Interfaces Homme-Machine : un regard critique

Joëlle Coutaz
Laboratoire de Génie Informatique (IMAG-Campus)
BP 53 X
38041 Grenoble Cedex
tel. 76 60 91 57, 76 51 48 54
UUCP email : joelle@imag.imag.fr

Résumé

Cet article examine l'état de l'art dans le domaine de l'interaction homme-machine en prenant le point de vue de l'informaticien. Il inclut une analyse de l'apport des sciences cognitives et identifie les implications de cet apport sur les architectures et les outils logiciels. La contribution des sciences cognitives est envisagée ici en fonction de ses retombées sur les méthodes de conception. Nous constaterons que ces méthodes s'appuient nécessairement sur une approche itérative avec, pour conséquence, une architecture logicielle modulaire particulière. Une analyse critique des erreurs du passé nous conduira à la définition du modèle multiagent qui propose des architectures adaptées à la conception itérative et à la prise en compte de critères ergonomiques. L'article considère ensuite les outils logiciels favorisant l'application d'un tel modèle.

1. Introduction

"La mise en œuvre de logiciel interactif est une tâche difficile, longue et coûteuse". Cette affirmation refuse de disparaître en dépit des progrès techniques et des efforts de recherche dans le domaine de l'interaction homme-machine. Les raisons à cela sont multiples :

- l'attitude de l'utilisateur change. A l'usager inquiet et soumis aux "ordres de la machine", succède un consommateur mieux averti qui veut voir en l'ordinateur un compagnon de travail productif ;
- l'innovation technologique accélère l'allure. Elle se manifeste par le progrès des performances qui s'expriment maintenant en multiples de MIPS. Elle se constate aussi par l'apparition de nouveaux modes de communication comme l'audiovisuel et des périphériques d'avant-garde tels le gant électronique [Zimmerman 87].

Les systèmes informatiques doivent répondre à cette évolution par un double effort d'adaption : satisfaire aux exigences grandissantes des utilisateurs et tenir compte d'une technologie

hautement évolutive. Les méthodes actuelles et les outils logiciels de l'interaction homme-machine ne permettent pas de remplir ces conditions facilement. En raison d'une approche guidée par l'expérimentalisme, en raison aussi de l'absence de théories adaptées, le savoir-faire dans ce domaine n'est pas généralisable. Il est donc difficilement applicable aux situations nouvelles. Actuellement, il n'est pas de solution prête à l'emploi. Il n'est que des compromis à pratiquer avec bon sens.

Cet article tente de décrire le paysage du domaine de l'interaction homme-machine en prenant le point de vue de l'informaticien. Une autre analyse serait celle de l'ergonome et du psychologue. Nous allons identifier l'apport de l'ergonomie cognitive au paragraphe suivant et indiquer comment ces connaissances s'intègrent dans une méthode de conception. Nous constaterons que, dans l'état actuel de notre savoir-faire (ou de notre incompétence!), la conception itérative des interfaces constitue la seule approche raisonnable et effective. L'itération a ses implications sur l'architecture des logiciels. Au paragraphe 3, nous analyserons les architectures possibles en identifiant les erreurs du passé et en insistant sur les espoirs de solution avec la tendance actuelle du modèle multiagent. La mise en place d'une architecture peut être facilitée par des outils logiciels. Au paragraphe 4, nous décrirons l'apport mais surtout les insuffisances des techniques actuelles.

2. Apport de l'ergonomie cognitive et méthode de conception

La psychologie et l'ergonomie cognitives procèdent selon deux approches. L'une produit des théories, l'autre, expérimentale, énonce des conseils pratiques. On trouvera dans [Scapin 87, Shneiderman 87] une énumération de tels principes. Nous retenons ici l'application de théories à la mise en place d'une méthode de conception.

2.1. Apports théoriques

2.1.1. Les théories de l'interaction homme-machine : évaluation

Une théorie permet d'expliquer un phénomène ou d'en prédire l'occurrence. Dans le domaine de la psychologie cognitive, il n'existe pas de théorie explicative ou prédictive unique qui recouvre tous les aspects du sujet humain. Il existe une multitude de théories spécifiques dédiées chacune à l'étude d'une classe de phénomènes précis : théories de la résolution de

problèmes, théories de la représentation mentale, théorie de l'apprentissage, théorie des langages, et bien d'autres.

Chaque théorie sensibilise l'informaticien à une classe particulière de concepts et de processus cognitifs ; mais la dispersion a comme résultat une somme impressionnante d'enseignements partiels. Cette situation est probablement tolérable pour le cogniticien qui s'intéresse à un aspect précis du comportement. Elle ne l'est plus pour l'informaticien qui se doit d'envisager les besoins d'un sujet humain sans le découper en tranches fonctionnelles!

Malgré l'absence de cohésion et en dépit d'un champ d'application limité, les modèles théoriques de la psychologie cognitive tels le modèle du processeur humain, GOMS [Card 83] et la théorie de l'action [Norman 86] définissent un cadre utile de réflexion. Ils constituent, selon Allen Newell, des "outils pour la pensée" [Newell 86]. La théorie de l'action, bien qu'informelle, est à ce titre intéressante.

2.1.2. La théorie de l'action

La théorie de l'action introduit la notion de modèle conceptuel et précise les étapes cognitives mises en jeu dans la réalisation d'une tâche à l'aide d'un système informatique.

Un *modèle conceptuel* est une représentation mentale de la connaissance en termes de variables psychologiques. Une variable psychologique correspond à un concept, une unité de connaissance (mnème), un sujet d'intérêt. Par exemple, dans une tâche d'édition de texte, les notions de paragraphe, de mot, de phrase, de point d'insertion, etc., sont des variables psychologiques propres au domaine de l'édition de texte. Un modèle conceptuel est incomplet et imprécis mais il évolue avec l'expérience et guide l'essentiel du comportement.

La première étape de la réalisation d'une tâche est *l'établissement d'un but*. Un but est une représentation mentale de l'état souhaité du système. Cet état se manifeste par des variables physiques, c'est-à-dire des informations produites par le système, perceptibles et éventuellement manipulables par l'utilisateur au moyen de dispositifs physiques de contrôle comme le clavier et la souris. Par exemple, dans le domaine de l'édition de documents, le mot affiché sur l'écran et le curseur clignotant sont des variables physiques gérées par le système et manipulées par l'utilisateur avec une souris, un clavier.

La seconde étape est la *formulation d'une intention*. Celle-ci exprime la distance entre le but recherché et la représentation mentale de l'état effectif du système. Cette représentation est le

produit de la traduction de l'état du système en termes psychologiques et de la comparaison de ce résultat avec le but. On remarque que la traduction n'est possible que si le modèle conceptuel inclut la connaissance de la correspondance entre les variables physiques et les variables psychologiques. Par exemple, dans le domaine de l'édition de document, l'utilisateur doit associer la variable physique "curseur" à la notion de point d'insertion.

La troisième étape est la *mise en place d'un plan d'actions* dont l'exécution à l'aide des dispositifs de contrôle devrait permettre d'atteindre le but. La mise en place de ce plan requiert la connaissance des relations entre les variables psychologiques impliquées dans le but et les variables physiques. Elle repose aussi sur la connaissance des liens entre les variables physiques et les dispositifs de contrôle. Par exemple, l'utilisateur doit savoir que le curseur se déplace à l'aide de la souris ou de touches réservées du clavier.

La quatrième étape concerne *l'exécution* du plan d'actions en agissant sur les dispositifs de contrôle. Le système change d'état. Les étapes suivantes incluent la *perception* du changement d'état (les variables physiques ont évolué) et *l'interprétation* de la modification des variables physiques en termes psychologiques. Ce processus aboutit à une représentation mentale du nouvel état du système qui est alors comparée au but. Cette *évaluation* peut donner lieu à la poursuite du plan ou à sa modification.

