2

LES MODELES GOMS ET KEYSTROKE

2.1. Introduction

Le Modèle du Processeur Humain nous présente une image simplifiée mais synthétique de notre structure mentale. Cette image repose sur des principes parmi lesquels nous retenons, pour les besoins de ce chapitre, le principe de rationalité. Selon ce principe, un individu s'efforce de s'adapter aux conditions de la tâche qu'il s'est fixé. Ceci signifie que le comportement est conditionné par l'environnement.

Cette hypothèse selon laquelle la complexité du comportement n'est pas due à la complexité interne de l'individu mais à celle de l'environnement, nous vient de Herbert Simon. Dans [Simon 84, p. 63-64], Simon nous fait observer que le cheminement sinueux de la fourmi sur le sable ne provient pas de la complexité de l'insecte mais bien du terrain tourmenté de la plage. Dans le domaine de l'édition de texte, nous faisons la même constatation. Par exemple, pour reproduire deux mots quelques lignes plus bas, un utilisateur d'Emacs va taper une suite complexe de caractères magiques (par exemple, "Ctrl-@ Esc-f Esc-f Ctrl-w Ctrl-y Ctrl-n Ctrl-n Ctrl-n Ctrl-y") alors que dans l'environnement Macwrite, il agira directement sur le document. La complexité apparente du comportement n'est plausible que lorsque le sujet est observé à un niveau d'abstraction élevé.

Ce chapitre présente GOMS [Card 83], un modèle de description du comportement, qui prend comme hypothèse le caractère adaptatif du sujet

humain. GOMS permet de modéliser le comportement à différents niveaux d'abstraction, depuis la tâche jusqu'aux actions physiques. La première partie du chapitre en présente les éléments généraux ; la seconde décrit le modèle Keystroke [Card 83], une application de GOMS à la représentation des actions physiques.

2.2. Le modèle GOMS

GOMS utilise comme point de départ le principe de rationalité du Modèle du Processeur Humain ; son apport essentiel est une structure formelle qui permet d'organiser le processus de conception. La méthode de conception induite par GOMS s'effectue selon deux axes : l'analyse de tâche (puisque c'est elle qui détermine le comportement) et l'évaluation prédictive du comportement de l'utilisateur dans l'accomplissement de cette tâche. Le paragraphe 2.2.1 suivant introduit les éléments formels du modèle. Il sera suivi d'une évaluation.

2.2.1. Les éléments du modèle

GOMS (Goal, Operator, Method, Selection) introduit quatre ensembles pour représenter l'activité cognitive d'un individu engagé dans la réalisation d'une tâche : les Butsles Opérateurs, les Méthodes et les règles de Sélection.

Un but est une structure symbolique qui définit un état recherché. Il lui est associé un ensemble de méthodes qui toutes conduisent à cet état. En cas d'échec, il constitue un point de reprise à partir duquel il est possible d'amorcer d'autres tentatives. Les buts sont organisés de manière hiérarchique : un but complexe est atteint lorsque plusieurs sous-buts sont satisfaits et ceci récursivement jusqu'à atteindre les buts élémentaires. Les buts élémentaires sont réalisés par l'exécution d'une suite d'opérateurs. Les buts forment donc une structure arborescente dont les feuilles sont des opérateurs.

Un opérateur est une action élémentaire dont l'exécution provoque un changement d'état (état mental de l'utilisateur et/ou état de l'environnement). L'analogie entre l'ensemble des opérateurs et le

répertoire des instructions d'une machine abstraite est assez claire : un opérateur se caractérise par des opérandes d'entrée et de sortie et par le temps nécessaire à son exécution. De même que l'unité d'exécution d'une instruction dépend du niveau d'abstraction de la machine abstraite, de même l'action définie par l'opérateur dépend du niveau de raffinement auquel la modélisation s'effectue. Lorsque l'analyse est fine, l'opérateur reflète des mécanismes psychologiques élémentaires (sensoriels, moteurs ou cognitifs). Lorsqu'elle s'effectue à un niveau d'abstraction élevé, les opérateurs sont des unités d'action spécifiques à l'environnement (par exemple les commandes du système).

