinateurs actuels, ces calculs devraient être optimisés et une simplification du monde virtuel est souvent nécessaire.

En outre, les mécanismes de synchronisation entre les scènes visuelles et tactile impliquent un processus d'exclusion mutuelle. Ces mécanismes peuvent mis en jeu pendant l'existence de différents canaux sensoriels, kinesthésiques et tactiles, où les besoins en termes de fréquence d'information ne sont pas les mêmes.

Une deuxième caractéristique spécifique des interfaces tactiles concerne la nature, la position, et la taille de l'espace de travail. L'espace de travail des interfaces tactiles est souvent réduit à l'accomplissement d'un geste simple. Les interfaces à base fixe ou mobile sont des tentatives, pour répondre aux nombreux problèmes impliquant la vitesse de transmission d'informations, destinées au dispositif tactile, le contrôle du robot porteur ou peur traiter des forces de résistance. L'espace de travail est donc un facteur limitant dans l'accomplissement du geste écologiquement valable, c'est-à-dire un geste qui est naturel et représentatif des expériences sensorielles de la vie réelle. De même, une différence de position ou d'échelle entre l'espace de visualisation et l'espace de manipulation induit une dégradation de l'immersion. Lorsque nous déplaçons la souris de l'ordinateur sur le coté du bureau pour déplacer le curseur à l'écran en est un exemple évident.

Fig.4.3 l'écran tactile.

L'avantage de l'utilisation des écrans tactiles et des stylos optiques, au-delà de l'interaction directe, est qu'elle nécessite seulement peu d'apprentissage facile. Cependant, leur inconvénient consiste à cacher une partie de l'écran avec une limitation au niveau de la précision.

Chapitre 5 les systèmes interactifs : de la spécification à la mise enœuvre

1. Introduction

Le prototypage rapide d'applications est, aujourd'hui largement, reconnu comme étant une activité si non indispensable à la qualité du système futur, au moins hautement souhaitable. Si le prototypage couvre en principe l'application complète [45], l'interface homme-machine de cette application constitue un composant de premier choix pour engager, le plus rapidement possible, un prototypage [46].

2. Prototypage

✓ **Définition**

Un prototype est une version simplifiée d'une partie de l'application finale. La méthode de prototypage consiste à effectuer un test sur cette version partielle et toujours inachevée du système en cours de développement pour obtenir, rapidement et à moindre coût, une idée du bon ou du mauvais fonctionnement de l'application.

Il existe une très grande variété de prototypes, du prototype très simple, rapide et économique à réaliser, au prototype « haute fidélité », beaucoup plus proche de l'application finale mais également beaucoup plus coûteux en temps et en ressources. En général, on considère que le prototype se définit selon trois grands critères (mais la terminologie employée peut varier).

Une première distinction peut être effectuée entre :

- Le **prototype « à jeter »**, qui est uniquement utilisé pour les tests de l'application mais ne sert pas de base à l'application finale.
- Le **prototype évolutif (ou « itératif »)**, qui consiste à réutiliser certaines ou toutes les parties du prototype pour servir de base à l'application finale. Par exemple, plusieurs versions intermédiaires successives d'un site web peuvent mener au site web final, en fonction des remarques émises lors des différentes simulations.

- Le **prototypage incrémental**, qui consiste à ajouter au prototype de nouvelles fonctionnalités au fur et à mesure du développement de l'application.

Ensuite, un prototype peut être :

- **Horizontal ou « statique »**, c'est-à-dire qu'il passe en revue les différents composants de l'interface homme-machine sans véritable interactivité (les commandes ne fonctionnent pas).

- **Vertical ou « dynamique »**, c'est-à-dire qu'il propose un ensemble de fonctionnalités de la future application et permet à un utilisateur de dérouler un scénario d'utilisation typique lors d'un test d'utilisabilité intermédiaire.

Enfin, le prototype soit « basse-fidélité », soit « haute-fidélité ». La fidélité du prototypage exprime la similarité entre l'esquisse de l'interface prototypée et la représentation graphique de l'interface finale [46], [47].

- Le **prototype « basse-fidélité »** est généralement réalisé rapidement et à moindre coût. Il n'offre pas de caractère interactif et ne présente pas nécessairement de ressemblance visuelle avec l'application finale. On dit que la fidélité est faible [48] si la représentation du prototype évoque l'interface finale sans la représentation totalement en détails.