Les sept étapes de la théorie de l'action ne se succèdent pas forcément dans l'ordre cité mais elles identifient des points clefs. En particulier, elles mettent en évidence l'existence de processus de traduction dont le succès repose sur la mise en correspondance entre variables psychologiques et variables physiques ainsi que sur l'établissement d'associations entre les variables physiques et les dispositifs de contrôle. La conséquence de cette observation est la conception d'interfaces homme-machine qui favorisent la mise en place de telles relations.

2.2. Méthode de conception

En l'absence de théorie analytique, non réductrice, et adaptée à la conception d'interface homme-machine, les informaticiens ont nécessairement recours à l'approche empirique. A la lumière de l'expérience, il semble que la tâche de conception puisse se décomposer en six sous-tâches essentielles : esquisse du problème à résoudre, identification de l'utilisateur, définition des tâches et des objets spécifiques au domaine d'application, définition des objets et fonctions informatiques, définition de la présentation de l'interface, et évaluation expérimentale. Ce

schéma est une vue simplifiée du processus de conception. En pratique, les sous-tâches sont chacune soumises à des révisions accompagnées de nombreux retours arrière.

2.2.1. Définition d'un modèle de l'utilisateur

La définition d'un modèle de l'utilisateur type s'appuie généralement sur des données informelles recueillies au moyen de questionnaires, de discussions, d'enregistrements vidéo. L'analyse de ces données est un art difficile que les informaticiens ne maîtrisent pas toujours. Un point de départ possible est une classification générale en utilisateurs novices, experts et occasionnels. A cette dimension, il faut ajouter la catégorie socio-professionnelle et le type de connaissances (connaissances dans le domaine d'application, connaissances dans le domaine informatique).

"Parler aux utilisateurs n'est pas un luxe. C'est une nécessité!" affirme Gould [Gould 88]. Rencontrer l'utilisateur n'est cependant pas toujours possible ; il faut alors "le synthétiser". Même dans ce cas extrême, mieux vaut définir un modèle incomplet et approché que d'évacuer le problème. En effet, en l'absence d'un modèle de référence, le concepteur sera dépourvu de guide pour la mise en œuvre des compromis qui, inévitablement, se présenteront aux étapes suivantes.

2.2.2. Définition des tâches et des objets du domaine d'application

La définition des tâches et des objets du domaine d'application s'effectue sans considération aucune pour la dimension informatique. Les objets correspondent aux variables psychologiques introduites par la théorie de l'action. Ils décrivent les notions propres au domaine. Des opérateurs, qui constituent les tâches élémentaires du domaine, leur sont associés.

Parmi ces notions, celle d'erreur est trop souvent négligée parce que mal cernée. Une erreur sémantique est un objet qui relève du domaine. Elle doit donc être modélisée.

Une fois les objets et leurs opérateurs introduits, il faut identifier les tâches et déterminer leurs relations. La technique usuelle consiste à décomposer une tâche en sous-tâches jusqu'à la rencontre des tâches élémentaires. La difficulté ici est double : l'absence de formalisme adapté et le rôle déterminant de la décomposition.

- La nature du formalisme n'est pas l'objet de notre discussion mais ce point, et notamment l'expression du parallélisme nécessaire à la modélisation de tâches parallèles ou d'actions simultanées, reste un sujet de recherche important.
- La décomposition des tâches du domaine définit la structure de l'espace de travail futur de l'utilisateur. Elle détermine la forme générale des plans d'action introduits au paragraphe 2.1.2. Lorsque cette structure n'est pas conforme au modèle de résolution de l'utilisateur, l'accomplissement de tâche paraît complexe, voire impossible.

La spécification du domaine est une opération difficile mais essentielle puisque, nous l'avons vu, elle conditionne la facilité d'utilisation du système futur. Une technique pratique consiste à définir des scénarios pertinents, de les faire résoudre par différentes équipes, puis de comparer les solutions. Il est clair que la participation de l'utilisateur est ici précieuse.

2.2.3. Définition des objets et des fonctions informatiques

Le domaine cerné, il faut pratiquer sa projection en termes de concepts informatiques : objets et commandes. Les objets se répartissent à leur tour en objets sémantiques et syntaxiques.

Un objet sémantique modélise un objet du domaine. La règle générale veut qu'à tout objet du domaine corresponde un objet sémantique. Par exemple, si la notion de mot est importante dans le domaine de l'édition de texte, l'objet sémantique "mot" doit être explicite dans la réalisation informatique. La notion d'erreur doit être précisée. Au cours d'expériences successives, nous avons trouvé utile de modéliser le concept d'erreur sous forme d'un objet sémantique : une structure de données qui inclut un niveau de gravité et l'identification des objets sémantiques impliqués dans l'erreur ou dans la correction de l'erreur. Le code de gravité peut être exploité dans la présentation de l'interface pour attirer l'attention de l'utilisateur (au moyen, par exemple, d'une icône évocatrice) ; et l'identification des objets sémantiques fournit à l'utilisateur des informations précieuses qui facilitent la compréhension et la correction de l'erreur. Les messages laconiques tel "erreur de syntaxe", sont évidemment faciles à produire mais ne sont d'aucun secours pour remédier à la situation.

Un objet syntaxique participe à la réalisation de tâches syntaxiques. Une tâche syntaxique ne relève pas du domaine mais est induite par le support informatique. Par exemple, l'utilisation des systèmes de fenêtrage implique la manipulation de fenêtres, le défilement d'informations ("scrolling"), ou permet d'ouvrir plusieurs vues sur un même document. Ces tâches sont dites syntaxiques ; la notion de vue multiple, qui intervient dans une tâche syntaxique, peut se voir comme un objet syntaxique. Son rôle concerne la gestion de la cohérence visuelle entre les diverses présentations d'un même objet sémantique.

La distinction entre objet sémantique et objet syntaxique est parfois difficile à établir. Pour illustrer ce problème, nous choisissons l'exemple du "status-man" de la figure 1. Cet objet indique à l'utilisateur l'état général du système. Il suppose un comptage des erreurs par niveau de gravité (critique ou tolérable) et, selon les résultats, exprime son mécontentement, son indifférence ou sa satisfaction. Une vue synthétique de l'état du système correspond à un besoin de l'utilisateur, donc à une variable psychologique du modèle conceptuel. Faut-il lui associer un objet sémantique? Si les fonctions de comptage relèvent du domaine, alors le status-man est un objet sémantique ; inversement, si elles sont indépendantes du domaine, il faut considérer le status-man comme un objet syntaxique. Nous verrons au paragraphe 3 comment la distinction entre objets sémantiques et objets syntaxiques établit les bases de l'architecture logicielle des systèmes interactifs.

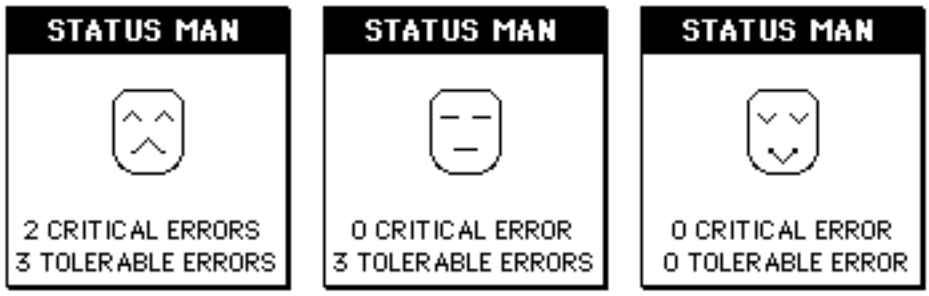

Figure 1 : Le Status-Man, objet sémantique ou objet syntaxique?