Une méthode décrit le procédé qui permet d'atteindre un but. Elle s'exprime sous la forme d'une suite conditionnelle de buts et d'opérateurs où les conditions font référence au contenu de la mémoire à court terme et à l'état de l'environnement. Les méthodes représentent un savoir-faire : elles constituent la connaissance procédurale. Elles ne sont pas des plans d'action construits dynamiquement pendant l'accomplissement de la tâche. Elles sont le résultat de l'expérience acquise.

Une règle de Sélection exprime le choix d'une méthode lorsqu'il y a conflit, c'est-à-dire lorsque plusieurs méthodes conduisent au même but. Une règle est de la forme :

si <condition-sur-la-situation-actuelle-est-vraie>
alors utiliser la méthode M;

Par exemple, pour un éditeur pleine-page qui permet à la fois les commandes à la emacs et les commandes à la Macwrite, il existe plusieurs façons de déplacer le curseur : soit avec la souris, soit avec le clavier, soit par une combinaison des deux techniques. Le choix entre les deux méthodes peut s'exprimer en fonction de la distance entre la position actuelle du curseur et la localisation visée :

si le but à atteindre est placer le curseur au bas de la fenêtre
et
si la position actuelle du curseur est loin du bas de la fenêtre
alors utiliser la méthode M1;

si le but à atteindre est placer le curseur au bas de la fenêtre
et

si la position actuelle du curseur est près du bas de la fenêtre
alors utiliser la méthode M2;

avec, pour M1 :
prendre la souris;
déplacer la souris au point désiré;
sélectionner;

et, pour M2 :
tant que le curseur n'est pas sur la ligne désirée taper ctrl-n;
tant que le curseur n'est pas au point désiré taper esc-f;

M1 est adaptée aux grands déplacements tandis que M2 l'est pour les trajets locaux. Nous verrons dans la seconde partie de ce chapitre comment le modèle Keystroke confirme la validité de ces règles.

Parallèlement aux notions de but, opérateur, méthode et règle de sélection, la notion de niveau de modélisation permet de considérer GOMS comme un générateur d'une famille de modèles. Un niveau de modélisation se définit essentiellement par les temps d'exécution des opérateurs. Dans la mesure du possible, ces temps sont du même ordre de grandeur. Un modèle de niveau "x secondes" peut être raffiné en un modèle de niveau "y secondes" (avec y<x) en convertissant les opérateurs du niveau x en buts et en produisant des sous-buts jusqu'à ce que ces sous-buts soient atteints par des opérateurs de y secondes. A l'inverse, il est possible de constituer un modèle de niveau x par agrégation d'éléments du niveau y. De toute évidence, GOMS applique à la conception des interfaces, les méthodes de la programmation structurée par raffinement et par abstraction.

Card, Moran et Newell définissent quatre niveaux d'analyse : les niveaux tâche, fonctionnel, argument et physique. Le niveau tâche structure l'espace de travail en une hiérarchie de sous-tâches dont la nature dépend uniquement du domaine. Les éléments terminaux de la décomposition sont des tâches conceptuelles élémentaires. L'analyse fonctionnelle modélise les tâches élémentaires en termes de fonctions du système. A ce niveau de modélisation, l'accomplissement d'une tâche est décrit par une suite de fonctions. Le niveau argument précise, pour chaque fonction, sa réalisation par une suite de commandes. A ce niveau de modélisation, l'accomplissement d'une tâche est décrit par une suite de commandes. Le niveau physique décrit en termes d'actions physiques la spécification des

commandes. Keystroke présenté au paragraphe 2.3 est une illustration de cette dernière classe de modélisation.

2.2.2. Evaluation du modèle GOMS

Les points forts de GOMS concernent les aspects méthode de conception et technique d'évaluation mais les cognitivistes reprochent à GOMS de n'être qu'un modèle de performance.

2.2.2.1. L'apport de GOMS

GOMS véhicule une méthode de conception compatible avec celle que pratiquent les informaticiens. La modélisation d'une tâche peut être raffinée ou, au contraire, élaborée à partir de constituants élémentaires. L'informaticien procède de la même façon pour définir une hiérarchie de machines abstraites.