- Le **prototype « haute-fidélité »**est alors, beaucoup plus lourd et coûteux à développer, généralement interactif et ressemblant aussi fidèlement que possible à l'application finale. On dit que la fidélité est élevée [48], [49], [50] si la représentation du prototype est la plus proche possible de celle de l'interface finale. Dans ce cas, un prototype à un niveau de fidélité élevée devrait être aussi proche que possible de l'interface finale en termes de représentation [51] (quels sont les objets interactifs graphiques utilisés), de navigation globale (comment naviguer entre les conteneurs graphiques), de navigation locale (comment naviguer à l'intérieur d'un conteneur graphique), de contenu (la richesse du modèle de données) et de fonctionnalités.

Ainsi, un **prototype papier** (ou réalisé sur base de transparents Powerpoint) constitue un prototype « à jeter » (il n'est pas repris pour servir de base au développement de

l'application finale), horizontal (il s'intéresse essentiellement à l'interface homme-machine) et « basse-fidélité ».

A l'inverse, une partie d'un site web testée par quelques utilisateurs est un prototype évolutif (il sera repris, une fois validé, dans la version finale du site), vertical et « haute-fidélité ».

La *Fig.5.1* montre une comparaison des caractéristiques des différents niveaux de fidélité.

Fidélité\critère	**Basse**	**Moyenne**	**Elevée**
Phase de développement	Enumération des besoins, conception préliminaire, conceptualisation, début de l'application	Conception continue, validation de l'ergonomie du prototype, application en cours	Conception détaillée, application en fin de spécification
Contenu	Présentation surtout	Présentation, layout, contenu, début de la navigation	Présentation et navigation, contenu, layout, fonctionnalités
Usage	Exploration, découverte, évocation, communication	Simulation, raffinement, itération, amélioration, validation de l'utilisabilité, test	Propagation générale à l'application, spécification finale, documentation, marketing
Type de prototypage	Horizontal	Diagonal	Vertical
Type d'approche	Ascendante (bottom up)	Expansive (middle-out)	Descendante (top down)
Facilité de changement	Elevée	Moyenne	Très faible
Coût	Faible	Moyen	Elevé
Temps requis	Faible	Moyen	Elevé
Naturalité de représentation	Très élevée	Moyenne	Faible
Détail	Faible	Moyen	Elevé
Fréquence d'itération	Très élevée	Elevée	Faible
Niveau d'interactivité	Faible	Moyen	Elevé
Représentation	Esquisse	Dessin, dessin vectoriel	Présentation et navigation réelles
Applications cibles	De grande taille	De taille moyenne	De taille réduite ou fragments
Niveau d'abstraction des spécifications	Abstrait (indépendante des modalités d'interaction et des plates-formes	Mixte (propre à une modalité, indépendante des plates-formes)	Concret (propre à une modalité d'interaction sur une plate-forme donnée)

Fig.5.1 Comparaison des caractéristiques des différents niveaux de fidélités [51].

Les différentes techniques de prototypage servent en général de base de travail à d'autres méthodes d'implication des utilisateurs. A titre d'exemple, la méthode du prototypage peut être complétée par :

- Un focus groupe ou une séance de brainstorming, par exemple lorsque les concepteurs présentent, aux autres membres du groupe de développement de l'application, une

maquette ou un « prototype papier » du site web ou du logiciel en les invitant à réagir et donner leurs avis à propos de la structure, de l'interface ou du graphisme du site.

- Une évaluation experte, par exemple lorsqu'un prototype horizontal présentant l'interface de l'application sert de base à l'inspection des critères d'ergonomie et d'utilisabilité par un expert.
- Un test d'utilisabilité, dans le cas d'un prototype vertical « dynamique » testé par un ou plusieurs utilisateurs représentatifs. Dans ce cas, plus le développement est avancé et plus le prototype est proche de l'application finale, plus la méthode du prototypage se rapprochera du test d'utilisabilité.

Le principal objectif du prototypage est de pouvoir pré-tester l'application avant même qu'elle ne soit prête et de déceler et corriger les erreurs de conception au fur et à mesure du développement. Selon le type de prototype choisi, la méthode du prototypage est complétée par d'autres méthodes d'implication des utilisateurs (observation, brainstorming, focus groupe, test d'utilisabilité, évaluation expert, etc.).

Un avantage du prototypage est aussi de pouvoir tester une partie de l'application développée indépendamment des autres éléments de l'application.