Les objets interviennent comme paramètres de commande. La définition des commandes du système s'appuie nécessairement sur la décomposition de tâches. Le point de départ usuel est de considérer les tâches élémentaires comme des commandes. Il n'est cependant pas toujours possible d'établir une correspondance biunivoque entre les deux ensembles, commandes et tâches élémentaires. Le problème de la granularité des commandes reste ouvert : si le niveau d'abstraction des commandes est trop fin, l'utilisateur doit prendre en charge la composition des commandes pour accomplir les tâches usuelles. Si le niveau est trop élaboré (macroscopique), les tâches usuelles s'expriment directement mais l'utilisateur risque de rencontrer de graves difficultés pour l'expression des tâches qui ne se décomposent pas dans les termes du langage de commande.

2.2.4. Définition de la présentation de l'interface

La présentation de l'interface définit le comportement perceptible du système. L'objectif est de choisir la présentation des objets sémantiques et syntaxiques en sorte que l'utilisateur élabore un modèle conceptuel en accord avec le fonctionnement du système. Cette étape peut s'effectuer en s'appuyant sur deux règles essentielles : (1) tout objet sémantique et syntaxique doit avoir son correspondant sous forme d'objet de présentation; et (2) l'état du système doit être explicite.

La règle (1) découle naturellement de la théorie de l'action. Nous avons observé dans cette théorie qu'une correspondance directe entre variables psychologiques et variables physiques, facilitait les processus de traduction entre les représentations mentales et physiques. Un objet de présentation (on parle aussi d'objet d'interaction, d'interacteur, ou d'objet interactif) est la version électronique de la notion de variable physique. La difficulté pour le concepteur d'interface est le choix d'objets de présentation qui véhiculent la sémantique du domaine. Il est clair qu'une canne à pêche ne permet pas d'exprimer la notion de température! Un thermomètre semble mieux approprié.

Lorsque c'est possible, le choix des objets de présentation peut se laisser guider par la métaphore du monde réel [Hutchins 86]. L'application d'une telle métaphore consiste à reproduire électroniquement le comportement des objets du monde réel et à insister sur la composante manipulation directe. Par exemple, le "desktop" du Macintosh utilise la métaphore du bureau. La souris agit comme l'extension électronique de la main et tend ainsi à provoquer chez l'utilisateur le sentiment d'engagement et de participation active.

La règle (2), rendre l'état du système explicite, découle également de la théorie de l'action. Si l'état est explicite, l'étape perception-interprétation-évaluation s'en trouve facilitée et le but, l'intention et le plan d'actions peuvent être ajustés en connaissance de cause. Expliciter l'état a de multiples facettes, parmi lesquelles :

- les variables physiques, donc les objets d'interaction, qui présentent un intérêt pour la tâche en cours doivent être visibles ou facilement accessibles ;
- les actions invalides doivent "se voir". La technique usuelle consiste à afficher les interdits en grisé clair et inhiber toute réaction aux agissements de l'utilisateur ;
- les gestes de la manipulation directe doivent laisser une trace à l'écran. Par exemple, un objet en cours de déplacement doit suivre les mouvements de la souris, un objet dont on change la taille doit devenir élastique, etc.

Ces principes paraissent relever du simple bon sens mais leur mise en application se heurte bien souvent à l'absence d'outils logiciels appropriés. Face à la difficulté de mise en œuvre, le

réalisateur d'interface fera usage de compromis (par exemple, il utilisera une forme particulière de curseur pour traduire le déplacement d'un objet) ou même renoncera à satisfaire le principe ergonomique.

2.2.5. Evaluation de la conception

La description ci-dessus des différentes phases de conception est émaillée de remarques montrant l'a peu près de la méthode. Dans ces conditions, l'évaluation des choix de conception est inévitable. On se reportera à [Shneiderman 87, Gould 88] pour une description des techniques possibles d'évaluation : simulation, maquette, prototype, etc.

L'évaluation à partir d'un prototype semble offrir des résultats plus précis que ceux obtenus au moyen de simulations. Cette constatation explique le développement des générateurs d'interface, qui, à partir d'une spécification de l'interface homme-machine, sont capables de produire une interface exécutable. Cette classe d'outils ne garantit pas la qualité ni l'adéquation de l'interface mais elle permet de réaliser des prototypes très rapidement à moindre coût. Elle encourage donc la mise au point itérative des systèmes interactifs. Nous en verrons les limites actuelles au paragraphe 4.2.

2.3. Apport de l'ergonomie cognitive : conclusion

L'apport des sciences cognitives à la conception des systèmes interactifs est un ensemble confus de connaissances mal reliées. Les théories sont imprécises ou, lorsqu'elles sont formalisées, sont trop réductrices. Néanmoins, elles définissent un cadre de pensée utile. Ce cadre peut servir de canevas à une ébauche de méthode comme celle esquissée ici. Si la méthode explicite des étapes directrices, elle n'offre aucun support pour effectuer les bons choix. En particulier, les outils d'évaluation jugent après coup mais n'offrent pas de solution. L'informaticien a besoin d'outils dont la critique soit constructive.

Le message essentiel de ce paragraphe sur l'apport des sciences cognitives est notre incapacité actuelle à produire des systèmes sans avoir recours à la démarche itérative. La conséquence, pour l'informaticien, est la mise en place de techniques logicielles adaptées, et notamment d'architectures logicielles souples.

3. Architecture logicielle

En génie logiciel, la conception itérative (ou plus précisément : cyclique) se traduit par l'usage de la modularité. Si les principes de cette technique sont connus, leur application est difficile à maîtriser en l'absence de modèle de décomposition. En interface homme-machine, un consensus s'est formé autour d'un modèle de base, esquisse de l'organisation générale de tout logiciel interactif. Les premières applications de ce modèle n'ont pas abouti au succès escompté. L'expérience nous conduit aujourd'hui à une vision plus dynamique du processus de l'interaction. Ce paragraphe présente successivement le modèle de base, les erreurs du passé et la tendance.

3.1. Le modèle d'architecture de base

La théorie de l'action distingue deux mondes qu'elle modélise avec les notions de variables psychologiques et de variables physiques. La méthode de conception présentée ci-dessus reprend cette dichotomie en indiquant l'existence d'un domaine d'application, qui correspond aux variables psychologiques, et d'un domaine de présentation, qui renferme les variables physiques. La mise en œuvre logicielle peut donc s'inspirer de cette distinction fondamentale.

Comme le montre la figure 2, un système interactif comprend l'Application et le Composant de Présentation.

- L'Application modélise le domaine à l'aide d'objets sémantiques liés par des relations.
- Le Composant de Présentation est un ensemble organisé d'objets interactifs qui définissent la partie perceptible du système.

La conception itérative et sa compagne immédiate, la réutilisation logicielle, suggèrent que l'Application n'ait aucune connaissance des objets de présentation et réciproquement. Ici, l'absence de connaissance ne signifie pas absence de communication mais implique des références par indirection. La double nécessité d'assurer à la fois l'indépendance et les échanges d'information entre l'Application et le Composant de Présentation, se traduit par la présence d'un troisième composant : le Contrôleur de Dialogue.

Le Contrôleur de Dialogue est responsable de la gestion des relations et des échanges entre l'Application et le Composant de Présentation.

- Une première classe de relations désigne la correspondance entre les objets sémantiques et les objets interactifs. Nous rappelons qu'à tout objet sémantique doit correspondre au

moins une technique de présentation. Le lien entre ces deux classes d'objets, c.-à-d. le mécanisme d'indirection, est géré par le Contrôleur de Dialogue.