Le second aspect intéressant de GOMS est de fournir au concepteur un support formel pour des évaluations prédictives de performance. Une modélisation GOMS est une description mesurable du comportement de l'utilisateur. En effet, la description d'une tâche donnée définit la suite des opérateurs que l'utilisateur va employer pour la réaliser. Connaissant le temps d'exécution de chaque opérateur, il est possible de prédire le temps nécessaire à la réalisation de la tâche. Le temps prédit concerne le cas d'un utilisateur expert qui ne commet pas d'erreur. Cette remarque nous conduit à la présentation des limites de GOMS.

2.2.2.2. Les limites de GOMS

GOMS n'offre aucun support théorique d'aide à la structuration d'une tâche. Nous avons vu que l'informaticien retrouve dans GOMS le repère familier de l'analyse descendante et ascendante. Ici, le sujet d'analyse n'est pas un programme mais une tâche, notion que l'informaticien n'a pas l'habitude de manipuler. Or la réussite d'une analyse de tâche suppose la connaissance approfondie des mécanismes de représentation mentale. GOMS n'offre aucun support théorique dans ce sens : il est un modèle prédictif et quantitatif de performance.

Le caractère quantitatif d'un modèle donne une coloration "scientifique" à la description d'un phénomène. Malheureusement, dans le cas de GOMS, le phénomène observé est l'accomplissement de tâches de routine réalisées sans la moindre erreur. Or, l'erreur est inévitable et le traitement des erreurs est un cauchemar omniprésent y compris dans le cas simple des systèmes déterministes. Dans le cas du sujet humain, le traitement d'une erreur peut se voir comme la réalisation d'une tâche particulière. S'il s'agit d'une tâche de routine, alors il lui correspond un plan qui peut être greffé sur l'arbre de résolution. La question qui se pose maintenant est le lieu d'insertion du sous-plan. A ce problème, GOMS n'apporte aucun élément de réponse.

Tout comme le Modèle du Processeur Humain, GOMS est trop réducteur. Il ne décrit qu'un aspect limité des mécanismes cognitifs : celui de la résolution de problèmes déjà résolus! La planification hiérarchique introduite en intelligence artificielle [Sacerdoti 74] et reprise par GOMS, convient à des domaines où la dynamique et les faits imprévus n'interviennent pas. La réalité d'un individu et de son environnement est différente : le raisonnement linéaire de l'approche descendante s'applique lorsque les éléments d'un problème sont parfaitement maîtrisés. A la rencontre d'une difficulté, le recours opportuniste à l'observation de l'état de l'environnement déclenche des "déviations ascendantes" qui viennent se greffer au schéma descendant. Un courant important en intelligence artificielle illustré par les études sur la planification opportuniste [Hayes-Roth 79] s'intéresse à ces combinaisons de méthodes de résolution. Moins ambitieux mais plus proche des propos de l'interaction homme-machine est le modèle de Kieras et Polson [Kieras 85, Polson 85] qui, par extension des principes de GOMS, permet de prédire, pour des cas simples, les temps d'apprentissage de nouvelles méthodes ainsi que l'effet du transfert de connaissance.

Etant donné les conditions restrictives de GOMS, il n'est pas étonnant que Keystroke soit la seule classe des modèles GOMS applicable de manière réaliste [Card 83].

2.3. Le modèle Keystroke

Keystroke [Card 83] concerne les aspects syntaxiques et lexicaux de l'interaction. Ses éléments relèvent des actions physiques que l'utilisateur doit effectuer pour spécifier une commande.

2.3.1. Les éléments du modèle Keystroke

Le problème que se propose de résoudre Keystroke se formule ainsi :

Etant donné :

- une tâche (constituée éventuellement de plusieurs sous-tâches),
- le langage de commande du système,
- les paramètres caractéristiques des capacités motrices de l'utilisateur,
- les paramètres des temps de réponse du système, et
- la méthode de réalisation de la tâche,

prédire le temps d'exécution de cette tâche par un utilisateur expert.