✓ **Contexte d'utilisation**

La méthode du prototypage est utile tout au long de la conception de l'application, lorsque l'interface définitive n'est pas encore déterminée. Le prototypage est particulièrement utile pour impliquer, rapidement et à moindre coût, les utilisateurs dans le développement de l'application.

Dans le cas du prototypage itératif ou évolutif, plus le développement est avancé, plus le prototype est de « haute-fidélité » et plus ses caractéristiques sont proches de celles de l'application finale (à ce stade, le prototype devient une sorte de version préliminaire de l'application).

Lors des premières étapes de la conception, il s'agit généralement d'un prototype peu évolué (de basse-fidélité), par exemple réalisé à base de papier et reprenant les fonctions de l'application finale, mais présentant peu ou pas de ressemblances visuelles, ou d'un prototype horizontal reprenant la structure (l'interface homme-machine) sans interactivité.

Plus la conception du prototype est avancé, plus le prototype (de haute-fidélité), se rapproche des tests d'utilisabilité.

✓ **Les acteurs**

Ici, il faut distinguer la conception du prototype (sa réalisation) de son utilisation (la façon dont il permet de prendre en compte l'avis des utilisateurs).

En ce qui concerne la conception du prototype, il faut avoir une équipe de développement impliquée dès la phase d'analyse pour élaborer rapidement une maquette de ce que pourrait être l'application ou pour réaliser un prototype parallèlement au développement de l'application (beaucoup plus dans le cas de prototypes de haute-fidélité ou verticaux, relativement lourds à concevoir). Cependant, certains prototypes peuvent être réalisés assez rapidement et sans nécessiter de compétences techniques particulières (une maquette ou un prototype papier par exemple).

Quant à l'utilisation du prototype pour récolter les informations des utilisateurs, différentes méthodes d'implication des utilisateurs et donc différentes personnes interviendront au cours du prototypage selon le type de prototype choisi. Par exemple :

- Un (voire plusieurs) observateur (s) et des utilisateurs représentatifs dans le cas du test d'utilisabilité d'un prototype vertical.
- Un ou plusieurs experts en ergonomie et utilisabilité dans le cas de l'évaluation experte d'un prototype vertical.

✓ **Quelques questions liées à la mise en œuvre d'un prototype**

Dans cette section, on aborde quelques questions largement posées lorsqu'on vient à mettre en œuvre un prototype.

Faut il faire appel à un sous traitant pour réaliser le prototype ?

Compte tenu des principaux objectifs de la méthode de prototypage (pré-tester une application à moindre coût et de manière relativement rapide), il semble peu opportun de faire appel à un sous-traitant pour réaliser un prototypage.

Ainsi, un prototype papier est simple et rapide à réaliser. Il est donc préférable de ne pas faire appel à un sous-traitant car cela augmenterait le coût et les délais de mise en œuvre de

la démarche. De plus, ce sont les concepteurs de l'application qui sont les mieux placés pour réaliser le prototype et pour tirer les enseignements des tests effectués.

De même, dans le cadre d'un prototypage évolutif, les parties testées et validées du prototype sont reprises et intégrées dans l'application finale. La réalisation du prototype n'engendre pas nécessairement une surcharge de travail trop conséquente et il semble à nouveau peu opportun de faire appel à un sous-traitant.

Si un prototype de « haute-fidélité » requiert un travail de conception tel que l'équipe de développement de l'application ne peut l'assumer, alors il est préférable de se tourner vers une autre méthode d'implication des utilisateurs plutôt que de faire appel à un sous-traitant.

Quel type de prototypage choisir ?

La principale question qui se pose lors du recours au prototypage est celle de savoir si l'on opte plutôt pour un prototype de « basse fidélité » ou pour un prototype de « haute fidélité ».

Au niveau de la mise en œuvre de la démarche, les prototypes de « basse fidélité » (comme ceux réalisés à partir de transparents Powerpoint ou, surtout, les prototypes papier) présentent quelques avantages non négligeables par rapport à un prototype de « haute fidélité » :

- Les prototypes de haute-fidélité sont trop longs et trop lourds à développer et à modifier. Même avec les outils les plus performants, il faut parfois quelques semaines pour développer certains prototypes de haute-fidélité. Il suffit de quelques heures pour réaliser un prototype papier.
- Avec un prototype de haute-fidélité, les remarques des testeurs tendent à se disperser sur l'apparence du prototype plutôt que sur son contenu. En revanche, étant donné le caractère « fait main » du prototype papier, les testeurs se concentrent davantage sur le contenu, l'apparence étant secondaire.
- Dans le cas du prototypage évolutif, la réalisation du prototype a parfois demandé tellement de temps et d'énergie aux développeurs qu'ils pourraient être peu enclins à tenir compte de toutes les remarques des utilisateurs et à modifier radicalement leur

prototype. Ces résistances n'interviennent pas dans le cas d'un prototype papier réalisé en quelques heures.