- Une seconde classe de relations concerne l'expression de la communication d'informations. L'Application et le Composant de Présentation définissent deux perspectives d'un même domaine mais chaque perspective a un rôle précis à remplir. Dans l'Application, la modélisation du domaine est guidée par des considérations de calcul alors que dans le Composant de Présentation, le modèle est conçu pour manipuler et rendre perceptible l'état du système. En conséquence, l'Application et le Composant de 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 de l'Application, l'autre à ceux du Composant de Présentation.

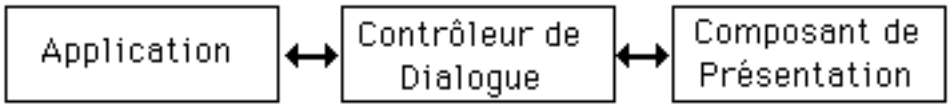

Figure 2 : Le modèle d'architecture de base. Une flèche à double sens traduit l'existence d'échanges d'informations entre deux composants.

En résumé, l'Application met en œuvre les objets sémantiques définis dans la phase 3 de la conception (paragraphe 2.2.3), le Composant de Présentation réalise la partie perceptible du système définie dans la phase 4 de la conception (paragraphe 2.2.4), et le Contrôleur de Dialogue assure les mécanismes d'indirection et de traduction entre les deux mondes.

3.2. Les erreurs de passé

L'une des erreurs dont les effets pervers s'infiltrent encore dans de nombreux systèmes interactifs, est l'assimilation de l'organisation tripartite du modèle de base au fonctionnement des systèmes de traitement des langages.

Certains ont vu dans l'interaction homme-machine une analogie avec le dialogue entre individus [Foley 84]. Un dialogue s'appuie sur un langage. Le système et l'utilisateur engagés dans un dialogue font donc usage d'un langage qui, selon la théorie, est défini par une sémantique, une syntaxe et un lexique. En plaquant cette décomposition sur l'organisation du modèle de base, nous observons que l'Application correspond aux actions sémantiques, que le Contrôleur de Dialogue remplit les fonctions d'un analyseur syntaxique et que le Composant de Présentation effectue l'analyse lexicale.

Cette vision, qui s'appuie sur des concepts connus et bien maîtrisés, a le double avantage de fournir un cadre de pensée simple et de permettre la réutilisation des techniques de compilation. Par exemple, la génération d'analyseurs syntaxiques est applicable à la production automatique de Contrôleurs de Dialogue. Par ce transfert, on hérite d'un fonctionnement prêt à l'emploi mais ce fonctionnement répond mal aux règles de l'ergonomie cognitive.

Par définition, l'approche langage concentre l'attention sur la forme des échanges au détriment de leur dynamique. Ce faisant, le fonctionnement du système obtenu procède séquentiellement. En compilation, cette propriété n'est pas une restriction. En interaction homme-machine, elle devient gênante : la forme d'une expression d'entrée peut évoluer en parallèle avec la production d'une expression de sortie ; ou encore plusieurs expressions d'entrée, éventuellement indépendantes, peuvent être spécifiées en parallèle, ou de manière alternée.

L'entrelacement des expressions d'entrée et de sortie se manifeste dans les interfaces de manipulation directe qui exigent de la part du système, une réaction immédiate et informative. L'exemple des formes élastiques est évocateur. La forme élastique, qui sert à repérer le geste, est généralement utilisée pour les opérations de changement de dimension d'objet. Une syntaxe possible est la suivante : la spécification de la taille de l'objet est complète lorsque l'utilisateur relâche le bouton de la souris. Tant que le bouton reste enfoncé, une expression d'entrée partielle (la localisation actuelle de la souris) est entrelacée avec la production d'une expression de sortie (la forme élastique qui traduit la taille de l'objet si le bouton était relâché à cet instant). Un compilateur, qui travaille sur des expressions complètes, effectue, en séquence, la réception de l'expression d'entrée, son analyse, puis l'appel aux actions sémantiques. Ce fonctionnement séquentiel sur des expressions complètes ne convient pas aux interfaces de manipulation directe qui produisent des expressions partielles et exigent des résultats sémantiques intermédiaires pour des expressions incomplètes.

La spécification parallèle ou entrelacée de plusieurs expressions d'entrée est le fait des interfaces à fils d'activité multiples ("multithread dialogues"). Un fil d'activité modélise une branche du plan de résolution de l'utilisateur. Ce dernier, qui se comporte de manière opportuniste [Hayes-Roth 79], passe de branche en branche en fonction des résultats de l'étape d'évaluation. Face à un système informatique, ce comportement se traduit par la spécification d'un début de commande, puis par une suite d'actions en vue de la spécification d'une seconde ou d'une troisième commande pour enfin reprendre la spécification de la première. Un compilateur traite une expression à la fois alors que le système informatique doit être capable de mémoriser les divers fils d'activité.

En résumé, un fonctionnement séquentiel centralisé, (1) n'encourage pas les retours d'information sémantiquement riches, (2) ne soutient pas le comportement opportuniste de l'utilisateur mais au contraire, (3) héritant de la machine d'états finis des techniques de compilation, tend à guider l'utilisateur dans une suite prédéfinie d'étapes. Enfin, le modèle langage suppose une frontière bien nette entre l'Application et le reste du système. Or, nous avons noté dans notre méthode de conception, l'existence d'objets (les objets syntaxiques) dont l'appartenance à l'Application n'est pas toujours claire. Ces observations conduisent à une nouvelle approche qui modélise un système interactif comme un continuum de niveaux d'abstraction animés par des agents coopérants.

3.3. La tendance : le modèle multiagent

Le modèle multiagent s'inspire du fonctionnement des systèmes de type stimuli-réponses. Un système stimuli-réponses est un ensemble organisé d'agents capables de réagir à des phénomènes externes déterminés (des stimuli) et de produire à leur tour des stimuli.

3.3.1. Définition

Le modèle multiagent structure un système interactif en un ensemble d'agents spécialisés qui produisent et réagissent à des événements (des stimuli).

Un agent est un système de traitement de l'information : il possède des récepteurs et des émetteurs pour acquérir et produire des événements ; il dispose d'une mémoire à deux niveaux : l'une pour enregistrer les événements détectés par les récepteurs, l'autre pour mémoriser un état ; il comprend un processeur cyclique spécialisé dans le traitement d'une ou plusieurs classes d'événements. Le résultat d'un traitement se traduit généralement par un changement d'état de l'agent et par l'émission de nouveaux événements. L'état fournit à tout instant une image de l'évolution d'un agent.

Le modèle multiagent se caractérise par une organisation fortement modulaire, des traitements exécutés en parallèle et une communication par événements. Ce modèle abstrait est à rapprocher des modèles à objets pour lesquels il existe des outils de réalisation. Au-delà de sa correspondance directe avec une technique d'implémentation, le modèle multiagent apporte un début de solution aux problèmes évoqués jusqu'ici.

3.3.2. L'apport du modèle multiagent

L'agent définit la granularité de la modularité. Il est donc possible de modifier localement un comportement sans mettre en cause le fonctionnement de l'ensemble. La machine d'états finis de la vision langage rend au contraire cette opération périlleuse.

L'agent définit l'unité d'exécution. Il est donc possible de l'exécuter sur un calculateur physiquement distinct du lieu de création. Cette propriété est essentielle à la réalisation de systèmes répartis tels les logiciels de travail de groupe (ou "groupware") [Ellis 89].

L'agent définit une unité d'état. Cet état peut contenir les informations nécessaires à la reprise d'une activité locale et l'agent modélise alors un fil d'activité interruptible. Un ensemble cohérent d'agents coopérants réalise avec l'utilisateur (ou avec un groupe d'utilisateurs), un dialogue à plusieurs fils d'activité. Un agent, de même qu'un utilisateur, sont les acteurs de l'accomplissement d'une tâche.