Comme dans GOMS, Keystroke s'intéresse aux performances sans erreur. Contrairement à GOMS, Keystroke ne prédit pas de choix de méthode : la méthode est donnée. Comme seconde restriction par rapport à GOMS, Keystroke évalue le temps d'exécution, non pas le temps total d'accomplissement d'une tâche. Le temps d'accomplissement d'une tâche est la somme du temps d'acquisition et du temps d'exécution. Pendant l'acquisition, l'utilisateur construit une représentation mentale de la tâche. L'exécution est la réalisation effective physique de la tâche. Temps d'acquisition et temps d'exécution sont considérés indépendants. Keystroke s'intéresse uniquement au temps d'exécution.

Puisque Keystroke se situe au niveau lexical et que la méthode de réalisation de la tâche est donnée, les notions de buts et de règles de GOMS deviennent inutiles. Keystroke utilisent donc deux ensembles d'entités : les opérateurs et les méthodes.

2.3.1.1. Les opérateurs

Keystroke introduit six opérateurs pour décrire l'exécution d'une tâche élémentaire :

- K ("Keystroking", frappe de touches du clavier ou de la souris),
- P ("Pointing", désignation),
- H ("Homing", rapatriement de la main),

- D ("Drawing", action de dessiner), et
- M ("Mental activity", activité mentale).
- R ("Response time", temps de réponse du système)

Le temps d'exécution T_{exec} d'une tâche est tout simplement la somme des temps passés à exécuter chaque classe d'opérateurs.

$$T_{exec} = T_K + T_P + T_H + T_D + T_M + T_R$$

L'opérateur K représente la frappe d'une touche du clavier ou l'acte d'appuyer sur un bouton de la souris (ou tout autre dispositif de désignation). La détermination du temps t_K nécessaire à la frappe d'une touche est délicate. Ce temps dépend de la touche, du clavier et de l'aptitude de l'utilisateur. Une approximation acceptable consiste à soumettre à un test une population d'individus dont les aptitudes sont similaires puis de déterminer t_K en appliquant la formule suivante :

t_K = (durée totale des tests) / (nombre touches frappées sans erreur)

L'opérateur P représente le déplacement du curseur de la souris vers une cible. Le temps t_p nécessaire à ce placement est déterminé à partir de la variante de la loi de FittsCette variante nous indique que le temps T pour sélectionner une cible avec la souris s'exprime avec l'équation :

$$T = K_0 + I \log_2(D/L+0.5) \text{ secondes}$$

où

- K_0 est une constante qui tient compte du temps nécessaire pour ajuster la saisie initiale de la souris et pour appuyer sur un bouton de sélection. Les mesures expérimentales décrites par Card, Moran et Newell indiquent que pour la souris K_0 = 1.03 s [Card 83, p. 242],
- I est une constante évaluée à : I = 0.1 s,
- D est la distance entre la position actuelle de la souris et celle de la cible,
- L est la taille (par exemple la largeur) de la cible.

Dans ces conditions, t_p s'obtient en soustrayant de T le temps t_K nécessaire à la pression du bouton soit :

$$t_p = (K_0 - t_K) + I \log_2(D/L+0.5)$$

En remplaçant K_0, t_K (de l'ordre de 0.2 s, [Card 83] p.266) et I par leur valeur dans l'équation, on obtient pour t_p :

- borne inférieure : $t_p = 0.8$ s,
- borne supérieure : $t_p = 1.5$ s (avec D/L=128), et
- moyenne : $t_p = 1.1$ s, valeur cohérente avec plusieurs résultats expérimentaux (voir Card 83, p. 237, où la figure 7.4 montre, pour la souris, une valeur moyenne de placement de 1.29 s à laquelle il faut retrancher 0.2 s utilisé pour appuyer sur le bouton de la souris).

L'opérateur H représente les aspects pragmatiques de l'interaction homme-machine, en particulier le changement d'utilisation d'un dispositif physique. Les mesures effectuées par Card, Moran et Newell indiquent que le temps t_H nécessaire à la main pour changer de dispositif et se placer correctement sur le nouveau dispositif est :

$$t_H = 0.4 \text{ s.}$$

L'opérateur D représente l'utilisation de la souris pour construire un dessin sur l'écran. Le temps t_D nécessaire à la construction d'un dessin dépend des fonctions disponibles. Si le tracé de droite est la seule possibilité, alors t_D est une fonction linéaire du nombre n de segments tracés et de la somme l de leur longueur. En appliquant la technique des moindres carrés à un ensemble de données expérimentales recueillies sur des tests de tracé de segments de droite, Card, Moran et Newell ont obtenu :

$$t_D = 0.9\ n + 0.16\ l$$

D représente un ensemble d'opérateurs graphiques. Si l'on a besoin d'effectuer une étude précise, t_D devrait être raffinée pour tenir compte de chaque cas.