- Un prototype de haute-fidélité fait naître des attentes parfois démesurées. Si un prototype interactif et ressemblant assez fidèlement à l'application finale est testé avec succès, les utilisateurs et même certains membres de l'équipe de projet peuvent s'attendre à ce que l'application finale soit prête dans un laps de temps relativement court. Or, la charge de travail entre un prototype de haute-fidélité performant et l'application finale reste parfois très importante.
- Un simple bug dans un prototype de haute-fidélité peut réduire à néant l'ensemble du test, tandis qu'il est toujours possible de corriger très rapidement (voire instantanément) une erreur constatée dans le prototype papier.
- Enfin, le prototype de haute-fidélité intervient généralement assez tard dans le développement de l'application. Les erreurs repérées, à ce stade du développement, risquent donc d'être plus difficiles, plus lourdes et plus chères à corriger que si elles avaient été repérées plus tôt dans la conception de l'application. Les prototypes papiers sont efficaces car ils contribuent à conscientiser très tôt les développeurs aux démarches d'évaluation et d'utilisabilité.

Quelles sont les ressources nécessaires ?

Le matériel requis pour mette en œuvre un prototype de « haute-fidélité » est celui nécessaire pour développer l'application. Cependant, le matériel requis pour réaliser un prototype papier est par nature très simple. En pratique, chaque concepteur ou chaque équipe de projet développera ses propres méthodes de constructions à partir de quelques matériaux de prédilection, mais de manière générale, toutes les fournitures de bureau « classiques » peuvent être utiles comme :

- Du papier carton blanc, au minimum de *format A4* (*le format A3* est préférable) et suffisamment robuste pour résister aux multiples manipulations et corrections.
- Des cartons ou des « *post-its* » de différentes tailles et couleurs, qui seront notamment utiles pour figurer les boites de dialogue ou les « boutons ».
- Du papier collant, du ruban adhésif, de la colle, une agrafeuse, des trombones ou toute autre fourniture d'assemblage.

- Des marqueurs, des crayons, des surligneurs de différentes couleurs et différentes épaisseurs de trait.
- Des transparents et les marqueurs correspondants.
- Un tableau blanc ou un tableau en papier.
- Une gomme et du correcteur.
- Une paire de ciseaux.
- Une photocopieuse.

Pour réaliser un prototype papier visuellement proche d'une application informatique, il est également possible de réaliser à partir de captures d'écran une bibliothèque de ressources de base :

- Des fenêtres de navigateur.
- Des boutons de commandes.
- Des boîtes de dialogue.
- Des barres de déroulement.

Outre ces différentes fournitures, il faudra également prévoir pour le jour du test :

- Un local accueillant.
- Une table de taille suffisante pour *4* à *5* personnes et pour tout le matériel relatif au prototype.

Quel est le résultat du prototype ?

Sur la base des remarques des observateurs et des impressions des utilisateurs, on établit la liste des problèmes (avec éventuellement les solutions proposées) en les classant selon leur degré de gravité ou de priorité.

Un rapport écrit, synthétisant ces problèmes et les priorités, est rédigé et sert de base à une réunion durant laquelle les membres de l'équipe de projet discuteront des modifications à apporter à l'application.

En fonction de la rapidité avec laquelle on parvient à recruter et rassembler les utilisateurs, un prototypage papier peut livrer ses résultats en environ une semaine.

3. La génération automatique du code

La génération de code source est une opération permettant de générer automatiquement du code source. Son but est d'automatiser la production de code source répétitif afin de minimiser les risques d'erreurs et de permettre au programmeur de se concentrer sur l'écriture de code à plus grande valeur ajoutée.

Il existe de nombreuses sources à partir desquelles on génère le code source :

- ✓ Génération à partir de programmes informatiques écrits dans un autre langage de programmation, généralement de plus haut niveau. Le logiciel réalisant cette transformation peut être appelé compilateur ou traducteur.
- ✓ Génération à partir de métadonnées. Par exemple, on peut générer la couche logicielle d'accès à une base de données relationnelle grâce aux métadonnées des tables.