Le modèle multiagent ici décrit est la définition générique d'une approche. Les modèles MVC [Goldberg 84] et PAC [Coutaz 87, Coutaz 88, Coutaz 90, Nigay 90] sont des exemples relevant de cette approche.

3.3.3. Exemples : MVC et PAC

MVC structure un agent en trois composantes réalisables chacune sous forme d'un objet Smalltalk : le Modèle pour représenter un comportement interne, la Vue pour afficher l'état du Modèle sur l'écran, et le Contrôle pour interpréter les actions de l'utilisateur sur la Vue.

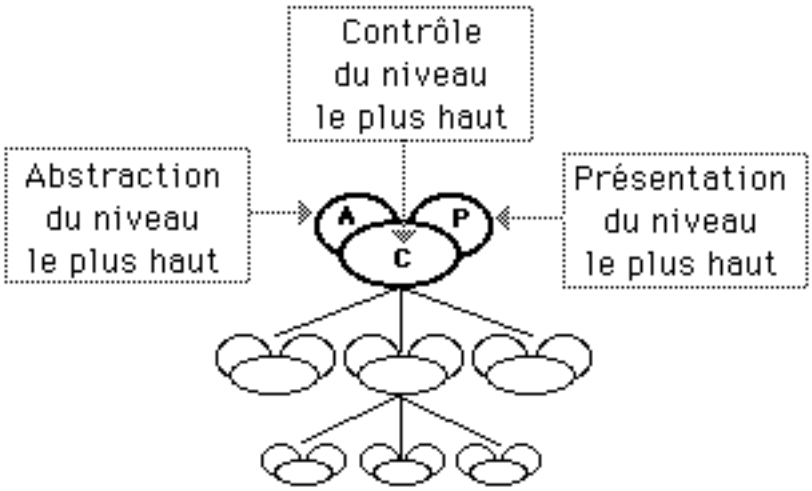

Figure 3 : Une architecture PAC. Les traits pleins figurent des relations entre agents de niveaux d'abstraction distincts.

Comme le montre la figure 3, PAC structure récursivement un système sous forme d'une hiérarchie d'agents. La hiérarchie permet d'exprimer des relations entre agents et reflète un continuum de niveaux d'abstraction depuis l'Application jusqu'aux éléments fins de l'interaction. Un agent PAC définit une compétence à un niveau d'abstraction donné. C'est un acteur à trois facettes : la Présentation définit le comportement perceptible de l'agent, l'Abstraction représente son expertise, et le Contrôle a un double rôle : (1) il sert de lien entre les facettes Présentation et Abstraction de l'agent et (2) il gère des relations avec d'autres agents PAC de la hiérarchie.

Dans la figure 3, l'Abstraction du niveau le plus haut correspond à la notion d'Application du modèle de base et le Contrôle du niveau le plus haut remplit des fonctions similaires au Contrôleur de Dialogue. La figure 4 montre un agent PAC élémentaire, feuille de hiérarchie : la Présentation sait dessiner une forme de réchaud et reçoit les événements qui traduisent les actions de l'utilisateur ; l'Abstraction, avec ses valeurs d'état EnMarche, Chaleur et qqChoseDessus, constitue un modèle de l'état d'un réchaud ; le Contrôle effectue le pont entre les deux facettes en gérant la correspondance des phénomènes abstraits et concrets. Par exemple, à la réception du signal (émis par la Présentation) d'une pression sur l'interrupteur, le Contrôle demande à l'Abstraction d'appliquer une opération de négation sur la valeur booléenne EnMarche. Noter que les mécanismes d'indirection et de traduction évoqués à propos du modèle de base, se retrouvent dans PAC à tous les niveaux d'abstraction. L'avantage est une souplesse accrue qui joue en faveur de la mise au point itérative. On trouvera dans [Coutaz 90] des exemples complets d'application de ce modèle.

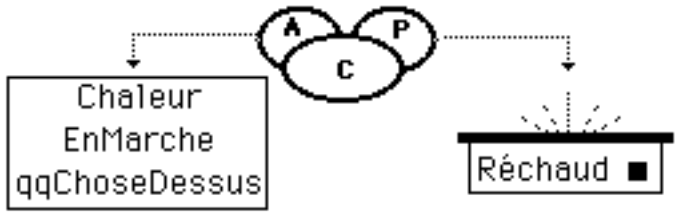

Figure 4 : Un exemple d'agent PAC.

MVC et PAC diffèrent en plusieurs points :

- MVC distingue, au sein d'un agent, le traitement des entrées (assuré par un Contrôleur) des opérations de sortie (la Vue) alors que PAC réunit ces deux aspects dans un composant unique de Présentation. La distinction entre les comportements d'entrée et de sortie offre une meilleure souplesse. Cependant dans les couches basses de l'interaction, on observe fréquemment l'asservissement mutuel des expressions de sortie et des événements d'entrée. La séparation de MVC entraîne des échanges par messages qui risquent d'allonger le temps de réaction du système.

- PAC explicite la notion de Contrôle, organe arbitre et traducteur entre le monde abstrait et concret. Dans MVC, cet aspect est inexistant, mettant en péril l'indépendance entre les deux facettes.

- PAC suggère une organisation systématique par une décomposition récursive. Cette structure est réalisable avec le modèle MVC mais elle n'est pas explicite. L'expérience montre l'utilité de relations hiérarchiques [Coutaz 89]. Par exemple, un agent peut être utilisé pour assurer la cohérence visuelle entre les différentes vues d'un même objet sémantique (chaque vue étant réalisée par un agent) ; ou encore, un agent peut cimenter en une commande une suite d'actions de l'utilisateur réparties sur plusieurs agents. Cette situation est typique des interfaces qui proposent à l'utilisateur une palette de choix et une zone de travail. Toute commande exige une action sur la palette et sur la zone de travail, deux agents qui s'ignorent et qui doivent donc être mis en contact par un tiers.

3.4. Architecture : conclusion

Le modèle multiagent remplace le serveur séquentiel de la vue langage par une organisation d'acteurs coopérants en parallèle. A la gestion centralisée de l'état de l'interaction se substitue une totale répartition des charges. Ce parallélisme et cette répartition sont les conditions nécessaires à la prise en compte des méthodes de résolution de l'utilisateur. La répartition présente en sus deux retombées intéressantes : une grande modularité et l'ouverture vers les traitements physiquement distribués.

Les modèles multiagent ici décrits, et bien d'autres, se présentent comme des variations sur un thème commun. Tous mettent en évidence la répartition des charges et la distinction entre comportement interne et comportement perceptible, mais aucun, jusqu'ici, ne s'est imposé comme solution véritable. Cela tient essentiellement à leur manque de précision et à la présence

d'ambiguïtés qui laissent entier le problème de l'identification des agents. Par exemple, PAC suggère une organisation mais ne précise pas le rôle des agents intermédiaires dans la hiérarchie. Un travail de formalisation de règles heuristiques permettant d'identifier les agents et leur coopération est en cours dans le cadre du projet ESPRIT-BRA AMODEUS [Amodeus 89].

De toute évidence, la voie est ouverte vers des logiciels mieux adaptés à la conception itérative et à la prise en compte de critères ergonomiques. Cependant, il est essentiel que cette approche soit disponible sous forme d'outils.

4. Outils logiciels

Les outils logiciels pour la construction de système interactif se répartissent en deux catégories : les boîtes à outils et les environnements de développement d'interfaces.

4.1. Boîte à outils

Une boîte à outils est une bibliothèque (ou un ensemble de bibliothèques) dont les services couvrent les différents niveaux fonctionnels d'un logiciel interactif : fenêtrage, affichage, objets de présentation, voire squelette de gestion de dialogue tels MacApp [Schmucker 86].