L'opérateur M représente l'activité mentale dont l'individu a besoin pour se préparer à exécuter un opérateur physique K, P, H ou D. Cette préparation revêt plusieurs formes et son temps de réalisation t_M peut de ce fait varier d'un cas à l'autre. Comme pour l'évaluation de t_D, Card,

Moran et Newell simplifient délibérément la situation et proposent une valeur unique pour t_M:

$t_M = 1.35$ s

L'opérateur R a trait aux temps de traitement des commandes par le système. t_R est le temps pendant lequel le système fait attendre l'utilisateur. Sachant que :

- n est le temps de traitement d'une commande par le système, et que
- t est le temps exploité par l'utilisateur pour exécuter un opérateur pendant le traitement de la commande, t_R vaut ceci :

$t_R = 0$ si $n \leq t$,
$t_R = n - t$ si $n > t$.

2.3.1.2. Codage des méthodes

Une méthode s'exprime sous la forme d'une suite d'opérateurs. Par exemple, si l'utilisateur doit entrer la commande unix **ls** au clavier, la méthode correspondante s'écrit :

M K[l] K[s] K[retour-chariot] ou, de manière plus condensée :
M 3K[l s retour-chariot].

Si maintenant, la commande **ls** est spécifiée avec la souris, la méthode devient :
H[souris] M P[souris] K[bouton-souris] H[clavier]

Les occurrences de M dans une méthode dépendent des opérateurs physiques et du savoir-faire de l'utilisateur. Le savoir-faire est une donnée spécifique à chaque utilisateur. Pour modéliser cette spécificité aux caractéristiques imprécises, Card, Moran et Newell utilisent des règles heuristiques. La définition de ces règles s'appuie sur la théorie suivante (confirmée par des observations expérimentales) : l'utilisateur tend à partitionner une méthode en sous-méthodes (c'est-à-dire en mnèmes) et à insérer une activité mentale entre chaque sous-méthode. Il se trouve que les mnèmes d'une méthode correspondent essentiellement aux unités syntaxiques d'une commande. Les règles suivantes identifient la décomposition d'une méthode en mnèmes :

Règle 0

Insérer M devant tous les K qui ne font pas partie de chaînes argument. Insérer M devant un P qui correspond à la désignation d'un nom de commande.

Règle 1

Supprimer M si l'opérateur qui suit M peut être anticipé avec l'opérateur qui précède M (par exemple, dans PMK, K représente l'acte d'appuyer sur un bouton de la souris. On considère que l'activité mentale pour K est anticipée dans P).

Règle 2

Si une chaîne de la forme MKMK.....MK, constitue un mnème, par exemple, le nom d'une commande, supprimer tous les M sauf le premier.

Règle 3

Si K est un symbole de terminaison redondant, supprimer le M qui le précède. K est un symbole de terminaison redondant s'il suit un autre symbole de terminaison, par exemple s'il termine la commande et s'il est précédé d'un symbole de fin d'argument.

Règle 4

Si K termine une constante (par exemple un nom de commande, non pas un argument), supprimer le M qui le précède. Si K termine une variable (par exemple, un argument), alors conserver M.

Ces règles déterminent de manière approximative les mnèmes d'une méthode. Elles constituent une base à laquelle il est possible d'ajouter des heuristiques plus précises pour, par exemple, tenir compte des différences entre les classes d'utilisateurs, différences qui, entre experts et novices, se traduisent précisément par la nature des mnèmes.

Voyons un exemple d'utilisation de ces règles de base.