Le problème de la génération de code source est la maintenance de grande quantité de code source généré.

Conclusion Générale

Nous avons présenté, dans ce cours, les concepts de base du domaine de génie des systèmes interactifs, une continuité pour le domaine de l'interface homme machine, ainsi que les différentes architectures et la conception liée aux systèmes interactifs tenant en compte les spécificités de tels systèmes.

Références

1. Nigay, L. (1994). Conception et modélisation logicielles des systèmes interactifs: application aux interfaces multimodales (Doctoral dissertation, Université Joseph-Fourier-Grenoble I).
2. Catherine Recanati. Cours « Introduction sur l'interaction Homme Machine » sur le site web : https://lipn.univ-paris13.fr/~recanati/docs/M2-InHM/Interaction.ppt Université de Paris 13.
3. Michel Beaudouin cours « Ingénierie des systèmes interactifs » sur le site web : https://www.lri.fr/~mbl/ENS/IHM/ecole-in2p3/Cours/cours1.html
4. Samaan, K. (2006). Prise en Compte du Modèle d'Interaction dans le Processus de Construction et d'Adaptation d'Applications Interactives (Doctoral dissertation, PhD thesis, École central de Lyon).
5. Norman, D. A., & Draper, S. W. (1986). User centered system design: New perspectives on human-computer interaction. CRC Press.
6. Scapin, D. L., & Bastien, J. C. (1996). Inspection d'interfaces et critères ergonomiques (Doctoral dissertation, INRIA).
7. Bastien, J. C., Leulier, C., & Scapin, D. L. (1998). L'ergonomie des sites web. Créer et maintenir un service Web, 111-173.
8. Palanque, P., & Bastide, R. (1993, October). Interactive cooperative objects: an object-oriented formalism based on petri nets for user interface design. In Systems, Man and Cybernetics, 1993.'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on (Vol. 3, pp. 274-278). IEEE.
9. Paternò, F., & Faconti, G. (1992). On the use of LOTOS to describe graphical interaction. People and computers, 155-155.
10. Card, S. K. (2017). The psychology of human-computer interaction. CRC Press.
11. Card, S. K., Moran, T. P., & Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. Communications of the ACM, 23(7), 396-410.
12. Kieras, D. (1997). A guide to GOMS model usability evaluation using NGOMSL. In Handbook of Human-Computer Interaction (Second Edition) (pp. 733-766).
13. Gray, W. D., John, B. E., & Atwood, M. E. (1993). Project Ernestine: Validating a GOMS analysis for predicting and explaining real-world task performance. Human-computer interaction, 8(3), 237-309.
14. Hartson, H. R., Siochi, A. C., & Hix, D. (1990). The UAN: A user-oriented representation for direct manipulation interface designs. ACM Transactions on Information Systems (TOIS), 8(3), 181-203.
15. Kobsa, A. (1989). User models in dialog systems (p. 10). W. Wahlster (Ed.). Berlin: Springer-Verlag.
16. Petri, C. A. (1962). Fundamentals of a Theory of Asynchronous Information Flow.
17. Jensen, K. (2013). Coloured Petri nets: basic concepts, analysis methods and practical use (Vol. 1). Springer Science & Business Media.
18. Merlin, P. M. (1975). A study of the recoverability of computing systems.
19. Le Bail J., Alla H., David R., Réseaux de Petri Hybrides, Techniques et Science Informatiques, Vol 11, N° 5, 1992, pp. 95-119.