Il y a cinq ans, à l'arrivée du Macintosh, la situation était simple : le marché était dénué d'outils graphiques (à l'exception de GKS, utilisé à des fins particulières). Aujourd'hui, nous assistons à une prolifération de propositions autour des trois grandes catégories de plate-formes : Unix, MS-DOS/OS2, et Mac OS et ses proches (GEM, Intuition). Si le niveau d'abstraction de ces services est semblable d'un environnement à l'autre, l'interface de programmation et les styles de présentation varient sensiblement. La conséquence de cette diversité est, pour l'informaticien, le problème du transport des logiciels et le choix d'une boîte à outils et, pour l'utilisateur, l'impossibilité de transférer d'un système à l'autre ses connaissances sur un "mode d'emploi".

Très récemment, la tendance inflationiste des différences s'est transformée en une course aux normes qu'alimente, pour une large part, le milieu industriel. Pour la plate-forme Unix-X windows, citons l'effort Open Look du groupement SUN Microsystems et AT&T qui s'attache à définir la nature et la présentation des objets de dialogue de base, et en face, UEC (User Environment Component) dans le cadre du consortium OSF (Open Software Foundation). Les caractéristiques de Motif, la boîte à outils produite par OSF, laissent imaginer la difficulté des

compromis : en gros, Motif utilise l'interface de programmation de DEC et le style de présentation de NewWave de Hewlett Packard.

A ces travaux de large envergure s'opposent les initiatives ponctuelles illustrées par les environnements de programmation. Ces environnements tel le Prolog de Delphia [Delphia 87], incluent une boîte à outils portable. Ces bibliothèques se comportent comme des "boîtes à outils virtuelles" dont les primitives sont interprétables par différentes boîtes à outils cibles. XVT [Valdez 89], qui assure la portabilité des applications entre MS-Windows et le Toolkit du Macintosh, est une autre tentative relevant du même principe.

Si la normalisation des interfaces est incontournable, il faut cependant se demander si les tentatives actuelles ne sont pas prématurées. Dans un domaine aussi sensible à l'évolution de la technologie que l'est l'interface homme-machine, il semble risqué d'arrêter aujourd'hui des décisions que dénonceront dès demain les progrès des techniques informatiques et des sciences cognitives. En particulier, on ne semble pas s'inquiéter de l'intégration des différentes modalités d'interaction, tel l'audiovisuel et le graphique, pour lesquelles on se contente d'une cohabitation cloisonnée. Un fait est malheureusement certain : la guerre des normes qui vient de s'ouvrir entre les grands groupements ne facilite pas la tâche à court terme du programmeur, ni celle de l'utilisateur (pour lequel la cohérence des détails de présentation ne suffit pas à garantir l'adéquation d'une interface).

4.2. Environnement de développement d'interfaces

Un environnement de développement d'interfaces comprend un ensemble d'outils d'aide à la conception, à la construction et à l'évaluation de systèmes interactifs. Jusqu'ici, seule la construction d'interfaces a reçu une attention particulière avec l'étude de la génération automatique d'interfaces. L'expérience acquise dans ce domaine nous permet de tirer quelques enseignements utiles.

4.2.1. La génération d'interface : les leçons du passé

Jusqu'à ces dernières années, la génération d'interfaces s'appuyait sur une description textuelle de l'interface à l'aide d'un langage spécialisé (Domain/Dialogue [Schulert 85], Cousin [Hayes 85]) ou d'une notation, extension de formalismes existants (graphes de transitions, grammaires formelles [Olsen 83, Jacob 84, Wasserman 85]).

L'expérience montre que la description du dialogue ne se suffit pas d'un langage hors contexte. Pour cette raison, les grammaires hors contexte sont exclues dès l'instant où la spécification d'interface va au-delà d'une simple description de la forme syntaxique. Les graphes de transition sont inadaptés à l'expression du parallélisme ou à la traduction du caractère opportuniste du comportement de l'utilisateur. Une description qui souhaite refléter ces aspects fondamentaux de l'interaction explose sous l'effet combinatoire des cas possibles. Les formalismes à événements visent à améliorer la qualité des échanges tout en tenant compte de l'expression du parallélisme [Hill 87].

D'une façon générale, les outils développés jusqu'ici autour des automates d'états finis ne spécifient que la moitié du dialogue (les expressions d'entrée) et ne garantissent pas des échanges suffisants et permanents entre l'Application et le Contrôleur de Dialogue. De ce fait, ils laissent à l'Application la charge de produire les expressions de sortie. La distinction des rôles entre l'Application et la Présentation, condition nécessaire à la mise au point itérative, est donc remise en question ; et le Contrôleur de Dialogue ainsi que la Présentation, laissés à l'écart de la production des expressions de sortie, ne peuvent réutiliser les expressions produites par l'Application comme expressions d'entrée. La pauvreté des échanges avec l'Application rend difficile, voire impossible, l'entrelacement de l'analyse d'expressions d'entrée partielles avec la production d'expressions de sortie.

4.2.2. Génération d'interface : la situation actuelle

A l'heure actuelle, la génération d'interface s'effectue selon trois axes complémentaires : description de la Présentation, expression du Contrôle du dialogue et spécification de l'interface avec l'Application.

Pour la plupart des générateurs, la spécification d'interface concerne la composante Présentation. Ces outils permettent de constituer des schémas d'écran statique à partir des objets de présentation d'une boîte à outils. Ils offrent la possibilité d'associer ces objets à des traitements internes (pouvant agir sur la Présentation ou faire appel à des traitements sémantiques de l'Application). Dans cette catégorie, il faut citer UIL [OSF 89], UIMX de Hewlett Packard et DevGuide de Sun. Pour certains, tel UIL, la spécification de la Présentation nécessite l'apprentissage d'un langage textuel, alors que pour d'autres, tel UIMX, la construction des écrans s'effectue de manière interactive.

Les outils d'expression du Contrôle du dialogue permettent au concepteur de décrire les règles d'enchaînement entre les écrans statiques. Le terme "règle d'enchaînement" recouvre plusieurs

phénomènes dynamiques et notamment : les conditions de création et de destruction des exemplaires d'objets de présentation, les conditions d'apparition et de disparition de ces objets à l'écran, etc. Dans cette catégorie d'outils, citons Serpent [Bass 88] qui dispose d'un langage spécialisé pour l'expression du contrôle du dialogue. Parmi les outils les plus avancés, tel Interface Builder [Webster 1989], l'expression du contrôle du dialogue combine à la fois la programmation par démonstration et la programmation dans un langage général. La programmation par démonstration consiste à exprimer "un algorithme" par le geste. Le système, interprète des gestes, produit le code correspondant à l'algorithme. Dans Interface Builder, la programmation par démonstration se cantonne à l'expression de liens et d'enchaînements simples entre objets de présentation. Il est clair que l'essentiel de l'expression du contrôle du dialogue doit être exprimé en Objective-C.

La troisième catégorie de générateurs s'appuie sur une description de l'interface avec l'Application pour produire automatiquement la Présentation et le Contrôle du dialogue. On parle alors de spécification d'interface utilisateur à un haut niveau d'abstraction. La spécification de l'interface avec l'Application décrit la sémantique de l'Application du point de vue de l'interface utilisateur. Elle inclut la définition des structures de données et des fonctions exportées par l'Application à destination de l'utilisateur. La génération automatique de la Présentation s'appuie essentiellement sur une technique de correspondance entre types abstraits et classes d'objets de présentation (par exemple, un type énuméré est représenté par un menu) et le contrôle du dialogue utilise les conditions d'activation des fonctions d'interface (par exemple, l'opération de destruction d'un type d'objet n'est activable que s'il existe au moins un objet de cette classe). Mickey [Olsen 89], UIDE [Foley 88], ITS [Bennett 89] et MacIda UIMS [Petoud 89] relèvent de cette catégorie de générateurs.