2.3.2. Exemple d'application de Keystroke

Soit la tâche évoquée au paragraphe 2.2.1 qui consiste à déplacer le curseur vers le bas de la fenêtre. Cette tâche peut être effectuée selon deux méthodes M1 et M2. Le but est de prédire dans quelles conditions M1 est préférable à M2. Nous dirons que M1 est préférable à M2 si, pour un état initial identique de l'environnement, T_{M1}, le temps d'exécution de M1 est inférieur au temps d'exécution T_{M2} de M2.

Le codage d'une méthode s'effectue en trois étapes :

1) Codage de la méthode avec les opérateurs physiques uniquement.
2) Application de la règle 0 pour introduire M.
3) Application des règles 1, 2, 3 et 4 pour éliminer M.

Soient M1 et M2 exprimées de manière informelle :

Méthode M1 :
prendre la souris;
déplacer la souris au point désiré;
sélectionner;

Méthode M2 :
tant que le curseur n'est pas sur la ligne désirée taper ctrl-n;
tant que le curseur n'est pas sur le mot désiré taper esc-f;

M1 et M2 se transcrivent ainsi :

Méthode M1 :

- Etape 1 :
  H[souris] P[souris] K[bouton-souris] H[clavier]

- Etape 2, règle 0 : insertion de M devant les K et P
  H[souris] M P[souris] M K[bouton-souris] H[clavier]

- Etape 3, règle 1 : PMK = PK
  H[souris] M P[souris] K[bouton-souris] H[clavier]

- On obtient pour T_{M1}
  $T_{M1} = 2t_H + t_P + t_K + t_M$

Méthode M2 :

- Etape 1 : sachant que l'utilisateur a tapé m fois ctrl-n et p fois esc-f
  K[touche control] m {K[touche n]} p {K[touche esc] K[touche f]}

- Etape 2, règle 0 : insertion de M devant les K
  M K[touche control] m {M K[touche n]} p {M K[touche esc] M K[touche f]}

- Etape 3, règle 2 :
  M K[touche control] m {K[touche n]} M p {K[touche esc] K[touche f]}

- On obtient pour T_{M2}
  $T_{M2} = 2t_M + (m+ 2p +1)\ t_K$

M1 est préférable à M2 si :
  $T_{M1} < T_{M2}$
c'est-à-dire si :
  $2t_H + t_P + t_K + t_M < 2t_M + (m+ 2p +1)\ t_K$

Nous connaissons les valeurs moyennes de t_H, t_P, t_K et t_M soit en moyenne :

t_H = 0.4 s [Card 83, p. 263]
t_P = 1.1 s [Card 83, p. 234, p. 262],
t_K = 0.2 s [Card 83, p. 266],
t_M = 1.35 s [Card 83, p. 263].

Il est intéressant maintenant de déterminer les conditions sur m et p pour que l'utilisation de la souris devienne préférable à celle du clavier. Ces conditions s'expriment par la relation :

$2t_H + t_P + t_K + t_M < 2t_M + (m+ 2p +1)\ t_K$
$m+2p > (2t_H + t_P - t_M) / t_K$
$m+2p > 2.75$ (1)

Ce résultat indique que pour m=1 et p=1 (passer à la ligne suivante puis au mot suivant), l'utilisation de la souris est mieux adaptée. Cette conclusion

est un peu hâtive. En effet, dans la relation : $m+2p > (2t_H + t_P - t_M) /t_K$, le facteur influent est t_M (1.35 s). Si l'on considère le cas d'un utilisateur expert en Emacs, t_M est non seulement surévalué mais il est très probable que cet utilisateur ne marque pas de pause entre l'exécution de la commande "ligne-suivante" et celle de "mot-suivant". Si l'on considère que cette suite d'actions est une information compilée donc un mnème unique, t_M doit être éliminé de la relation. On obtient alors :

$$m+2p > (2t_H + t_P) /t_K \text{ soit :}$$
$$m+2p > 9.5 \qquad (2)$$

résultat compatible avec le comportement que l'auteur a observé sur des utilisateurs expérimentés : par exemple, si le déplacement doit s'effectuer sur la même ligne, il devient plus intéressant d'utiliser la souris au-delà de cinq mots. Si le déplacement est essentiellement vertical, le clavier reste l'organe privilégié dans les limites d'une dizaine de lignes.