20. Garlan, D., & Perry, D. E. (1995). Introduction to the special issue on software architecture. IEEE Trans. Software Eng., 21(4), 269-274.
21. Nigay, L., & Coutaz, J. (1993, May). A design space for multimodal systems: concurrent processing and data fusion. In Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems (pp. 172-178). ACM.
22. Guittet, L. (1995). Contribution à l'ingéniérie des interfaces homme-machine: théorie des interacteurs et architecture H4 dans le système NODAOO (Doctoral dissertation, Poitiers).
23. Kasik, D. J. (1982, July). A user interface management system. In ACM SIGGRAPH Computer Graphics (Vol. 16, No. 3, pp. 99-106). ACM.
24. Bass, L., Faneuf, R., Little, R., Mayer, N., Pellegrino, B., Reed, S., ... & Szczur, M. R. (1992). A metamodel for the runtime architecture of an interactive system. SIGCHI bulletin, 24(1), 32-37.
25. Ferber, J. (1997). Les systèmes multi-agents: vers une intelligence collective. InterEditions.
26. Krasner G. E., Pope S. T., A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80. Journal of Object Oriented Programming, Vol. 1, N°3, 1988, pp. 26-49.
27. Coutaz, J. (1988). Interface homme-ordinateur: conception et réalisation (Doctoral dissertation, Université Joseph-Fourier-Grenoble I).
28. Ouadou, K. E. (1994). AMF: Un modèle d'architecture multi-agents multi-facettes pour Interfaces Homme-Machine et les outils associés (Doctoral dissertation, Ecully, Ecole centrale de Lyon).
29. Poquet, J. (1998). Architecture AMF des systèmes interactifs: conception d'un moteur de gestion d'interactions en Java. Mémoire de DEA, Ecole Centrale de Lyon.
30. Alexander, C. (1977). A pattern language: towns, buildings, construction. Oxford university press.
31. Geary, D. M. (1999). Graphic Java 2.0: die JFC beherrschen. 1.(AWT) (Vol. 1). Pearson Deutschland GmbH.
32. Ousterhout, J. K., & Jones, K. (2009). Tcl and the Tk toolkit. Pearson Education.
33. Jeff, P. (1999). Programming windows with MFC.
34. Aït-Ameur, Y., Aït-Sadoune, I., & Baron, M. (2005). Modélisation et Validation formelles d'IHM: LOT 1 (LISI/ENSMA). Délivrable pour le projet RNRT-VERBATIM, 73.
35. Gaudel, M. C. (1995). Formal specification techniques for interactive systems. In Design, Specification and Verification of Interactive Systems' 95 (pp. 21-26). Springer, Vienna.
36. Harel, D. (1987). Statecharts: A visual formalism for complex systems. Science of computer programming, 8(3), 231-274.
37. Wasserman, A. I. (1981, March). User Software Engineering and the design of interactive systems. In Proceedings of the 5th international conference on Software engineering (pp. 387-393). IEEE Press.
38. Markopoulos, P. (1997). A compositional model for the formal specification of user interface software (Doctoral dissertation, University of London).
39. Paternò, F., & Faconti, G. (1992). On the use of LOTOS to describe graphical interaction. People and computers, 155-155.
40. Enderle, G., Kansy, K., & Pfaff, G. (2012). Computer Graphics Programming: GKS—The Graphics Standard. Springer Science & Business Media.

41. Duke, D. J., & Harrison, M. D. (1995). Event model of human-system interaction. Software Engineering Journal, 10(1), 3-12.
42. Duke, D. J., & Harrison, M. D. (1993, August). Abstract interaction objects. In Computer Graphics Forum (Vol. 12, No. 3, pp. 25-36). Blackwell Science Ltd.
43. Duke, D. J., & Harrison, M. D. (1995). Event model of human-system interaction. Software Engineering Journal, 10(1), 3-12.
44. Boar, B. H. (1984). Application prototyping. A requirements definition strategy for the 1980's. A Wiley-Interscience Publication, New York: Wiley, 1984.
45. Rettig, M. (1994). Prototyping for tiny fingers. Communications of the ACM, 37(4), 21-27.
46. Hong, J. I., Li, F. C., Lin, J., & Landay, J. A. (2001, March). End-user perceptions of formal and informal representations of web sites. In CHI'01 Extended Abstracts on Human Factors in Computing Systems (pp. 385-386). ACM.
47. Rudd, J., Stern, K., & Isensee, S. (1996). Low vs. high-fidelity prototyping debate. interactions, 3(1), 76-85.
48. Walker, M., Takayama, L., & Landay, J. A. (2002, September). High-fidelity or low-fidelity, paper or computer? Choosing attributes when testing web prototypes. In Proceedings of the human factors and ergonomics society annual meeting (Vol. 46, No. 5, pp. 661-665). Sage CA: Los Angeles, CA: SAGE Publications.
49. Sefelin, R., Tscheligi, M., & Giller, V. (2003, April). Paper prototyping-what is it good for?: a comparison of paper-and computer-based low-fidelity prototyping. In CHI'03 extended abstracts on Human factors in computing systems (pp. 778-779). ACM.
50. Vanderdonckt, J., & Coyette, A. (2007). Modèle, méthodes et outils de support au prototypage multi-fidélité des interfaces graphiques. Revue d'Interaction Homme-Machine Vol, 8(1).