Les générateurs de la première catégorie, qui mettent l'accent sur la Présentation, offrent une souplesse raisonnable dans le choix et l'assemblage des objets de présentation à condition toutefois que ces objets prédéfinis conviennent à la présentation des concepts de l'Application. Par définition, ces outils ne permettent pas de décrire la dynamique du dialogue qui doit alors être exprimée dans le programme client. Dès lors, ils imposent au concepteur l'apprentissage des mécanismes sous-jacents d'une boîte à outils et la définition d'une architecture logicielle adaptée. Ces outils ne garantissent donc pas la distinction modulaire entre les traitements qui relèvent du domaine de l'Application de ceux qui concernent l'interface : une procédure "callback" associée à un objet d'interaction peut mêler des calculs relevant à la fois du domaine d'application et de la présentation.

Un langage spécialisé de description du contrôle du dialogue permet au contraire de distinguer la nature des objets de présentation de leurs relations dynamiques. Toutefois, comme pour les outils de spécification de la Présentation, les outils de description du contrôle du dialogue sont limités par la base d'objets de présentation disponibles ainsi que par leur pouvoir d'expression. L'extensibilité passe nécessairement par la programmation usuelle. L'approche dont HyperCard [Harvey 88] s'est fait le pionnier, qui intègre la spécification interactive de la Présentation, la programmation par démonstration et la programmation traditionnelle du Contrôle du dialogue, est une voie prometteuse à condition toutefois que le concepteur soit guidé au sein d'une architecture saine. A ce titre, la notion de script d'HyperCard et d'Interface Builder fournit un canevas louable mais insuffisant.

Les outils de spécification de l'interface avec l'Application fournissent un canevas conceptuel qui garantit l'indépendance du code de l'Application vis-à-vis des techniques de restitution. Ils ont aussi l'avantage de proposer une présentation prête à l'emploi et cohérente de système en système. Les concepteurs héritent ainsi des heuristiques ergonomiques câblées dans le générateur. La spécification à un haut niveau d'abstraction peut en outre être exploitée à deux fins utiles : rendre explicite l'état sémantique du système et fournir de l'aide contextuelle. Par exemple, MacIda UIMS produit automatiquement un graphe d'état qui montre à l'utilisateur ce qu'il a accompli et ce qu'il lui reste à faire pour mener à bien une tâche donnée (par exemple, émettre un bon de commande). Ce graphe se déduit directement de la description du fonctionnement logique de l'Application. De même, les pré- et post- conditions associées aux fonctions peuvent être exploitées à la manière de STRIPS [Fikes 71] pour produire des plans d'aide [Senay 90]. L'encapsulation inhérente aux outils de spécification de haut niveau a ses contre-parties. Dans le cas présent, le concepteur d'interface ne peut agir sur le contrôle du dialogue ni sur les techniques de présentation. L'approche choisie par ITS qui consiste à permettre au concepteur averti de programmer les règles heuristiques de présentation constitue une tentative intéressante.

Nous venons de constater que chaque catégorie de générateur a ses avantages et ses limites propres. Au-delà des spécificités, tous les outils actuels manquent de proposer bon nombre des services fondamentaux identifiés par les spécialistes en ergonomie et en psychologie cognitive. Parmi les lacunes les plus criantes, nous retenons :

- l'absence des services généraux usuels tels le couper-coller, le défaire-refaire ou le rafraîchissement des fenêtres-application qu'il faut encore programmer "à la main" au-dessus d'une boîte à outils (voire au niveau élémentaire du système de fenêtrage!) ;
- l'absence de greffes pour l'intégration d'un simple service d'aide factuelle qui fournit un minimum d'informations descriptives ;

- la non prise en compte du multilinguisme alors que la dimension internationale des logiciels n'est plus à démontrer. Les éditeurs de ressources n'apportent que des solutions élémentaires. En effet, le multilinguisme ne se suffit pas de simples mécanismes de traduction des termes lexicaux ;

- la non prise en compte des techniques multimédia alors que l'on parle de "réalité artificielle" ;

- l'absence de support pour les activités de groupe alors que des plate-formes d'accueil, tels les systèmes répartis à objets, commencent à apparaître [Black 86, Krakowiak 90] ;

- l'absence de technique pour l'insertion d'un modèle de l'utilisateur. Un modèle de l'utilisateur a deux utilités : il définit les fondements pour la coopération entre l'utilisateur et le système informatique ; il conduit au développement d'outils d'évaluation ergonomique. Reprenons ces deux derniers points.

Le succès de la coopération entre un système informatique et un (ou plusieurs) utilisateur(s) repose sur des facteurs à la fois psychologiques et techniques. Du point de vue psychologique, la coopération est réglée par la nature du domaine d'application, par le niveau de compétence de l'utilisateur dans le domaine, par sa culture informatique, etc. Les facteurs techniques qui participent à la coopération relèvent de services généraux telles que l'aide procédurale, les explications et les suggestions. L'aide procédurale complète les informations factuelles puisqu'elle est capable de répondre à la question "Comment faire pour ... ?" en tenant compte de l'état actuel de l'interaction. Les explications justifient l'état de la situation (par exemple, indiquent la raison de l'invalidité d'une commande). Les suggestions visent à proposer à l'utilisateur d'autres méthodes pour atteindre un objectif donné.

Les outils d'aide à l'évaluation ergonomique ont été largement négligés jusque-là dans la spécification et la génération d'interfaces. Cette évaluation est actuellemnt effectuée "à la main". Elle exige du matériel coûteux (enregistrements vidéo), des spécialistes (mise en place des tests d'observation) et du temps (dépouillement des données observées) [Gould 88]. Une part de ce travail pourrait être pris en charge de manière automatique par le noyau d'exécution du générateur d'interface. Actuellement, les données observées automatiquement par les générateurs concernent les actions physiques de l'utilisateur. Ces informations sont utiles pour une analyse Keystroke [Card 83] de l'interface mais elles sont muettes sur le modèle mental de l'utilisateur. Les données observées devraient se situer à tous les niveaux d'abstraction des tâches utilisateur : les actions physiques, les tâches syntaxiques (c.-à-d. les tâches dûes au média informatique tels le déplacement et le changement de taille des fenêtres), et les tâches sémantiques (c.-à-d. les tâches relevant du domaine). Les observations à un haut niveau d'abstraction peuvent alimenter des outils d'évaluation ergonomique : par exemple analyser la fréquence d'utilisation des commandes pour raisonner sur l'importance ou l'utilité d'une

fonction ; détecter des suites de commandes similaires et en déduire les intentions de l'utilisateur.

5. Conclusion

Dans cet article, j'ai présenté une vision restreinte de l'interaction homme-machine : l'analyse de l'apport de l'ergonomie et de la psychologie cognitive et, pour le domaine de l'informatique, la description des architectures et des outils de construction des interfaces graphiques. L'interaction homme-machine fait intervenir d'autres disciplines, d'autres savoir-faire que je ne saurais analyser.

La compétence pluridisciplinaire est précisément l'une des exigences de ce domaine. Actuellement, les concepts, les modèles et les outils développés dans chaque discipline cohabitent davantage qu'ils ne s'intègrent et ne coopèrent. La prise de conscience de la situation s'affermit. Il nous faut maintenant persévérer dans le sens de l'ouverture des frontières entre les disciplines afin que la mise en commun du savoir d'équipes de formation complémentaire conduise à des idées fécondes assorties de solutions sures.

6. Références

[Amodeus 89] AMODEUS : *Assimilating Models Of Designers, Users and Systems* ; Technical Annex of the Esprit Basic Research Action 3066, Amodeus, 1989.