La différence entre les conclusions (1) et (2) révèle deux difficultés : celle de définir une heuristique adaptée et celle de déterminer la valeur des paramètres. Ces aspects interviennent dans l'évaluation du modèle.

2.3.3. Evaluation de Keystroke

Keystroke hérite des caractéristiques de GOMS, de ses avantages comme de ses limitations.

2.3.3.1. Avantages

Keystroke est un outil d'analyse quantitative. Il permet de comparer ou de prédire les performances de l'utilisateur en fonction d'une syntaxe donnée. Par exemple, Keystroke confirme que les éditeurs pleine-page permettent d'accomplir plus rapidement les tâches usuelles d'édition de texte que les éditeurs à lignes ; il démontre que la sélection de texte s'effectue plus vite avec la souris qu'avec le manche à balai ou les touches du clavier.

Le second avantage de Keystroke est sa simplicité. La terminologie et la technique de modélisation sont compréhensibles pour un non-spécialiste en psychologie cognitive. Toutefois, la tentative de modélisation du

déplacement du curseur présentée au paragraphe 2.3.2 montre que cette simplicité n'est qu'apparente.

2.3.3.2. Précautions d'utilisation

Le placement de l'opérateur M est une première source de difficultés : la notion d'opération mentale présentée par Keystroke comme "une pause entre deux mnèmes", est trop imprécise. La nature des mnèmes varie avec le temps et les individus, d'où l'écart des résultats du paragraphe 2.3.2 entre l'heuristique "officielle" et la version "personnalisée". L'ajustement des règles générales au cas d'une étude particulière requiert des compétences que l'informaticien n'a généralement pas.

La seconde difficulté provient de l'imprécision des évaluations. Citons deux cas : le déplacement du curseur et le type de touche utilisée.

1) Dans la modélisation du paragraphe 2.3.2, les temps de réponse du système ont été ignorés. Ceci suppose que le système est suffisamment rapide pour ne pas faire attendre l'utilisateur. Dans le cas inverse, il faut être capable de déterminer t_R. Cette évaluation est certainement imprécise dans le cas d'un système en cours de conception.

2) Keystroke ne fait aucune distinction entre la saisie d'un texte ordinaire et la frappe des touches particulières comme les touches de contrôle. L'expérience montre qu'il existe des différences de temps d'exécution sensibles entre la frappe des touches usuelles et celle des touches particulières [John 87]. De même, Keystroke ne tient pas compte de l'existence de schémas ou de programmes moteurs précompilés.

Keystroke concentre l'analyse sur les aspects syntaxiques et lexicaux sans tenir compte de la problématique générale de l'utilisateur. Buxton dans [Buxton 82] fait observer qu'un gain de performance au niveau lexical peut s'estomper dans le cadre général de l'accomplissement d'une tâche. Le comportement de l'utilisateur n'est pas seulement dirigé par des considérations d'efficacité lexicale locale. Il l'est aussi par une logique sémantique globale. En conséquence, il est nécessaire que les efforts de conception soient effectués à différents niveaux de granularité et que les contradictions susceptibles de survenir entre les niveaux soient résolues par des compromis.

2.4. Résumé du chapitre

GOMS représente l'activité cognitive d'un individu engagé dans la réalisation d'une tâche de routine. Cette représentation s'effectue en termes de :

1) Buts organisés en hiérarchie.
2) Opérateurs (ou actions élémentaires).
3) Méthodes (ou procédures de réalisation d'un but).
4) Règles de sélection de méthode lorsque plusieurs d'entre elles permettent d'atteindre un même but.

Une modélisation GOMS peut s'effectuer à divers niveaux de raffinement : tâche, fonction, argument, physique. Keystroke est un exemplaire de la classe GOMS du niveau physique.

Keystroke et GOMS sont des modèles quantitatifs de performance. La modélisation est celle du comportement sans faute d'un utilisateur expert pour des tâches de routine. Il n'y a donc pas, dans ces modèles, d'éléments pour représenter l'apprentissage ni d'indication sur la nature ou la fréquence des erreurs.

GOMS constitue néanmoins une étape marquante vers la conception scientifique d'interfaces et Keystroke est un outil de mesure objective pour comparer, par exemple, les choix possibles syntaxiques et lexicaux d'un langage de commande.