[Bass 88] L. BASS, E. HARDY, K. HOYT, R. LITTLE, R. SEACORD :*The Serpent run time architecture and dialogue model* ; Carnegie Mellon Technical Report, CMU/SEI-88-TR-6, January, 1988.

[Bennett 89] W.E. BENNETT, S. J. BOIES, J. D. GOULD, S. L. GREEN, C. F. WIECHA : *Transformations on a Dialog Tree : Rule Based Mapping of Content to Style* ; Proceedings ACM SIGGRAPH Symposium on User Interface Software and Technology, Williamsburg, Virginia, Nov, 1989, 67-75.

[Black 86] A. BLACK, N. HUTCHINSON, H. LEVY : *Object Structure in the Emerald System* ; Proceedings of the OOPSLA'86 conference, Portland, ACM Publ., Sept. , 1986, 78-86.

[Card 83] S. CARD, T. MORAN, A. NEWELL : *The Psychology of Human-Computer Interaction* ; ISBN 0-89859-243-7, Lawrence Erlbaum Associates Publ., 1983.

[Coutaz 87] J. COUTAZ : *PAC, an Implemention Model for Dialog Design* ; Interact'87, Stuttgart, September, 1987, 431-436.

[Coutaz 88] J. COUTAZ : *Interface Homme-Ordinateur : Conception et Réalisation* ; Thèse de doctorat d'état, Université Joseph Fourier de Grenoble, 1988.

[Coutaz 89] J. COUTAZ : *Architecture Models for Interactive Software : Failures and Trends* ; IFIP WG2.7 Working Conference Proceedings, Engineering for Human-Computer Interaction, Napa Valley, August, 1989.

[Coutaz 90] J. COUTAZ : *Interface Homme-Ordinateur : Conception et Réalisation* ; Dunod Publ., 1990.

[Delphia 87] DELPHIA : *PIXIA, une boîte à outils portable pour l'environnement Delphia-Prolog* ; Société Delphia, Seyssinet, Isère.

[Ellis 89] C.A. ELLIS, S.J. GIBBS, G.L. REIN : *Design and Use of a Group Editor* ; IFIP WG2.7 Working Conference Proceedings, Engineering for Human-Computer Interaction, Napa Valley, August, 1989.

[Fikes 71] R.E. FIKES, N.J. NILSSON : *STRIPS : A New Approach to the Application of Theorem Proving to Problem Solving* ; Artificial Intelligence, vol. 2, 1971, 189-208.

[Foley 84] J. D. FOLEY, A. VAN DAM : *Fundamentals of Interactive Computer Graphics* ; Addison Wesley Publ., 1984.

[Foley 88] J.D. FOLEY, W.C. KIM, S. KOVACEVIC, K. MURRAY : *The User Interface Design Environment* ; Technical Report GWU-IIST-88-04, Department of Electrical Engineering and Computer Science, The George Washington University, Washington D.C. 20052, January, 1988.

[Goldberg 84] A. GOLDBERG : *Smalltalk-80 : The Interactive Programming Environment* ; Addison-Wesley Publ., 1984.

[Gould 88] J.D. GOULD : *How to Design Usable Systems* ; Handbook of Human-Computer Interaction, M. Helander ed., Elsevier Science Publ. B. V. (North Holland), 1988, 757-789.

[Harvey 88] G. HARVEY : *Understanding Hypercard for Version 1.1* ; Sybex Books Publish., 1988.

[Hayes 85] P.J. HAYES, P. SZEKELY, R. LERNER : *Design Alternatives for User Interface Management Systems Based on Experience with Cousin* ; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April, 1985, 169-175.

[Hayes-Roth 79] B. HAYES-ROTH, F. HAYES-ROTH : *A Cognitive Model for Planning* ; Cognitive Science, 3, 1979, 275-310.

[Hill 87] R.D. HILL : *Event Response Systems-A Technique for Specifying Multi-thread Dialogues* ; Proceedings of the CHI+GI'87 Conference, The Association for Computing Machinery Publ., 1987, 241-248.

[Hutchins 86] E. L. HUTCHINS, J. D. HOLLAN, D. A. NORMAN : *Direct Manipulation Interfaces* ; User Centered System Design ; Lawrence Erlbaum Associates, Publishers, 1986, 87-124.

[Jacob 84] R.J.K. JACOB : *An Executable Specification Technique for Describing Human-Computer Interaction* ; Advances in Human Computer Interaction, H.R. Hartson, ed. Alex Publishing Co., 1984.

[Krakowiak 90] S. KRAKOWIAK, M. MEYSEMBOURG, M. RIVEILL, H. NGUYEN VAN, C. ROISIN : *Design and Implementation of an Object-Oriented Strongly Typed Language for Distributed Applications* ; to appear in Journal of Object Oriented Programming, 1990.

[Newell 86] A. NEWELL, S. CARD : *Straightening Out Softening UP : Response to Carroll and Campell* ; Human Computer Interaction, 2(3), 1986, 251-267.

[Nigay 90] L. NIGAY : *Modélisation des architectures logicielles des systèmes interactifs* ; Rapport de DEA, Institut National Polytechnique de Grenoble, juin, 1990.

[Norman 86] D. A. NORMAN, S. W. DRAPER : *User Centered System Design* ; Lawrence Erlbaum Associates Publ., 1986.

[Olsen 83] D.R. OLSEN, E.P DEMPSEY : *Syngraph : A Graphical User Interface Generator* ; Computer Graphics, July 1983, 43-50.

[Olsen 89] D.R. OLSEN : *A Programming Language Basis for User Interface Management* ; CHI'89 Conference proceedings, K. Bice, C. Lewis editors, ACM Press publish., April, 1989, 171-176.

[OSF 89] OSF : *OSF/Motif, Programmer's Reference Manual, Revision 1.0* ; Open Software Foundation, Eleven Cambridge Center, Cambridge, MA 02142, 1989.

[Petoud 89] I. PETOUD, Y. PIGNEUR : *An Automatic and Visual Approach for User Interface Design* ; IFIP WG2.7 Working Conference Proceedings, Engineering for Human-Computer Interaction, Napa Valley, August, 1989.

[Scapin 87] D.L. SCAPIN : *Guide ergonomique de conception des interfaces homme-machine* ; Rapport INRIA 77, octobre, 1987.

[Schmucker 86] K. SCHMUCKER : *MacApp : An Application Framework* ; Byte 11(8), 1986, 189-193.

[Schulert 85] A.J. SCHULERT, G.T. ROGERS, J.A. HAMILTON : *ADM - A Dialog Manager* ; Proceedings of the CHI'85 Conference, The Association for Computing Machinery Publ., April 1985, 177-183.

[Senay 90] H. SENAY, P. SUKAVIRIYA, L. MORAN : Planning for Automatic Help Generation ; in Engineering for Human-Computer Interaction, G. Cockton Ed., Elsevier Science Publ., 1990, 293-308.

[Shneiderman 87] B. SHNEIDERMAN : *Designing the User Interface : Strategies for Effective Human-Computer Interaction* ; Addison Wesley Publishing Company, 1987.

[Valdez 89] VALDEZ : *XVT, a Virtual Toolkit* ; Byte 14(3), 1989.

[Wasserman 85] A. WASSERMAN : *Extending State Transition Diagrams for the Specification of Human-Computer Interaction* ; IEEE Transactions on Software Engineering, 11(8), August, 1985.

[Webster 89] B.F. WEBSTER : *The NeXT Book* ; Addison Wesley publisher, 1989.

[Zimmerman 87] T.G. ZIMMERMAN, J. LANIER, C. BLANCHARD, S. BRYSON, Y. HARVILL : *A hand Gesture Device Interface* ; Proceedings of the CHI'87 Conference, The Association for Computing Machinery Publ., April 1987, 189-192.